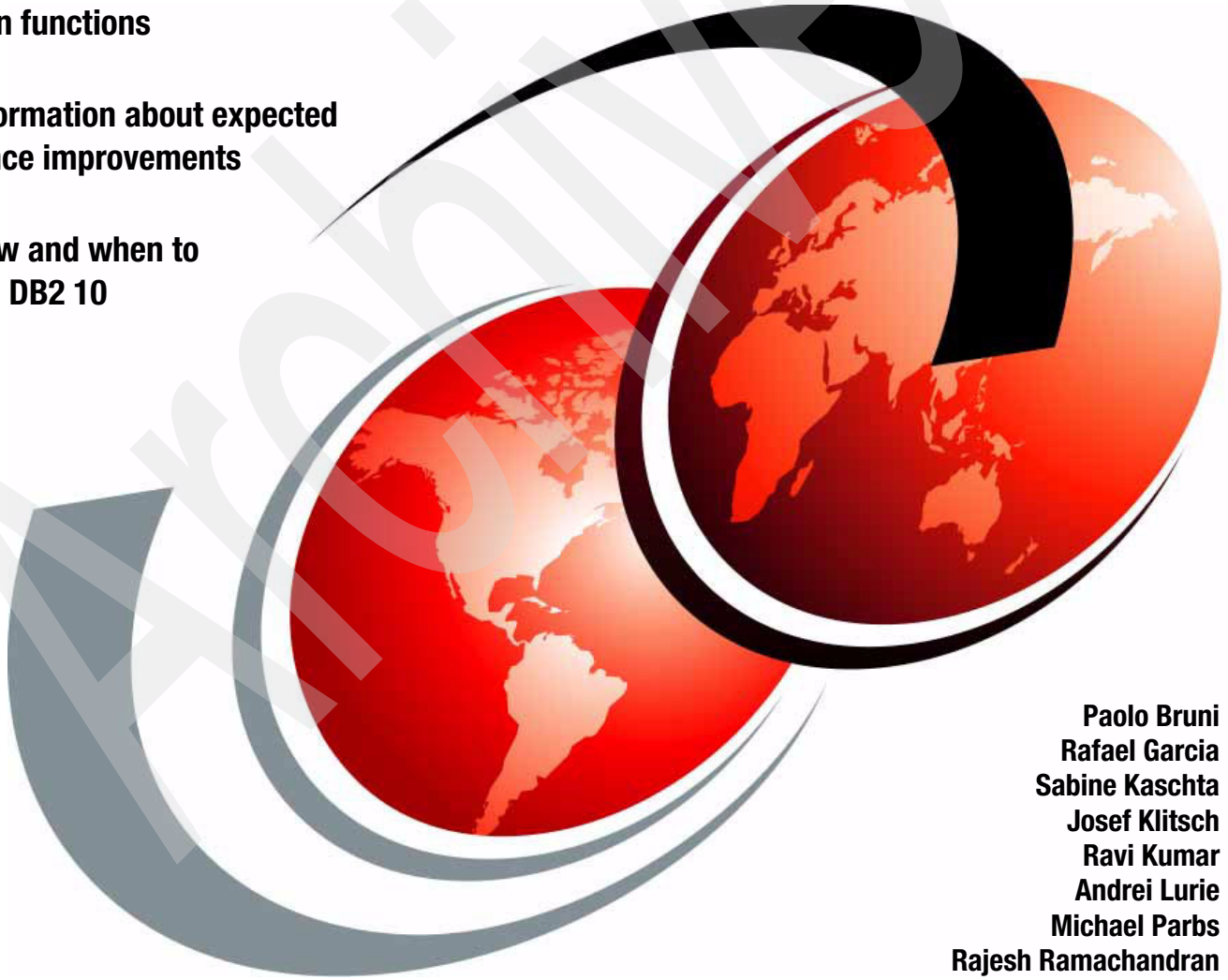
Information Management

IBM

DB2 10 for z/OS Technical Overview

- Explore the new system and application functions
- Obtain information about expected performance improvements
- Decide how and when to migrate to DB2 10



Paolo Bruni
Rafael Garcia
Sabine Kaschta
Josef Klitsch
Ravi Kumar
Andrei Lurie
Michael Parbs
Rajesh Ramachandran

Archive

Redbooks

ibm.com/redbooks

Archived

International Technical Support Organization

DB2 10 for z/OS Technical Overview

December 2010

Archived

SG24-7892-00

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

Archived

First Edition (December 2010)

This edition applies to IBM DB2 Version 10.1 for z/OS (program number 5605-DB2).

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

© Copyright International Business Machines Corporation 2010. 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 xiii

Examples xix

Tables xxv

Notices xxvii
Trademarks xxviii

Preface xxix
The team who wrote this book xxix
Acknowledgments xxxi
Now you can become a published author, too! xxxiii
Comments welcome xxxiii
Stay connected to IBM Redbooks publication xxxiii

Summary of changes xxxv
December 2010, First Edition xxxv
March 2011, First Update xxxv
December 2011, Second Update xxxvi
August 2013, Third Update xxxvi
July 2014, Fourth Update xxxvii

Chapter 1. DB2 10 for z/OS at a glance 1
1.1 Executive summary 2
1.2 Benefits of DB2 10 3
1.3 Subsystem enhancements 4
1.4 Application functions 7
1.5 Operation and performance enhancements 12

Part 1. Subsystem 1

Chapter 2. Synergy with System z 3
2.1 Synergy with System z in general 4
2.2 Synergy with IBM System z and z/OS 4
2.2.1 DBM1 64-bit memory usage and virtual storage constraint relief 4
2.2.2 ECSA virtual storage constraint relief 5
2.2.3 Increase of 64-bit memory efficiency 5
2.2.4 Improved CPU cache performance 7
2.2.5 HiperDispatch 8
2.2.6 XML virtual storage constraint relief 9
2.2.7 XML fragment validation 9
2.2.8 Improved DB2 startup times and DSMAX with z/OS V1R12 9
2.2.9 CPU measurement facility 9
2.3 Synergy with IBM zEnterprise System 10
2.4 Synergy with storage 13
2.4.1 Extended address volumes 13
2.4.2 More data set types supported on EAV 15
2.4.3 Dynamic volume expansion feature 16
2.4.4 SMS data set separation by volume 16

© Copyright IBM Corp. 2010. All rights reserved.

2.4.5 High Performance FICON . . . 17
2.4.6 IBM System Storage DS8800 . . . 20
2.4.7 DFSMS support for solid-state drives . . . 21
2.4.8 Easy Tier technology . . . 22
2.4.9 Data set recovery of moved and deleted data sets . . . 22
2.4.10 Synergy with FlashCopy . . . 23
2.4.11 DB2 catalog and directory now SMS managed . . . 24
2.5 z/OS Security Server . . . 25
2.5.1 RACF password phrases . . . 25
2.5.2 RACF identity propagation . . . 26
2.6 Synergy with TCP/IP . . . 26
2.6.1 z/OS V1R10 and IPv6 support . . . 27
2.6.2 z/OS UNIX System Services named pipe support in FTP . . . 27
2.6.3 IPSec encryption . . . 28
2.6.4 SSL encryption . . . 30
2.7 WLM . . . 31
2.7.1 DSN_WLM_APPLENV stored procedure . . . 31
2.7.2 Classification of DRDA workloads using DB2 client information . . . 32
2.7.3 WLM blocked workload support . . . 35
2.7.4 Extend number of WLM reporting classes to 2,048 . . . 38
2.7.5 Support for enhanced WLM routing algorithms . . . 38
2.8 Using RMF for zIIP reporting and monitoring . . . 38
2.8.1 DRDA workloads . . . 39
2.8.2 Batch workloads . . . 39
2.9 Warehousing on System z . . . 42
2.9.1 Cognos on System z . . . 42
2.10 Data encryption . . . 43
2.10.1 IBM TotalStorage for encryption on disk and tape . . . 43
2.11 IBM WebSphere DataPower . . . 44
2.12 Additional zIIP and zAAP eligibility . . . 45
2.12.1 DB2 10 parallelism enhancements . . . 46
2.12.2 DB2 10 RUNSTATS utility . . . 46
2.12.3 DB2 10 buffer pool prefetch and deferred write activities . . . 46
2.12.4 z/OS sort utility (DFSORT) . . . 49
2.12.5 DRDA workloads . . . 49
2.12.6 zAAP on zIIP . . . 49
2.12.7 z/OS XML system services . . . 49

Chapter 3. Scalability . . . 51
3.1 Virtual storage relief . . . 52
3.1.1 Support for full 64-bit run time . . . 52
3.1.2 64-bit support for the z/OS ODBC driver . . . 53
3.2 Reduction in latch contention . . . 54
3.3 Reduction in catalog contention . . . 55
3.4 Increased number of packages in SPT01 . . . 55
3.5 The WORKFILE database enhancements . . . 56
3.5.1 Support for spanned work file records . . . 56
3.5.2 In-memory work file enhancements for performance . . . 57
3.5.3 The CREATE TABLESPACE statement . . . 57
3.5.4 Installation changes . . . 58
3.6 Elimination of UTSERIAL for DB2 utilities . . . 61
3.7 Support for Extended Address Volumes . . . 62
3.8 Shutdown and restart times, and DSMAX . . . 64

3.9 Compression of SMF records 64

Chapter 4. Availability 67
4.1 Online schema enhancements 68
4.1.1 UTS with DB2 9: Background information. 68
4.1.2 UTS with DB2 10: The ALTER options 69
4.1.3 Pending definition changes 72
4.1.4 Online schema changes in detail 73
4.1.5 Materialization of pending definition changes 86
4.1.6 Impact on immediate options and DROP PENDING CHANGES 89
4.1.7 UTS ALTER for MEMBER CLUSTER. 92
4.1.8 Utilities support for online schema changes 93
4.2 Autonomic checkpoint 96
4.3 Dynamically adding an active log data set 98
4.4 Preemptible backout 100
4.5 Support for rotating partitions 100
4.6 Compress on insert 103
4.6.1 DSN1COMP considerations 105
4.6.2 Checking whether the data in a table space is compressed 106
4.6.3 Data is not compressed 106
4.7 Long-running reader warning message 106
4.8 Online REORG enhancements 107
4.9 Increased availability for CHECK utilities 108

Chapter 5. Data sharing 109
5.1 Subgroup attach name 110
5.2 Delete data sharing member 112
5.3 Buffer pool scan avoidance 114
5.4 Universal table space support for MEMBER CLUSTER 114
5.5 Restart light handles DDF indoubt units of recovery 117
5.6 Auto rebuild coupling facility lock structure on long IRLM waits during restart 118
5.7 Log record sequence number spin avoidance for inserts to the same page. 118
5.8 IFCID 359 for index split 119
5.9 Avoid cross invalidations 119
5.10 Recent DB2 9 enhancements 120
5.10.1 Random group attach DSNZPARM 120
5.10.2 Automatic GRECP recovery functions 120
5.10.3 The -ACCESS DATABASE command enhancements 121
5.10.4 Reduction in forced log writes 122

Part 2. Application functions 123

Chapter 6. SQL 125
6.1 Enhanced support for SQL scalar functions 126
6.1.1 SQL scalar functions syntax changes 128
6.1.2 Examples of SQL scalar functions 136
6.1.3 SQL scalar function versioning 138
6.1.4 Deploying non-inline SQL scalar functions 140
6.1.5 ALTER actions for the SQL scalar functions. 141
6.2 Support for SQL table functions 144
6.2.1 SQL table functions syntax changes. 144
6.2.2 Examples of CREATE and ALTER SQL table functions. 147
6.3 Enhanced support for native SQL procedures 148
6.4 Extended support for implicit casting 148

6.4.1 Examples of implicit casting . . . 150
6.4.2 Rules for result data types . . . 152
6.4.3 Function resolution . . . 152
6.5 Support for datetime constants . . . 154
6.6 Variable timestamp precision . . . 156
6.6.1 String representation of timestamp values . . . 157
6.6.2 Timestamp assignment and comparison . . . 158
6.6.3 Scalar function changes . . . 161
6.6.4 Application programming . . . 163
6.7 Support for TIMESTAMP WITH TIME ZONE . . . 166
6.7.1 Examples of TIMESTAMP WITH TIME ZONE . . . 167
6.7.2 String representation of TIMESTAMP WITH TIME ZONE values . . . 169
6.7.3 Determination of the implicit time zone . . . 170
6.7.4 TIMESTAMP WITH TIME ZONE assignment and comparison . . . 171
6.7.5 Rules for result data type with TIMESTAMP WITH TIME ZONE operands . . . 173
6.7.6 CURRENT TIMESTAMP WITH TIME ZONE special register . . . 174
6.7.7 Scalar function changes . . . 177
6.7.8 Statements changes . . . 180
6.7.9 Application programming . . . 182
6.8 Support for OLAP aggregation specification . . . 184

Chapter 7. Application enablement . . . 203
7.1 Support for temporal tables and versioning . . . 204
7.1.1 System period . . . 205
7.1.2 Application period . . . 215
7.2 Access plan stability and instance-based statement hints . . . 220
7.2.1 Access path repository . . . 221
7.2.2 BIND QUERY and FREE QUERY DB2 commands . . . 222
7.2.3 Access plan stability . . . 223
7.2.4 DB2 9 access plan stability support . . . 224
7.2.5 DB2 10 access plan stability support . . . 229
7.2.6 Instance-based statement hints . . . 230
7.3 Addition of extended indicator variables . . . 230
7.4 New Universal Language Interface program (DSNULI) . . . 235
7.5 Access to currently committed data . . . 238
7.6 EXPLAIN MODE special register to explain dynamic SQL . . . 240
7.7 Save LASTUSED information for packages . . . 242

Chapter 8. XML . . . 243
8.1 DB2 9 XML additional functions . . . 244
8.1.1 The XMLTABLE function . . . 244
8.1.2 The XMLCAST specification . . . 250
8.1.3 XML index for XML joins . . . 251
8.1.4 Index enhancements . . . 252
8.1.5 XML index use by queries with XMLTABLE . . . 254
8.1.6 XPath scan improvement . . . 257
8.1.7 XPath functions . . . 258
8.2 XML type modifier . . . 258
8.3 XML schema validation . . . 262
8.3.1 Enhancements to XML schema validation . . . 263
8.3.2 Determining whether an XML document has been validated . . . 268
8.4 XML consistency checking with the CHECK DATA utility . . . 269
8.5 Support for multiple versions of XML documents . . . 273

8.5.1 XML versions 274
8.5.2 Storage structure for XML data with versions 276
8.6 Support for updating part of an XML document 280
8.6.1 Updates to entire XML documents 280
8.6.2 Partial updates of XML documents 282
8.7 Support for binary XML 286
8.8 Support for XML date and time 290
8.8.1 Enhancements for XML date and time support 290
8.8.2 XML date and time support 291
8.9 XML in native SQL stored procedures and UDFs 300
8.9.1 Enhancements to native SQL stored procedures 301
8.9.2 Enhancements to user defined SQL scalar and table functions 302
8.9.3 Decompose to multiple tables with a native SQL procedure 305
8.10 Support for DEFINE NO for LOBs and XML 306
8.10.1 IMPDSDEF subsystem parameter 307
8.10.2 Usage reference 307
8.11 LOB and XML data streaming 308

Chapter 9. Connectivity and administration routines 309
9.1 DDF availability 310
9.1.1 Online communications database changes 310
9.1.2 Online DDF location alias name changes 311
9.1.3 Domain name is now optional 315
9.1.4 Acceptable period for honoring cancel requests 315
9.1.5 Sysplex balancing using SYSIBM.IPLIST 316
9.1.6 Message-based correlation with remote IPv6 clients 316
9.2 Monitoring and controlling connections and threads at the server 317
9.2.1 Create the tables 318
9.2.2 Insert a row in DSN_PROFILE_TABLE 318
9.2.3 Insert a row in DSN_PROFILE_ATTRIBUTES 319
9.2.4 Activate profiles 321
9.2.5 Deactivate profiles 321
9.2.6 Activating a subset of profiles 321
9.3 JDBC Type 2 driver performance enhancements 321
9.3.1 Limited block fetch 322
9.3.2 Conditions for limited block fetch 324
9.4 High performance DBAT 324
9.4.1 High performance DBAT to reduce CPU usage 324
9.4.2 Dynamic switching to RELEASE(COMMIT) 325
9.4.3 Requirements 325
9.5 Use of RELEASE(DEALLOCATE) in Java applications 325
9.6 Support for 64-bit ODBC driver 327
9.7 DRDA support of Unicode encoding for system code pages 328
9.8 Return to client result sets 328
9.9 DB2-supplied stored procedures 329
9.9.1 Administrative task scheduler 329
9.9.2 Administration enablement 331
9.9.3 DB2 statistics routines 332

Part 3. Operation and performance 335

Chapter 10. Security 337
10.1 Policy-based audit capability 338
10.1.1 Audit policies 338

10.1.2 Authorization . . . 348
10.1.3 Creating audit reports . . . 349
10.1.4 Policy-based SQL statement auditing for tables . . . 349
10.1.5 Summary . . . 354
10.2 More granular system authorities and privileges . . . 354
10.2.1 Separation of duties . . . 356
10.2.2 Least privilege . . . 356
10.2.3 Grant and revoke system privilege changes . . . 356
10.2.4 Catalog changes . . . 359
10.2.5 SECADM authority . . . 359
10.2.6 System DBADM authority . . . 363
10.2.7 DATAACCESS authority . . . 367
10.2.8 ACCESSCTRL authority . . . 369
10.2.9 Authorities for SQL tuning . . . 371
10.2.10 The CREATE_SECURE_OBJECT system privilege . . . 376
10.2.11 Using RACF profiles to manage DB2 10 authorities . . . 377
10.2.12 Separating SECADM authority from SYSADM and SYSCTRL authority . . . 377
10.2.13 Minimize need for SYSADM authorities . . . 380
10.3 System-defined routines . . . 380
10.3.1 Installing DB2-supplied system-defined routines . . . 381
10.3.2 Define your own system-defined routines . . . 381
10.3.3 Mark user-provided SQL table function as system defined . . . 381
10.4 The REVOKE dependent privilege clause . . . 383
10.4.1 Revoke statement syntax . . . 383
10.4.2 Revoke dependent privileges system default . . . 383
10.5 Support for row and column access control . . . 384
10.5.1 Authorization . . . 384
10.5.2 New terminology . . . 385
10.5.3 Object types for row and column based policy definition . . . 386
10.5.4 SQL DDL for managing new access controls and objects . . . 387
10.5.5 Built-in functions . . . 388
10.5.6 Catalog tables and row and column access control . . . 389
10.5.7 Sample customer table used in examples . . . 390
10.5.8 Row access control . . . 391
10.5.9 Row permissions . . . 393
10.5.10 Column access control . . . 395
10.5.11 Column masks . . . 396
10.5.12 Row permissions and column masks in interaction . . . 398
10.5.13 Application design considerations . . . 400
10.5.14 Operational considerations . . . 400
10.5.15 Performance considerations . . . 401
10.5.16 Catalog changes for access control . . . 406
10.5.17 Added and changed IFCIDs . . . 411
10.6 Support for z/OS security features . . . 411
10.6.1 z/OS Security Server password phrase . . . 411
10.6.2 z/OS Security Server identity propagation . . . 416

Chapter 11. Utilities . . . 425
11.1 Support FlashCopy enhancements . . . 426
11.1.1 Getting started with FlashCopy image copies with the COPY utility . . . 426
11.1.2 FlashCopy image copies and utilities other than COPY . . . 437
11.1.3 Requirements and restrictions for using FlashCopy image copies . . . 444
11.2 Autonomic statistics . . . 445

11.2.1 Using RUNSTATS profiles . . . 447
11.2.2 Updating RUNSTATS profiles . . . 447
11.2.3 Deleting RUNSTATS profiles . . . 447
11.2.4 Combining autonomic and manual statistics maintenance . . . 448
11.3 RECOVER with BACKOUT YES . . . 448
11.4 Online REORG enhancements . . . 452
11.4.1 Improvements to online REORG for base tables spaces with LOBs . . . 452
11.4.2 REORG SHRLEVEL options for LOB table spaces . . . 458
11.4.3 Improved usability of REORG of disjoint partition ranges . . . 458
11.4.4 Cancelling blocking claimers with REORG FORCE . . . 460
11.5 Increased availability for CHECK utilities . . . 464
11.6 IBM DB2 Sort for z/OS . . . 467
11.7 UTSERIAL elimination . . . 468
11.8 REPORT utility output improvement . . . 468

Chapter 12. Installation and migration . . . 471
12.1 Planning for migration . . . 472
12.1.1 DB2 10 pre-migration health check job . . . 473
12.1.2 Fallback SPE and maintenance . . . 474
12.1.3 Partitioned data set extended support . . . 477
12.1.4 Convert bootstrap data set to expanded format . . . 477
12.1.5 Plans and packages . . . 478
12.2 Some release incompatibilities . . . 479
12.2.1 CHAR and VARCHAR formatting for decimal data . . . 479
12.2.2 Fall back restriction for native SQL procedures . . . 480
12.2.3 Fall back restriction for index on expression . . . 480
12.3 DB2 10 product packaging . . . 480
12.3.1 Removed features and functions . . . 481
12.3.2 Deprecated features and functions . . . 483
12.3.3 Base engine and features . . . 484
12.4 Command changes . . . 486
12.5 Catalog changes . . . 489
12.5.1 SMS-managed DB2 catalog and directory data sets . . . 490
12.5.2 CLOB and BLOB columns added to the catalog . . . 490
12.5.3 Reduced catalog contention . . . 491
12.5.4 Converting catalog and directory table spaces to partition-by-growth . . . 492
12.5.5 Added catalog objects . . . 496
12.6 Implications of DB2 catalog restructure . . . 500
12.7 DSNZPARM change summary . . . 501
12.8 EXPLAIN tables in DB2 10 . . . 507
12.8.1 EXPLAIN table changes . . . 508
12.8.2 Tools to help you convert to new format and Unicode . . . 509
12.8.3 Converting the EXPLAIN table to new format and Unicode . . . 514
12.9 SMS-managed DB2 catalog . . . 516
12.10 Skip level migration . . . 517
12.11 Fallback . . . 519
12.11.1 Implication to catalog image copy job . . . 519
12.11.2 Frozen objects . . . 520
12.12 Improvements to DB2 installation and samples . . . 520
12.12.1 Installation pop-up panel DSNTIPSV . . . 520
12.12.2 Job DSNTIJXZ . . . 520
12.12.3 Installation verification procedure . . . 522
12.12.4 Sample for XML . . . 522

12.13 Simplified installation and configuration of DB2-supplied routines 523
12.13.1 Deploying the DB2-supplied routines when installing DB2 10 for z/OS 528
12.13.2 Validating deployment of DB2-supplied routines 529
12.14 Eliminating DDF private protocol. 529
12.15 Precompiler NEWFUN option 531

Chapter 13. Performance 533
13.1 Performance expectations 535
13.2 Improved optimization techniques. 536
13.2.1 Safe query optimization 537
13.2.2 RID pool enhancements 537
13.2.3 Range-list index scan 539
13.2.4 IN-LIST enhancements 540
13.2.5 Aggressive merge for views and table expressions 542
13.2.6 Improvements to predicate processing 544
13.2.7 Sort enhancements 544
13.3 Dynamic prefetch enhancements 545
13.3.1 Index scans using list prefetch 545
13.3.2 Row level sequential detection 546
13.3.3 Progressive prefetch quantity 548
13.4 DDF enhancements 548
13.4.1 The RELEASE(DEALLOCATE) BIND option 548
13.4.2 Miscellaneous DDF performance improvements 551
13.5 Dynamic statement cache enhancements 553
13.6 INSERT performance improvement 557
13.6.1 I/O parallelism for index updates. 558
13.6.2 Sequential inserts into the middle of a clustering index 559
13.7 Referential integrity checking improvement 560
13.8 Buffer pool enhancements 560
13.8.1 Buffer storage allocation 561
13.8.2 In-memory table spaces and indexes 561
13.8.3 Reduce latch contention 563
13.9 Work file enhancements 564
13.10 Support for z/OS enqueue management 565
13.11 LOB enhancements 565
13.11.1 LOAD and UNLOAD with spanned records 565
13.11.2 File reference variable enhancement for 0 length LOBs. 567
13.11.3 Streaming LOBs and XML between DDF and DBM1 568
13.11.4 Inline LOBs 569
13.12 Utility BSAM enhancements for extended format data sets 571
13.13 Performance enhancements for local Java and ODBC applications. 572
13.14 Logging enhancements. 574
13.14.1 Long term page fix log buffers 574
13.14.2 LOG I/O enhancements 575
13.14.3 Log latch contention reduction 575
13.15 Hash access 575
13.15.1 The hashing definitions 579
13.15.2 Using hash access 580
13.15.3 Monitoring the performance of hash access tables. 584
13.15.4 New SQLCODEs to support hash access. 585
13.16 Additional non-key columns in a unique index 586
13.17 DB2 support for solid state drive 590
13.18 Extended support for the SQL procedural language 591

13.19 Preemptable backout . . . 592
13.20 Eliminate mass delete locks for universal table spaces . . . 592
13.21 Parallelism enhancements . . . 593
13.21.1 Remove some restrictions. . . . 593
13.21.2 Improve the effectiveness of parallelism . . . 594
13.21.3 Straw model for workload distribution . . . 595
13.21.4 Sort merge join improvements . . . 597
13.22 Online performance buffers in 64-bit common . . . 597
13.23 Enhanced instrumentation . . . 598
13.23.1 One minute statistics trace interval . . . 598
13.23.2 IFCID 359 for index leaf page split . . . 598
13.23.3 Separate DB2 latch and transaction lock in Accounting class 8 . . . 599
13.23.4 Storage statistics for DIST address space . . . 599
13.23.5 Accounting: zIIP SECP values . . . 600
13.23.6 Package accounting information with rollup . . . 601
13.23.7 DRDA remote location statistics detail . . . 601
13.24 Enhanced monitoring support . . . 603
13.24.1 Unique statement identifier . . . 604
13.24.2 New monitor class 29 for statement detail level monitoring . . . 605
13.24.3 System level monitoring . . . 605

Part 4. Appendixes . . . 613

Appendix A. Information about IFCID changes . . . 615
A.1 IFCID 002: Dynamic statement cache . . . 616
A.2 IFCID 002 - Currently committed data . . . 616
A.3 IFCID 013 and IFCID 014 . . . 617
A.4 IFCID 106 . . . 618
A.5 IFCID 225 . . . 618
A.6 IFCID 267 . . . 621
A.7 IFCID 316 . . . 621
A.8 IFCID 357 . . . 622
A.9 IFCID 358 . . . 622
A.10 IFCID 359 . . . 622
A.11 IFCID 360 . . . 623
A.12 IFCID 363 . . . 623

Appendix B. Summary of relevant maintenance . . . 627
B.1 DB2 APARs . . . 628
B.2 z/OS APARs . . . 634
B.3 OMEGAMON PE APARs . . . 635

Abbreviations and acronyms . . . 637

Related publications . . . 641
IBM Redbooks publication . . . 641
Other publications . . . 641
Online resources . . . 642
How to get Redbooks publications . . . 643
Help from IBM . . . 643

Index . . . 645

Archive

Archived

Figures

2-1 DB2 9 and DB2 10 VSCR in the DBM1 address space 5
2-2 Address translation for 4 KB pages 6
2-3 Address translation for 1 MB pages 7
2-4 Memory and CPU cache latencies for z10 8
2-5 IBM zEnterprise System components 10
2-6 IBM zEnterprise System: Capacity and scalability 11
2-7 zEnterprise CPU reduction with DB2 9 13
2-8 EAV breaking the limit 14
2-9 Overview DS8000 EAV support 14
2-10 EAV support for ICF catalogs and VVDS 16
2-11 zHPF performance 17
2-12 zHPF link protocol comparison 18
2-13 DS8000 versus DS8300 for DB2 sequential scan 20
2-14 Log writes throughput comparisons 21
2-15 FLASHCOPY NO COPY utility accounting report 24
2-16 FLASHCOPY YES COPY utility accounting report 24
2-17 UNIX System Services file system information for a UNIX System Services named pipe 27
2-18 How FTP access to UNIX named pipes works 28
2-19 z/OS IPSec overview 29
2-20 IPSEC zIIP eligibility 30
2-21 z/OS V1R12 AT-TLS in-memory encrypt / decrypt improvement 31
2-22 DSN_WLM_APPLENV output 32
2-23 WLM classification DRDA work based on program name 33
2-24 SDSF enclave display of DRDA request 34
2-25 SDSF enclave information for client program ZSYNERGY 34
2-26 DB2 display thread command showing DB2 client information 35
2-27 Blocked workload RMF CPU activity report 37
2-28 Blocked workload RMF workload activity report 37
2-29 RMF workload activity report for RMF report class ZSYNERGY 39
2-30 WLM classification for batch job 40
2-31 SDSF display active batch WLM classification 40
2-32 RMF workload activity report for the RUNSTATS report class 41
2-33 DB2 RUNSTATS utility accounting report 42
2-34 WebSphere DataPower DRDA capabilities 44
2-35 Specialty engines applicability 45
2-36 Comparison of zAAP and zIIP 46
2-37 OMEGAMON® PE statistics report DBM1 zIIP usage 47
2-38 WLM report class assignment for the MSTR address space 48
2-39 SDSF display active panel to verify MSTR report class assignment 48
2-40 RMF workload activity report zIIP usage for buffer pool prefetches 48
2-41 z/OS XML system services zIIP and zAAP processing flow 50
3-1 Updated installation panel DSNTIP9 58
4-1 Possible table space type conversions 70
4-2 Possible online schema changes for table spaces and indexes 71
4-3 ALTER TABLESPACE ... BUFFERPOOL statement and resulting SQL code 75
4-4 SYSIBM.SYSPENDINGDDL after ALTER 76
4-5 -647 SQL code after ALTER TABLESPACE ... BUFFERPOOL 77

© Copyright IBM Corp. 2010. All rights reserved.

4-6 ALTER TABLESPACE ... SEGSIZE decreased 79
4-7 ALTER TABLE ADD ORGANIZE BY HASH 81
4-8 ALTER TABLE organization-clause 81
4-9 Partition-by-growth to hash 82
4-10 REORGed HASH SPACE 84
4-11 DROP ORGANIZATION 85
4-12 REORG TABLESPACE messages for materialization of pending changes 86
4-13 SQLCODE +610 message 89
4-14 One wrapped row showing pending change in SYSPENDINGDDL 90
4-15 SQLCODE for normally immediate change due to pending change 90
4-16 Recover after alter before materialization 93
4-17 Recover to current after materialization of pending definition changes 93
4-18 Effects of REPAIR SET NOAREORPEND 96
4-19 -DIS LOG OUTPUT for CHECKTYPE=BOTH 98
4-20 -SET LOG syntax 98
4-21 ROTATE 3 TO LAST sample 101
4-22 SELECT to identify right physical partition for ROTATE 102
4-23 ROTATE 4 TO LAST 102
4-24 -DIS DB after two ROTATE table space executions 102
4-25 LIMITKEY, PARTITION, and LOGICAL_PART after second rotate 103
4-26 Error message you get when trying to rotate the last partition 103
4-27 Message DSNU241I compression dictionary build 104
4-28 Compression dictionary pages spread over table space 105
5-1 MEMBER CLUSTER with RECOVER 117
5-2 ACCESS DATABASE command 122
6-1 CREATE FUNCTION (SQL scalar) syntax diagram: Main 128
6-2 CREATE FUNCTION (SQL scalar) syntax diagram: parameter-declaration 129
6-3 CREATE FUNCTION (SQL scalar) syntax diagram: SQL-routine-body 129
6-4 ALTER FUNCTION (SQL scalar) syntax diagram: Main 133
6-5 CREATE FUNCTION (SQL table) syntax diagram: Main 144
6-6 CREATE FUNCTION (SQL table): Options 145
6-7 CREATE FUNCTION (SQL table) syntax: SQL-routine-body 145
6-8 RETURN SQL control statement 146
6-9 ALTER FUNCTION (SQL table): Main 146
6-10 Enhanced assignment (SET) statement 148
6-11 Stored length of the timestamp based on its precision 157
6-12 Described length of the timestamp based on its precision 157
6-13 TIMESTAMP_STRUCT used for SQL_C_TYPE_TIMESTAMP 165
6-14 TIMESTAMP data types supported by DB2 167
6-15 Stored length of the TIMESTAMP WITH TIME ZONE based on its precision 168
6-16 Described (external) length of the timestamp based on its precision 169
6-17 Syntax for time zone specific expressions 176
6-18 TIMESTAMP_TZ syntax diagram 178
6-19 Syntax diagram for SET SESSION TIME ZONE 181
6-20 Scalar function processing 185
6-21 Aggregate function processing 185
6-22 OLAP specification processing 186
6-23 Window-partition-clause syntax 188
6-24 Window-ordering-clause syntax 189
6-25 Window aggregation-group-clause 195
6-26 Aggregation specification syntax 201
7-1 Access path repository 221
7-2 BI ND QUERY syntax 222

7-3 FREE QUERY syntax 223
7-4 REBIND (TRIGGER) PACKAGE...PLANMGMT option 226
7-5 REBIND (TRIGGER) PACKAGE...PLANMGMT(BASIC) 226
7-6 REBIND (TRIGGER) PACKAGE...PLANMGMT(EXTENDED) 227
7-7 REBIND (TRIGGER) PACKAGE...SWITCH (1 of 2) 228
7-8 REBIND (TRIGGER) PACKAGE...SWITCH (2 of 2) 228
7-9 FREE PACKAGE command with PLANMGMTSCOPE option 229
7-10 Application program or stored procedure linked with DSNULI 236
7-11 BIND option CONCURRENTACCESSRESOLUTION 238
7-12 USE CURRENTLY COMMITTED clause 238
7-13 CREATE PROCEDURE option list option 238
7-14 SET CURRENT EXPLAIN MODE 240
8-1 The XMLTABLE function syntax 244
8-2 Avoid re-evaluation of XMLEXISTS 258
8-3 XML schemas 259
8-4 XML schemas in XML Schema Repository 263
8-5 Schema determination 264
8-6 CHECK DATA syntax: New keywords 270
8-7 The CHECK DATA utility: SHRLEVEL REFERENCE considerations 272
8-8 CHECK DATA: SHRLEVEL CHANGE considerations (1 of 2) 273
8-9 CHECK DATA: SHRLEVEL CHANGE considerations (2 of 2) 273
8-10 Multiversioning scheme 276
8-11 Multiversioning for XML data 278
8-12 XML locking scheme (with DB2 9 APAR PK55966) 279
8-13 XML Locking scheme with multiversioning 279
8-14 Insert expression syntax 282
8-15 Insert expression examples (1 of 3) 283
8-16 Insert expression examples (2 of 3) 283
8-17 Insert expression examples (3 of 3) 284
8-18 Replace expression syntax 284
8-19 Replace expression examples (1 of 2) 285
8-20 Replace expression examples (2 of 2) 285
8-21 Delete expression syntax 286
8-22 Delete expression example 286
8-23 Binary XML is not the same as FOR BIT DATA 287
8-24 Binary XML in the UNLOAD and LOAD utilities 289
8-25 XML date and time support 291
8-26 XML date and time related comparison operators 293
8-27 XML date and time comparison with SQL date 294
8-28 Arithmetic operations on XML duration 294
8-29 Arithmetic operations on XML duration, date, and time 295
8-30 Date and time related XPath functions 296
8-31 Date and time related XPath functions examples 297
8-32 Time zone adjustment functions on date and time (1 of 2) 298
8-33 Time zone adjustment functions on date and time (2 of 2) 299
8-34 XML index improvement for date and time stamp 300
8-35 Native SQL stored procedure using XML parameter 301
8-36 Native a SQL stored procedure using XML variable 302
8-37 Decompose to multiple tables with a native SQL procedure 305
8-38 Decompose to multiple tables with a native SQL procedure (DB2 9) 306
9-1 -DISPLAY LOCATION output 311
9-2 -MODIFY DDF ALIAS syntax diagram 311
9-3 DISPLAY DDF command 313

9-4 DISPLAY DDF DETAIL command 314
9-5 Extended correlation token in messages 317
9-6 Contents of DSN_PROFILE_ATTRIBUTES 320
9-7 JDBC type 2 with DB2 9 322
9-8 JDBC type 2 with DB2 10 323
9-9 Administrative task scheduler 330
10-1 Start audit trace parameter AUDTPLCY 344
10-2 Use of AUDTPLCY in start, stop trace commands 345
10-3 Start AUDSYSADMIN audit policy 345
10-4 Start multiple audit trace policies with one start trace command 345
10-5 Display audit policy AUDSYSADM 346
10-6 Stop audit policy AUDSYSADMIN 346
10-7 Start AUDTPLCY reason code 00E70022 346
10-8 Start AUDTPLCY reason code 00E70021 347
10-9 Start AUDTPLCY reason code 00E70024 347
10-10 OMEGAMON PE IFCID 362 RECTRACE report 347
10-11 OMEGAMON PE IFCID 361 record trace 348
10-12 OMEGAMON PE V1R5 JCL sample 349
10-13 OMEGAMON PE record trace for static SQL 353
10-14 System authorities 355
10-15 SQL syntax grant system 357
10-16 SQL syntax revoke system 358
10-17 user DB2R53 is connected to RACF group DB0BSECA 362
10-18 OMEGAMON PE category SECMAINT audit report 363
10-19 OMEGAMON PE auth type SYSDBADM audit report 367
10-20 OMEGAMON PE auth type DATAACCESS audit report 369
10-21 OMEGAMON PE auth type ACCESSCTRL audit report 371
10-22 SQLADM authority and EXPLAIN privilege 372
10-23 EXPLAIN privilege failure with CURRENT EXPLAIN MODE special register 373
10-24 SQLADM authority to use CURRENT EXPLAIN MODE special register 375
10-25 OMEGAMON PE auth type SQLADM audit report 376
10-26 SECADM and ACCESSCTRL not separated from SYSADM 378
10-27 SECADM and ACCESSCTRL separated from SYSADM 379
10-28 SQLADM Execution failure non-system-defined routine 382
10-29 SQLADM run system-defined user provided UDF 382
10-30 Revoke syntax diagram 383
10-31 Row permission enforcement 385
10-32 Column mask enforcement 386
10-33 Alter table row access control 387
10-34 Alter table column access control 387
10-35 CREATE PERMISSION SQL DDL 387
10-36 ALTER PERMISSION DDL 387
10-37 CREATE MASK SQL DDL 388
10-38 ALTER MASK DDL 388
10-39 Catalog tables and access control 389
10-40 Sample data customer table 390
10-41 Default row permission stored in SYSIBM.SYSCONTROLS 391
10-42 Query row access control with default predicate 1=0 392
10-43 Impact of default predicate on query result 392
10-44 SYSIBM.SYSDEPENDENCIES RA01_CUSTOMER 394
10-45 Query permission RA01_CUSTOMER without query predicate 394
10-46 Query permission RA01_CUSTOMER and with query predicate 395
10-47 SYSIBM.SYSDEPENDENCIES rows for INCOME_BRANCH column mask 397

10-48 Query with column access control with column mask applied 398
10-49 Permission and mask policies activated for query sample 399
10-50 Customer table query for SQLID DB0B#B 399
10-51 Customer table query for SQLID DB0B#B salary > 80000 399
10-52 Customer table query for SQLID DB0B#C selecting rows of branch C 400
10-53 Alter trigger syntax 401
10-54 RA02_CUSTOMERS EXPLAIN output 403
10-55 RA02_CUSTOMERS DSN_PREDICAT_TABLE query output 404
10-56 TRA02_CUSTOMERS DSN_STRUCT_TABLE query output 404
10-57 OMEGAMON PE formatted audit report for audit category EXECUTE 405
10-58 Query SYSIBM.SYSPACKDEP 406
10-59 Query DSN_STATEMENT_CACHE_TABLE 406
10-60 DRDA Environment for password phrase scenario 412
10-61 SYSIBM.USERNAMES row DSNLEUSR encrypted 414
10-62 SPUFI and outbound translation with password phrase 414
10-63 OMEGAMON PE RECTRACE report on DRDA outbound translation 414
10-64 TestJDBC output 415
10-65 DISPLAY THREAD taken during COBOL SQL connect scenario 416
10-66 z/OS identity propagation TrustedContextDB2zOS flow 417
10-67 RACMAP LISTMAP command output 418
10-68 getDB2TrustedPooledConnection 420
10-69 getDB2Connection 420
10-70 Display thread output right after trusted context pool connection 421
10-71 Display thread output right after trusted context switch 422
10-72 Audit report trusted context SET CURRENT SQLID 422
10-73 Audit report establish trusted context 423
10-74 Audit report distributed identity: RACF authorization ID mapping 423
10-75 Audit report trusted context switching 424
11-1 FCCOPY keyword on COPY syntax 427
11-2 FLASHCOPY_COPY=YES versus FLASHCOPY_COPY=NO in DSNZPARM 429
11-3 Create FlashCopy image copy and sequential copy depending on ZPARM setting . 430
11-4 FlashCopy image copy of partitioned table space plus sequential copy 432
11-5 FLASHCOPY CONSISTENT 434
11-6 REBUILD_INDEX without FLASHCOPY 441
11-7 REBUILD INDEX with FLASHCOPY 442
11-8 REORG INDEX without FLASHCOPY 443
11-9 Profile definition for RUNSTATS 446
11-10 RECOVER BACKOUT YES: Base situation 449
11-11 RECOVER TABLESPACE without BACKOUT YES 449
11-12 RECOVER TABLESPACE ... BACKOUT YES 450
11-13 Recover multiple table spaces to the same UR with BACKOUT YES 451
11-14 Enhanced REORG TABLESPACE syntax for PART keyword 459
11-15 -DIS DB command output after a couple of ROTATE commands 460
11-16 LISTDEF syntax change for multiple partition ranges 460
11-17 FORCE keyword on REORG 461
11-18 Possible effect of option FORCE ALL on REORG 462
11-19 CHECK DATA with inconsistencies found and CHECK_SETCHKP=YES 465
11-20 CHECK DATA with inconsistencies found and CHECK_SETCHKP=NO 466
12-1 DB2 version summary 472
12-2 Migrating from DB2 9 to DB2 10 and fallback paths 475
12-3 Migrating from DB2 V8 to DB2 10 and fallback paths 476
12-4 DB2 10 optional features 484
12-5 DB2 catalog evolution 489

12-6 DB2 directory table changes 495
12-7 Catalog tables for security and auditing in DB2 10 496
12-8 Catalog tables for pending object changes in DB2 10 497
12-9 Catalog tables for BIND QUERY support in DB2 10 498
12-10 Installation pop-up panel DSNTIPSV 520
13-1 DB2 10 performance objective 535
13-2 Prefetch window 547
13-3 MODIFY DDF command 549
13-4 DB2 trace records IFCID 58 and 59 for OPEN and unbundled DRDA block prefetch 552
13-5 DB2 trace records IFCID 58 and 59 for OPEN and bundled DRDA block prefetch . 552
13-6 Statistics report 553
13-7 Attributes clause of the PREPARE statement 555
13-8 Summary of main insert performance improvements 557
13-9 ALTER BUFFERPOOL command 563
13-10 Streaming LOBs and XML 568
13-11 Hash space structure 576
13-12 Hash access and partitioning 577
13-13 ORGANIZE BY HASH clause 579
13-14 ORGANIZE BY HASH Partition clause 580
13-15 SQL PL multiple assignment statement 591
13-16 Key range partitioning 594
13-17 Dynamic record based partitioning 595
13-18 Workload balancing straw model 596
13-19 Accounting suspend times 599
13-20 Statistics, DIST storage above 2 GB 600
13-21 Statistics, DIST storage below 2 GB 600
13-22 System level monitoring - Invalid profile 607
13-23 System level monitoring - Attributes table 610

Archive

Examples

2-1 DSN_WLM_APPLENV stored procedure call 32
2-2 SET_CLIENT_INFO stored procedure invocation 34
2-3 JCL RMF workload activity report 40
3-1 Queries with parallelism enabled 53
3-2 DSNTWFG and description of its parameters 60
4-1 Table space and index space as partition-by-growth 81
4-2 REORG TABLESPACE with AUTOESTSPACE YES 83
4-3 REORG utility message DSNUGHSH 83
4-4 REORG TABLESPACE with AUTOESTSPACE NO 84
4-5 Data set sizes after REORG 84
4-6 REPORT RECOVERY job output 94
4-7 Define clusters for the new active log data sets 99
4-8 -DIS LOG command output 99
4-9 Message for LRDRTHLD threshold exceeded 107
6-1 inline SQL scalar function KM2MILES 136
6-2 Non-inline function KM2MILES_SLOW using option not previously available 136
6-3 Inline SQL scalar function TRYXML using XML data type 136
6-4 Non-inline SQL scalar function TRYSFS using scalar fullselect 137
6-5 Non-inline SQL scalar function TRYTBL with transition table parameter 137
6-6 Special register behavior within the SQL scalar function body 138
6-7 SQL scalar function versioning example 140
6-8 Deployment of a SQL scalar function 141
6-9 Initial definition of the function for the running example 141
6-10 ALTER FUNCTION (SQL scalar) ALTER option-list 141
6-11 ALTER FUNCTION (SQL scalar) ADD VERSION 142
6-12 ALTER FUNCTION (SQL scalar) ACTIVATE VERSION 142
6-13 ALTER FUNCTION (SQL scalar) REPLACE 142
6-14 ALTER FUNCTION (SQL scalar) REGENERATE 142
6-15 ALTER FUNCTION (SQL scalar) DROP 142
6-16 SQL table function definition and invocation 147
6-17 ALTER FUNCTION (SQL table) 147
6-18 Distinct type in SQL procedure 148
6-19 Implicit cast from numeric to string 150
6-20 Implicit cast from string to numeric 151
6-21 Rounding with implicit cast from string to decimal 152
6-22 Implicit cast with function resolution 153
6-23 Examples of datetime constants 155
6-24 Difference between datetime constant and character string constant 156
6-25 Examples of timestamp precision 157
6-26 Timestamp assignment example 158
6-27 Timestamp comparison example 159
6-28 Example of timestamp precision of the result 159
6-29 CURRENT TIMESTAMP reference examples 160
6-30 EXTRACT scalar function example for timestamp 161
6-31 SECOND scalar function example for timestamp 161
6-32 LENGTH scalar function example for timestamp 161
6-33 MICROSECOND scalar function example for timestamp 162
6-34 TIMESTAMP scalar function example for timestamp 162

© Copyright IBM Corp. 2010. All rights reserved.

6-35 TIMESTAMP declaration examples 167
6-36 Actual and reserved lengths of a TIMESTAMP WITH TIME ZONE 169
6-37 Examples of string representation of a TIMESTAMP WITH TIME ZONE. 169
6-38 Promotion of timestamp data type 170
6-39 Example of TIMESTAMP WITH TIME ZONE casts 171
6-40 Timestamp with time zone comparison examples 172
6-41 Result data type when operation involves TIMESTAMP WITH TIME ZONE 173
6-42 CURRENT TIMESTAMP WITH TIME ZONE examples 174
6-43 CURRENT TIMESTAMP: Current and implicit time zone. 175
6-44 SESSION TIME ZONE example. 175
6-45 Time zone specific expression examples 177
6-46 Timestamp with time zone arithmetic 177
6-47 TIMESTAMP_TZ invocation examples. 179
6-48 Invocations of EXTRACT 180
6-49 TRYOLAP table definition and its data 186
6-50 GROUP BY versus PARTITION BY 187
6-51 ORDER BY ordering in partitions 188
6-52 Aggregation using ROWS. 189
6-53 Sparse input data effect on physical aggregation grouping 190
6-54 Duplicate rows effect on physical aggregation grouping: Query. 191
6-55 Duplicate rows effect on physical aggregation grouping: Result 1 191
6-56 Duplicate rows effect on physical aggregation grouping: Result 2 191
6-57 Aggregation using RANGE 192
6-58 Sparse input data effect on logical aggregation grouping. 193
6-59 Duplicate rows effect on logical aggregation grouping 193
6-60 OLAP-specification aggregation group boundary example 1 196
6-61 OLAP-specification aggregation group boundary example 2 196
6-62 OLAP-specification aggregation group boundary example 3 197
6-63 OLAP-specification aggregation group boundary example 4 197
6-64 OLAP-specification aggregation group boundary example 5 198
6-65 OLAP-specification aggregation group boundary example 6 198
6-66 OLAP-specification aggregation group boundary example 7 199
6-67 OLAP-specification aggregation group boundary example 8 199
6-68 OLAP-specification aggregation group boundary example 9 200
6-69 OLAP-specification aggregation group boundary example 10 200
6-70 OLAP-specification aggregation group boundary example 11 200
6-71 OLAP-specification aggregation group boundary example 12 201
7-1 SYSTEM_TIME definition for a table 205
7-2 Creating a system-period temporal table and enabling data versioning. 206
7-3 Example of SYSIBM.SYSTABLE data versioning information 207
7-4 Enabling system-period data versioning for an existing table. 207
7-5 System-period data versioning with generated columns. 210
7-6 DML with system-period data versioning: INSERT. 211
7-7 DML with system-period data versioning: Update. 212
7-8 DML with system-period data versioning: Delete 212
7-9 Example of FOR SYSTEM_TIME AS OF 214
7-10 Example of FOR SYSTEM_TIME FROM... TO. 214
7-11 Example of FOR SYSTEM_TIME BETWEEN... AND. 215
7-12 BUSINESS_TIME period definition for a table 216
7-13 Examples of BUSINESS_TIME WITHOUT OVERLAPS clause. 216
7-14 Example of FOR BUSINESS_TIME clause 217
7-15 Example of FOR PORTION OF BUSINESS_TIME semantics: Case 1 218
7-16 Example of FOR PORTION OF BUSINESS_TIME semantics: Case 2 218

7-17 Example of FOR PORTION OF BUSINESS_TIME semantics: Case 3 219
7-18 Example of FOR PORTION OF BUSINESS_TIME semantics: Case 4 219
7-19 FREE QUERY Example 1. 223
7-20 FREE QUERY Example 2. 223
7-21 FREE QUERY Example 3. 223
7-22 Extended indicator variable example: Table declaration 233
7-23 Extended indicator variable example: C application for INSERT 233
7-24 Extended indicator variable example: C application for UPDATE. 234
7-25 Link-editing DSNULI to your application 237
8-1 Iteration over results using the XMLTABLE function 245
8-2 Sorting values from an XML document. 246
8-3 Stored XML documents 246
8-4 Inserting values returned from XMLTABLE 247
8-5 Returning one row for each occurrence of phone item as a string 247
8-6 Returning one row for each occurrence of phone item as an XML document 248
8-7 Specifying a default value for a column in the result table 249
8-8 Specifying an ordinality column in a result table 249
8-9 The XMLCAST specification: Implicit casting 250
8-10 The XMLCAST specification: Explicit casting 250
8-11 XML index usage by join predicates example 1 251
8-12 XML index usage by join predicates example 2 251
8-13 Query example using function fn:exists() 252
8-14 Query example using function fn:not() 252
8-15 Query example using function fn:upper-case() 253
8-16 Query example using function fn:starts-with(). 253
8-17 Query example using function fn:substring() 254
8-18 Query example using XMLTABLE (original query) 255
8-19 Query example using XMLTABLE (transformed query) 255
8-20 Query example using XMLTABLE with WHERE clause (original query) 256
8-21 Query example using XMLTABLE with WHERE clause (First transformation). 256
8-22 Query example using XMLTABLE with WHERE clause (Second transformation) . . 256
8-23 Specify XML type modifier for XML column at create time 260
8-24 Table definition without XML type modifier 260
8-25 Specify XML type modifier for XML column at alter time 260
8-26 Add an XML schema to the XML type modifier. 260
8-27 Reset XML type modifier for XML column at alter time. 261
8-28 Identify an XML schema by target namespace and schema location. 261
8-29 Identify an XML schema by target namespace. 261
8-30 No namespace 262
8-31 Specifying global element name. 262
8-32 Schema selection for validation from an XML type modifier: Example 1 265
8-33 Schema selection for validation from an XML type modifier: Example 2 265
8-34 Schema selection for validation from an XML type modifier: Example 3 265
8-35 Schema selection for validation from an XML type modifier: Example 4 266
8-36 Schema selection for validation for DSN_XMLVALIDATE: Example 1. 267
8-37 Schema selection for validation for DSN_XMLVALIDATE: Example 2. 267
8-38 Schema selection for validation for DSN_XMLVALIDATE: Example 3. 267
8-39 Schema selection for validation for DSN_XMLVALIDATE: Example 4. 268
8-40 Search for documents not validated 269
8-41 Retrieve target namespaces and XML schema names used for validation 269
8-42 CHECK DATA example 272
8-43 Self referencing UPDATE of an XML column 274
8-44 XML data. 281

8-45 JDBC application example . . . 281
8-46 COBOL example . . . 281
8-47 Row filtering using XMLEXISTS . . . 282
8-48 Application using JDBC 4.0 (fetch XML as SQLXML type) . . . 288
8-49 Application using JDBC 4.0 (insert and update XML using SQLXML) . . . 289
8-50 Samples of the time zone adjustment functions . . . 299
8-51 User-defined SQL scalar function using XML variable . . . 302
8-52 User-defined SQL table function using XML variable . . . 304
9-1 REBIND job example . . . 326
10-1 Audit all SQL for multiple tables . . . 343
10-2 Audit policy for category SYSADMIN . . . 343
10-3 Mark audit policy for automatic start during DB2 startup . . . 343
10-4 AUD dynamic SQL for auditing . . . 350
10-5 AUD SQL static SQL for auditing . . . 351
10-6 Create SQL statement auditing policy . . . 351
10-7 Start SQL statement auditing policy . . . 351
10-8 Display SQL statement audit policy . . . 352
10-9 Stop SQL statement audit policy . . . 352
10-10 OMEGAMON PE record trace for dynamic SQL . . . 354
10-11 SECAM to revoke privileges granted by others . . . 363
10-12 SECMAINT audit policy - grant - revoke auditing . . . 363
10-13 SECMAINT audit policy - grant DATAACESS SQL . . . 363
10-14 Grant or revoke system DBADM privilege . . . 365
10-15 Grant system DBADM WITHOUT DATAACCESS WITHOUT ACCESSCTRL . . . 365
10-16 DBADMIN audit policy - system DBADM auditing . . . 367
10-17 DBADMIN audit policy - create table by system DBADM . . . 367
10-18 Grant or revoke the DATAACCESS privilege . . . 368
10-19 DBADMIN audit policy: DATAACCESS auditing . . . 369
10-20 DBADMIN audit policy - query a table by DATAACCESS authority . . . 369
10-21 Grant and revoke the ACCESSCTRL privilege . . . 370
10-22 SECADM to revoke privileges granted by others . . . 370
10-23 DBADMIN audit policy: ACCESSCTRL auditing . . . 371
10-24 DBADMIN audit policy: Grant privilege by ACCESSCTRL authority . . . 371
10-25 Grant, revoke the explain privilege . . . 373
10-26 EXPLAIN . . . 374
10-27 Grant, revoke SQLADM privilege . . . 375
10-28 DBADMIN audit policy: SQLADM auditing . . . 376
10-29 DBADMIN audit policy: Query table by SQLADM authority . . . 376
10-30 SQLADM marking UDF as system defined through dummy alter . . . 382
10-31 Activate row access control on customer table . . . 391
10-32 Row permission object RA01_CUSTOMER SQL DDL . . . 393
10-33 Activate column access control on customer table . . . 395
10-34 Deactivate row access control on customer table . . . 396
10-35 Column mask INCOME_BRANCH . . . 396
10-36 RA02_CUSTOMERS row permission referencing SYSIBM.SYSDUMMY1 . . . 402
10-37 RA02_CUSTOMERS explain a query accessing the customer table . . . 402
10-38 Change the password phrase using DB2 CLP . . . 412
10-39 Configure DB2 communications database . . . 412
10-40 DSNLEUSR stored procedure invocation . . . 413
10-41 Shell script for invoking the TestJDBC application . . . 415
10-42 COBOL logic for connecting to a remote location . . . 415
10-43 Activate RACF class IDIDMAP . . . 418
10-44 RACMAP MAP command to add identity mapping . . . 418

10-45 Refresh IDIDMAP class profiles 418
10-46 Trusted context definition for identify propagation 419
11-1 SAMPLE COPY FLASHCOPY YES 427
11-2 Sample COPY FLASHCOPY YES job output 428
11-3 Invalid name error message 428
11-4 COPYTOCOPY messages for FlashCopy image copy input 431
11-5 Messages from RECOVER utility 433
11-6 First part of FLASHCOPY CONSISTENT 435
11-7 Messages for FlashCopy image copy consistency processing 436
11-8 Error message you get when no SYSCOPY DD specified 439
11-9 DSNU552I trying to recover TS in ICOPY 441
11-10 REBUILD INDEX FLASHCOPY YES output extract 442
11-11 RACF error message when trying to involve FlashCopy in recover 445
11-12 RUNSTATS PROFILE utility execution 447
11-13 RECOVER TABLESPACE ... BACKOUT YES job output 450
11-14 Error message preventing BACKOUT YES recovery 452
11-15 REORG ... AUX YES job output 454
11-16 REORG ... AUX YES and TEMPLATE job output 456
11-17 Unsuccessful attempt to work with FORCE READERS 463
11-18 -DIS DB ... LOCKS for database object 464
11-19 -DIS THREAD(*) output for agent 543 464
11-20 REPORT RECOVERY output with SLBs available 469
11-21 REPORT RECOVERY utility output new section 470
12-1 Message if fallback SPE is not applied 475
12-2 Messages to indicate whether BSDS was converted 477
12-3 Expanded format BSDS required to start DB2 10 477
12-4 Output of successful run of DSNJCNVB conversion program 478
12-5 Output of DSNJCNVB BSDS conversion program on an already converted BSDS . 478
12-6 CHAR casting in DB2 9 NFM and DB2 10 CM 479
12-7 Sample call to DSNXTA 510
12-8 Warning message DSNT0921 511
12-9 Sample call to DSNTXTB 512
12-10 Output of job DSNTIJXA program DSNTXTA to convert all schemas 514
12-11 Failure if no SMS definitions for DB2 catalog and directory are defined 517
12-12 Output of CATMAINT job DSNTIJTC DB2 V8 NFM to DB2 10 CM 517
12-13 Execution of DSNTRIN with PREVIEW option in INSTALL mode 521
12-14 Sample output from DSNTIJRW 527
12-15 Program DSNTWLMB needs to be APF-authorized to activate WLM changes . . . 528
13-1 OMEGAMON PE statistics report sample on DBATs 550
13-2 Unloading in spanned format 566
13-3 Query to identify indexes for possible INCLUDE 588
13-4 Possible INCLUDE candidates 589
A-1 Changes to IFCID 002 - Dynamic statement cache 616
A-2 IFCID 002 - Currently committed data 616
A-3 IFCID 013 and IFCID 014 617
A-4 Changes to IFCID 106 618
A-5 Changes to IFCID 225 618
A-6 Changes to IFCID 267 621
A-7 Changes to IFCID 316 621
A-8 New IFCID 357 622
A-9 New IFCID 358 622
A-10 New IFCID 359 622
A-11 New IFCID 363 623

Archived

Tables

2-1 Eligibility for zHPF of DB2 I/O types . . . 19
3-1 EAV enablement and EAS eligibility . . . 63
4-1 SYSIBM.SYSPENDINGDDL . . . 72
4-2 Maximum DSSIZE depending on table space attributes . . . 75
4-3 Number of partitions by DSSIZE and page size . . . 80
4-4 New SYSIBM.SYSCOPY information . . . 87
4-5 SYSIBM.SYSPENDINGOBJECTS . . . 88
5-1 DB2 9 GRECP/LPL maintenance . . . 121
6-1 CREATE and ALTER FUNCTION options resulting in rebind when changed . . . 143
6-2 Implicit cast target data types and length . . . 149
6-3 Datetime constant formats . . . 154
6-4 Formats for string representation of dates . . . 155
6-5 Formats for string representation of times . . . 155
6-6 Declarations generated by DCLGEN . . . 163
6-7 Equivalent SQL and Assembly data types . . . 164
6-8 Equivalent SQL and C data types . . . 164
6-9 Equivalent SQL and COBOL data types . . . 164
6-10 Equivalent SQL and PL/I data types . . . 164
6-11 Declarations generated by DCLGEN . . . 182
6-12 Equivalent SQL and Assembly data types . . . 182
6-13 Equivalent SQL and C data types . . . 182
6-14 Equivalent SQL and COBOL data types . . . 183
6-15 Equivalent SQL and PL/I data types . . . 183
6-16 Equivalent SQL and Java data types . . . 183
6-17 Supported aggregation group boundary combinations . . . 195
7-1 Values for extended indicator variables . . . 232
7-2 Catalog changes for CONCURRENTACCESSRESOLUTION . . . 239
8-1 Contents of CUSTADDR table after insert of a result table generated by XMLTABLE 247
8-2 Result table of a query using XMLTABLE to retrieve multiple occurrences of an item 248
8-3 Result table of a query using XMLTABLE to retrieve multiple occurrences of an item 248
8-4 Result table from a query in which XMLTABLE has a default value for an item . . . 249
8-5 Result table from a query in which XMLTABLE has a ordinality column . . . 249
8-6 CHECK DATA invocation . . . 271
8-7 Data in table T1 . . . 275
8-8 Return table . . . 304
9-1 Categories for system group monitoring . . . 319
9-2 Sample of contents of DSN_PROFILE_TABLE . . . 319
10-1 Audit categories . . . 339
10-2 The SYSIBM.SYSAUDITPOLICIES catalog table . . . 340
10-3 New and changed IFCIDs . . . 342
10-4 Catalog table changes for SYSIBM.SYSUSERAUTH . . . 359
10-5 Privileges held by SECADM authority . . . 360
10-6 DSNZPARMs for SECADM . . . 360
10-7 SECADM authority DSNZPARMs settings of our sample scenario . . . 362
10-8 System DBADM privileges . . . 364
10-9 Additional privileges required by system DBADM . . . 366
10-10 DATAACCESS implicitly held grantable privileges . . . 368
10-11 ACCESSCTRL implicitly held grantable privileges . . . 370

© Copyright IBM Corp. 2010. All rights reserved.

10-12 Privileges of EXPLAIN privilege . . . 372
10-13 DB2 RACF profiles for DB2 10 new authorities . . . 377
10-14 Default behavior for REVOKE_DEPENDING_PRIVILEGES . . . 384
10-15 DSN_FUNCTION_TABLE change for row and column access control . . . 401
10-16 DSN_PREDICAT_TABLE changes . . . 403
10-17 DSN_STRUCT_TABLE . . . 404
10-18 Access control changes for SYSIBM.SYSCOLUMNS . . . 407
10-19 Access control new table SYSIBM.SYSCONTROLS . . . 407
10-20 Access control changes for SYSIBM.SYSDEPENDENCIES . . . 409
10-21 Access control changes to SYSIBM.SYSOBJROLEDEP . . . 409
10-22 Access control changes to SYSIBM.SYSROUTINES . . . 409
10-23 Access control changes to SYSIBM.SYSTABLES . . . 410
10-24 Access control changes to SYSIBM.SYSTRIGGERS . . . 410
10-25 Access control changes to SYSIBM.SYSUSERAUTH . . . 410
10-26 Changed IFCIDs for row and column access control . . . 411
10-27 SYSIBM.USERNAMES change . . . 413
11-1 DSNZPARM, FLASHCOPY keyword, resulting copy . . . 430
11-2 REORG TABLESPACE ... AUX option . . . 453
11-3 REORG TABLESPACE ... AUX option . . . 453
11-4 Available SHRLEVELs for LOB REORG depending on DB2 versions . . . 458
11-5 DBET state by function . . . 466
12-1 System software requirements . . . 473
12-2 Catalog and directory tables and their partition-by-growth table spaces . . . 493
12-3 Catalog tables and table spaces in DB2 10 . . . 499
12-4 Removed system parameters . . . 501
12-5 Deprecated system parameters . . . 502
12-6 System parameters that are deprecated and have new default values . . . 502
12-7 System parameters with new maximums . . . 503
12-8 System parameters with new default settings . . . 503
12-9 DB2 10 new system parameters . . . 505
12-10 DPSEGSZ and type of table space created . . . 507
12-11 DSN_PTASK_TABLE column name corrections . . . 516
12-12 New or revised installation or migration verification jobs . . . 522
12-13 Revised sample job DSNTEJ2H: Programs . . . 522
12-14 DB2-supplied stored procedures and WLM environment setup . . . 523
12-15 DB2 default WLM environments . . . 527
13-1 PLAN_TABLE extract for range list access . . . 540
13-2 PLAN_TABLE extract for IN-list . . . 541
13-3 Column LITERAL_REPL values . . . 556
13-4 Catalog table changes for inline LOBs . . . 571
13-5 Catalog table changes for hash access . . . 582
13-6 Catalog table changes for INCLUDE . . . 587
13-7 Column DRIVETYPE . . . 590
13-8 DSN_PGROUP_TABLE changes . . . 596
13-9 Sort merge join PLAN_TABLE changes . . . 597
B-1 DB2 10 current function-related APARsPM43293 . . . 628
B-2 z/OS DB2-related APARs . . . 634
B-3 OMEGAMON PE GA and DB2 10 related APARs . . . 635

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 illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2010. All rights reserved.

Trademarks

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

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

AIX®
BladeCenter®
CICS®
Cognos®
DataPower®
DB2 Connect™
DB2 Universal Database™
DB2®
developerWorks®
Distributed Relational Database Architecture™
DRDA®
DS8000®
Enterprise Storage Server®
Enterprise Workload Manager™
FICON®
FlashCopy®
GDPS®
Geographically Dispersed Parallel Sysplex™
IBM®
IMS™
Language Environment®
MQSeries®
MVS™
Net.Data®
OMEGAMON®
OmniFind®
Optim™
OS/390®
Parallel Sysplex®
POWER6+™
POWER7™
PowerVM™
PR/SM™
pureXML®
QMF™
Query Management Facility™
RACF®
Redbooks®
Redbooks (logo) ®
Resource Measurement Facility™
RMF™
S/390®
Service Request Manager®
Solid®
System Storage®
System z10®
System z9®
System z®
Tivoli®
TotalStorage®
VTAM®
WebSphere®
z/Architecture®
z/OS®
z/VM®
z10™
z9®
zSeries®

The following terms are trademarks of other companies:

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

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

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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, or service names may be trademarks or service marks of others.

Archive

Preface

IBM® DB2® Version 10.1 for z/OS® (DB2 10 for z/OS or just *DB2 10* throughout this book) is the fourteenth release of DB2 for MVS™. It brings improved performance and synergy with the System z® hardware and more opportunities to drive business value in the following areas:

- Cost savings and compliance through optimized innovations
  - DB2 10 delivers value in this area by achieving up to 10% CPU savings for traditional workloads and up to 20% CPU savings for nontraditional workloads, depending on the environments. Synergy with other IBM System z platform components reduces CPU use by taking advantage of the latest processor improvements and z/OS enhancements.
  - Streamline security and regulatory compliance through the separation of roles between security and data administrators, column level security access, and added auditing capabilities.
- Business insight innovations
  - Productivity improvements are provided by new functions available for pureXML®, data warehousing, and traditional online TP applications
  - Enhanced support for key business partners that allow you to get more from your data in critical business disciplines like ERP
  - Bitemporal support for applications that need to correlate the validity of data with time.
- Business resiliency innovations
  - Database on demand capabilities to ensure that information design can be changed dynamically, often without database outages
  - DB2 operations and utility improvements enhancing performance, usability, and availability by exploiting disk storage technology.

The DB2 10 environment is available either for brand new installations of DB2, or for migrations from DB2 9 for z/OS or from DB2 UDB for z/OS Version 8 subsystems.

This IBM Redbooks® publication introduces the enhancements made available with DB2 10 for z/OS. The contents help you understand the new functions and performance enhancements, start planning for exploiting the key new capabilities, and justify the investment in installing or migrating or skip migrating to DB2 10.

The team who wrote this book

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

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

© Copyright IBM Corp. 2010. All rights reserved.

Rafael Garcia has been in the IT business for 28 years and has held various positions. He was a COBOL and CICS® Developer, a DOS/VSE Systems Programmer, an Application Development Manager, and a DB2 Applications DBA for one of the top 10 banks in the U.S. Rafael has 18 years of experience working with DB2 for z/OS. For the last 13 years, he has been a field DB2 Technical Specialist working for the IBM Silicon Valley Laboratory and supporting DB2 for z/OS customers across various industries, including migrations to data sharing, release migration support, and assistance in resolution of customer critical issues. He has an Associate's Degree in Arts and an Associate's Degree in Science in Business Data Processing from Miami-Dade Community College. He has co-authored several IBM Redbooks publications, including *DB2 for z/OS Application Programming Topics*, SG24-6300, *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079, *DB2 UDB for z/OS Version 8 Technical Preview*, SG24-6871, and *DB2 9 for z/OS Technical Overview*, SG24-7330.

Sabine Kaschta is a DB2 Specialist working for the IBM Software Group in Germany. Currently, she primarily works as a Segment Skills Planner for the worldwide curriculum for DB2 for z/OS training. She also works on course development and teaches several DB2 for z/OS classes worldwide. Sabine has 17 years of experience working with DB2. Before joining IBM in 1998, she worked for a third-party vendor providing second-level support for DB2 utilities. She is experienced in DB2 system programming and client/server implementations in the insurance industry in Germany. She co-authored the several IBM Redbooks publications, including *DB2 UDB for OS/390 and Continuous Availability*, SG24-5486, *Cross-Platform DB2 Distributed Stored Procedures: Building and Debugging*, SG24-5485, *IBM TotalStorage Migration Guide for the SAP User*, SG24-6400, *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079, *DB2 9 for z/OS Technical Overview*, SG24-7330, *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7604, and *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663.

Josef Klitsch is a Senior IT Specialist for z/OS Problem Determination tools with IBM Software Group, Switzerland. After he joined IBM in 2001, he provided DB2 consultancy and technical support to Swiss DB2 for z/OS customers and worked as a DB2 subject matter expert for IBM China and as a DB2 for z/OS technical resource for IBM France in Montpellier. Prior to his IBM employment, Josef worked, for more than 15 years, for several European customers as an Application Developer, Database Administrator, and DB2 systems programmer with a focus on DB2 for z/OS and its interfaces. His preferred area of expertise in DB2 for z/OS is stored procedures programming and administration. He co-authored the IBM Redbooks publication *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663.

Ravi Kumar is a Senior Instructor and Specialist for DB2 with IBM Software Group, Australia. He has about 25 years of experience in DB2. He was on assignment at the International Technical Support Organization, San Jose Center, as a Data Management Specialist from 1994 to 1997. He has coauthored many IBM Redbooks publications, including *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079 and *DB2 9 for z/OS Technical Overview*, SG24-7330. He is currently on virtual assignment as a Course Developer in the Education Planning and Development team, Information Management, IBM Software Group, U.S.

Andrei Lurie is a Senior Software Engineer in IBM Silicon Valley Lab, San Jose. He has 10 years of experience in developing DB2 for z/OS and has participated in the development of many major line items for DB2 since DB2 for z/OS Version 8. Andrei's main areas of expertise are in SQL and application development. Currently, he is a technical lead for Structure Generator and Runtime components of the DB2 RDS area. He holds a master degree in Computer Science from the San Jose State University, where he enjoys teaching DB2 for z/OS topics as a guest lecturer from time to time.

Michael Parbs is a Senior DB2 Specialist with IBM Global Technology Services A/NZ, from Canberra, Australia. He has over 20 years experience with DB2, primarily on the z/OS platform. Before joining IBM he worked in the public sector in Australia, as a DB2 DBA and DB2 Systems Programmer. Since joining IBM, Michael has worked as a subject matter expert with a number of DB2 customers both in Australia and China. Michael's main areas of expertise are data sharing, and performance and tuning, but his skills include database administration and distributed processing. Michael is an IBM Certified IT Specialist in Data Management and has coauthored several IBM Redbooks publications, including *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, *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079, and *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465.

Rajesh Ramachandran is a Senior Software Engineer in IBM System z e-Business Services. He currently works in the Design Center in Poughkeepsie as an IT Architect and DB2 assignee. He has 14 years of experience in application development on various platforms, which include z/OS, UNIX®, and Linux® using COBOL, Java™, CICS, and Forte.

Acknowledgments

The authors thank Dave Beulke for his contribution in written content. David is an internationally recognized DB2 consultant, author, and instructor. He is known for his extensive expertise in database performance, data warehouses, and Internet applications. He is currently a member of the IBM DB2 Gold Consultant program, an IBM Information Champion, past president of the International DB2 Users Group (IDUG), coauthor of the IBM DB2 V8 and V7 z/OS Administration and the Business Intelligence Certification exams, columnist for the IBM Data Management Magazine, former instructor for The Data Warehouse Institute (TDWI), and former editor of the IDUG Solutions Journal. With over 22 years experience working with mainframe, UNIX, and Windows® environments, he redesigns and tunes systems, databases, and applications dramatically, reducing CPU demand and saving his clients from CPU upgrades and millions in processing charges.

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

Rich Conway
Bob Haimowitz
Emma Jacobs
IBM International Technical Support Organization

Saghi Amirsoleymani
Jeff Berger
Chuck Bonner
Mengchu Cai
Gayathiri Chandran
Julie Chen
Chris Crone
Jason Cu
Marko Dimitrijevic
Margaret Dong
Cathy Drummond
Thomas Eng
Bill Franklin
Craig Friske
Shivram Ganduri
James Guo

Archive

Keith Howell
Akiko Hoshikawa
David Kuang
Laura Kunioka-Weis
Jeff Josten
Andrew Lai
Dave Levish
Kim Lyle
Bob Lyle
John Lyle
Tom Majithia
Bruce McAlister
Pat Malone
Roger Miller
Chris Munson
Ka-Chun Ng
Betsy Pehrson
Jim Pickel
Terry Purcell
Haakon Robers
Mike Shadduck
Michael Schenker
Jack Shedden
Akira Shibamija
Derek Tempongko
John Tobler
Lingyun Wang
Peter Wansch
Jay Yothers
David Zhang
Guogen Zhang
Binghui Zhong
IBM Silicon Valley Lab

Rick Butler
BMO Toronto

Heidi Arnold
Boris Charpiot
Norbert Heck
Norbert Jenninger
Joachim Pioch
Rosemarie Roehm-Frenzel
IBM Boeblingen Lab

Stephane Winter
IBM Montpellier

Chris Meyer
IBM Durham, NC USA

Christian Heimlich
IBM Frankfurt, Germany

Archived

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

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

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks publication

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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

Archived

Archived

Summary of changes

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

Summary of Changes
for SG24-7892-00
for DB2 10 for z/OS Technical Overview
as created or updated on July 15, 2014.

December 2010, First Edition

The revisions of this First Edition, first published on December 30, 2010, reflect the changes and additions described below.

March 2011, First Update

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

Changed information

- Corrections in 3.9, “Compression of SMF records” on page 64.
- Removed wrong sentence in 4.1.1, “UTS with DB2 9: Background information” on page 68.
- Removed the previous section 4.1.9, “Convert a partition-by-growth table space to a hash table space” which only cross referenced 13.15, “Hash access” on page 575.
- Corrections in 7.5, “Access to currently committed data” on page 238.
- Updated section 10.2, “More granular system authorities and privileges” on page 354.
- Corrections in text in 11.4.1, “Improvements to online REORG for base tables spaces with LOBs” on page 452.
- Corrections in 13.2.1, “Safe query optimization” on page 537.
- Corrections in 13.2.2, “RID pool enhancements” on page 537.
- Corrections in query in 13.2.5, “Aggressive merge for views and table expressions” on page 542.
- Updated section 13.7, “Referential integrity checking improvement” on page 560.
- Updated text to disclaimer in Example 13-3 on page 588.
- Replaced Figure 13-17 on page 595.
- Removed a section in 13.23, “Enhanced instrumentation” on page 598 on SMF Compression since it is already described in 3.9, “Compression of SMF records” on page 64.
- Updated the referenced bibliography in “Related publications” on page 641.

Archived

© Copyright IBM Corp. 2010. All rights reserved.

New information

- Added an Attention box in 2.12.3, “DB2 10 buffer pool prefetch and deferred write activities”.
- Added text in 3.9, “Compression of SMF records” on page 64.
- Added text in 4.1.2, “UTS with DB2 10: The ALTER options” on page 69.
- Added section 5.2, “Delete data sharing member” on page 112.
- Added PTFs to 9.1, “DDF availability” on page 310.
- Added section 9.8, “Return to client result sets” on page 328.
- Added section “Secure Audit trace” on page 344.
- Added text in section 11.1, “Support FlashCopy enhancements” on page 426.
- Added text in 11.4.3, “Improved usability of REORG of disjoint partition ranges” on page 458.
- Added section 12.2, “Some release incompatibilities” on page 479
- Added Example 12-6 on page 479.
- Added text in Table 12-8 on page 503.
- Added text in Table 12-9 on page 505.
- Added the whole Appendix B, “Summary of relevant maintenance” on page 627.

December 2011, Second Update

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

Changed information

- Correction in 3.1.1, “Support for full 64-bit run time” on page 52.
- Updated APARs in Appendix B, “Summary of relevant maintenance” on page 627.

New information

- Added section A.11, “IFCID 360” on page 623.
- Added APARs in Appendix B, “Summary of relevant maintenance” on page 627.
- Added section B.3, “OMEGAMON PE APARs” on page 635.

August 2013, Third Update

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

Changed information

- Correction in 13.5, “Dynamic statement cache enhancements” on page 553 for JCC driver literal support enabling.
- Updated APARs in Appendix B, “Summary of relevant maintenance” on page 627.

New information

- Added APARs in Appendix B, “Summary of relevant maintenance” on page 627.

Archived

July 2014, Fourth Update

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

Changed information

- Editing corrections in section 9.1.3, “Domain name is now optional” on page 315.
- Corrections in section 9.2.2, “Insert a row in DSN_PROFILE_TABLE” on page 318 from DNS_PROFILE_TABLE to DSN_PROFILE_TABLE.
- Correction in Table 9-1 on page 319 from IPADDR to LOCATION.
- Table 9-2 on page 319 has replaced the previous figure with correction from IPADDR to LOCATION.

Archived

Archived

1

DB2 10 for z/OS at a glance

Today's organizations need to lower operating costs by reducing CPU cycle times while building and maintaining a strong foundation for service-oriented architecture (SOA) and XML initiatives. Database administrators (DBAs) need to find improved database performance, scalability, and availability while also reducing memory management so that managing growth is a simpler process. DB2 10 for z/OS provides innovations in the following key areas:

- Improved operational efficiencies
- Unsurpassed resiliency for business-critical information
- Rapid application and warehouse deployment for business growth
- Enhanced query and reporting facilities

In this chapter, we provide a general overview of DB2 10 for z/OS. We discuss the following topics:

- Executive summary
- Benefits of DB2 10
- Subsystem enhancements
- Application functions
- Operation and performance enhancements

© Copyright IBM Corp. 2010. All rights reserved.

1.1 Executive summary

DB2 10 for z/OS is a tremendous step forward in database technology because of its improvements in performance, scalability, availability, security, and application integration. Its technology provides a clear competitive advantage through its continued improvements in SQL, XML, and integrated business intelligence capabilities.

DB2 10 takes advantage of the recent advances in chip technology, storage devices, and memory capacities through its extensive exploitation of System z 64-bit architecture to reduce CPU requirements, to improve performance, and potentially to reduce the total cost of ownership. The DB2 10 CPU reduction features provide enhanced capabilities so that companies can analyze every factor to improve the bottom line.

Initial testing of DB2 10 shows that its many enhancements optimize the runtime environment by reducing CPU consumption by 5-10% with rebind for traditional workloads, with an additional possible 10% benefit using new functions, and up to 20% for new workloads. In addition, DB2 10 handles five to 10 times more concurrent users, up to 20,000 users, providing improvements in capacity, scalability, and overall performance for any type of large scale application.

Capacity, scalability, and performance continue to be DB2 for z/OS strengths. Through extensive exploitation of the System z 64-bit architecture, DB2 enhancements such as shorter optimized processing, using solid-state disk, in-memory work file enhancements, index insert parallelism improvements, and better SQL/XML access paths, provide many reductions in CPU costs and performance improvements without requiring any application changes.

DB2 10 features also enhance continuous business processing, database availability, and overall ease of use. DB2 is continuously available due to more options with online database changes, more concurrent utilities, and easier administration processes. DB2 10 database systems are always available. Transactions are processed even as changes are made to the database. These new database change capabilities combined with more options for parallel utility execution provide a smaller concurrent window for processing and administration tasks to streamline overall system operations.

Security, regulatory compliance, and audit capability improvements are also included in DB2 10. Enhanced security extends the role-based model introduced in DB2 9. DB2 10 provides granular authorities that separate data access from the administration of the application, database, and system. DB2 10 provides administration flexibility for specific security role settings, preventing data access exposure to unauthorized applications or administrators. This role-based security model, combined with the label-based row and column access control and masking or encryption of sensitive information, enhances the ultimate secure database environment for your business. All of these features provide tighter controls, allow more security flexibility, and provide tremendous regulatory and audit compliance capabilities.

Application integration and portability benefit from the DB2 10 SQL and XML enhancements. DB2 10 SQL improvements further embrace the DB2 family with more enhancements for porting other DBMS vendor products into the DB2 10 for z/OS environment. Additional enhancements with timestamps with time zones, Java timestamp compatibility, and timestamp 12-digit pico-second precision granularity provide unique business transaction timestamps for every transaction in a table. These enhancements help international companies understand all their businesses throughout all the global events of the day.

Data versioning with current and history tables is also available with DB2 10. Data versioning provides flexibility to migrate current operational data into a historical data table based on set time periods. Data versioning enhances the performance for the application operations and maintains historical data for audit purposes and for better integration into regulatory compliance architecture.

In addition, data warehouse and business intelligence capabilities are now built in directly to the DB2 10 with the new temporal table capabilities and the SQL capabilities for calculating moving sums and moving averages. These capabilities enhance the bottom line by integrating business intelligence capabilities into front-line operational applications and significantly reduce programming effort.

DB2 10 XML improvements help application flexibility, usability, and performance. DB2 10 provides a number of important XML performance and management improvements through the following capabilities:

- Replace, delete, or insert XML document nodes
- Manage multiple XML schema version documents
- Use the binary pre-tokenized XML format

Additional enhancements within utilities, date time data types, and XML parameter support within SQL functions and procedures provide application flexibility to use XML within any application architecture solution.

DB2 10 for z/OS delivers performance, scalability, availability, and security and also improves application integration and regulatory compliance. DB2 10 remains the technology leader and best choice database for business systems that seek a competitive advantage.

1.2 Benefits of DB2 10

DB2 10 can reduce the total DB2 CPU demand from 5-20% when you take advantage of all the enhancements. Many CPU reductions are built in directly to DB2, requiring no application changes. Some enhancements are implemented through normal DB2 activities through rebinding, restructuring database definitions, improving applications, and utility processing. The CPU demand reduction features have the potential to provide significant total cost of ownership savings based on the application mix and transaction types.

Improvements in optimization reduce costs by processing SQL automatically with more efficient data access paths. Improvements through a range-list index scan access method, IN-list list prefetch, more parallelism for select and index insert processing, better work file usage, better record identifier (RID) pool overflow management, improved sequential detection, faster log I/O, access path certainty evaluation for static SQL, and improved distributed data facility (DDF) F transaction flow all provide more efficiency without changes to applications. These enhancements can reduce total CPU enterprise costs because of improved efficiency in the DB2 10 for z/OS.

Other enhancements require new database definitions or application programming techniques to reduce overall costs. These enhancements, such as the Hash Access space and access method, including columns on a unique index, consolidating SQL by using the Attributes feature, and automatic data statistics, can reduce CPU costs through improved access paths.

DB2 10 includes numerous performance enhancements for Large Objects (LOBs) that save disk space for small LOBs and that provide dramatically better performance for LOB retrieval,

inserts, load, and import/export using DB2 utilities. DB210 can also more effectively REORG partitions that contain LOBs.

Improved operations and availability with DB2 10 can also reduce costs. DB2 10 can eliminate system downtime costs through the following capabilities:

- Improved memory management
- Ability to handle five to 10 times more users
- Ability to skip locked rows
- Improvements in the DB2 catalog
- More online schema change capabilities
- More online utilities

DB2 10 keeps the application available, even while more users, applications, database changes, and utilities are executing in the system.

More automated processes help install, configure, and simplify the installation of the DB2 10 system. Pre-migration installation steps capture existing settings and provide appropriate settings for the DB2 10 supplied routines and programs. These procedures can help reduce the installation, configuration, and testing time.

The ability to migrate directly from DB2 V8 also allows you to skip the implementation of DB2 9. Although using this method, you cannot take advantage of all the DB2 9 enhancements until you migrate to DB2 10. By skipping DB2 9, you do not have to install, test, and implement DB2 9 as an interim step to DB2 10. You can then use all the DB2 9 and DB2 10 enhancements after going to DB2 10 NFM. Many improvements are available in DB2 10 CM, regardless of the release from which you started the migration.

1.3 Subsystem enhancements

As with all previous versions, DB2 10 for z/OS takes advantage of the latest improvements in the platform. DB2 10 increases the synergy with System z hardware and software to provide better performance, more resilience, and better function for overall improved value.

There are several new functions that are generally related to the synergy of DB2 subsystems with the z/OS platform that can help improve scalability and availability. We describe those functions briefly in this section. Many enhancements are related to the removal of the 2 GB constraint in virtual storage for the DBM1 address space by allocating most of DB2 virtual storage areas above the 2 GB bar.

Verification of real storage and ECSA/ESQA requirements still applies.

Work file enhancements

Work files are required for sorting and materialization. DB2 10 reduces work file usage for very small sorts and adds support for large sorts and sparse index builds. These enhancements might improve performance and reduce the number of merge passes that are required for sort.

In the WORKFILE database, DB2 10 also supports the definition of partition-by-growth table spaces to support in-memory tables for improved performance. This capability provides a dedicated definition type for in-memory data and relieves the extra administration work of defining a single table space within dedicated buffer pools.

In addition, DB2 10 expands the record length size for work files to 65,529 bytes for handling larger record size answer sets like sorts and joins.

Buffer pool enhancements

System z has improved memory allocations, and DB2 10 takes advantage of this capability through its buffer pool management enhancements. In previous versions, DB2 allocated the defined size of the buffer pool at startup for use by all the associated table and index objects. In DB2 10, the system allocates the memory for the buffer pools as the data is brought into the buffer pool. This method keeps the buffer pool size at the minimum that is needed by applications.

In addition, DB2 10 reduces its latch contention and provides the ability to define larger buffer pools from current megabytes to gigabytes of memory for critical active objects. This capability can improve system I/O rates and reduce contention.

In data sharing environments, when an object changes its state to or from inter-system read/write interest, DB2 10 avoids scanning the buffer pool that contains that object, thereby saving CPU time when using large buffer pools.

System z10® 1 MB page size handles larger buffer pools better. Because buffer pool memory allocations can quickly become very large, DB2 10 can use the z10 1 MB page size and keep overall system paging down to a minimum.

Specialty engines

DB2 10 continues the trend initiated with DB2 V8 and provides additional zIIP workload eligibility in the following areas:

- DB2 10 parallelism enhancements

  More queries qualify for query parallelism which in turn introduces additional zIIP eligibility.

- DB2 10 RUNSTATS utility

  Portions of the RUNSTATS utility are eligible to be redirected to run on a zIIP processor. The degree of zIIP eligibility entirely depends upon what statistics that you gather.

- DB2 10 buffer pool prefetch activities

  Buffer pool prefetch, including dynamic prefetch, list prefetch, and sequential prefetch activities, is 100% zIIP eligible in DB2 10. Because asynchronous services buffer pool prefetch activities are not accounted to the DB2 client, they are reported in the DB2 statistics report instead.

- Deferred write activity

  DB2 bunches together, sorting by data set, the modified buffers that are accumulated in the buffer pool by updates and writes them out asynchronously. This I/O activity executed by DB2 for the applications is now a candidate for zIIP.

Extended 64-bit runtime environment exploited

DB2 10 improves scalability with exploitation of the 64-bit System z environment. Exploiting the 64-bit environment and moving 80-90% of the DB2 memory that is now below the bar—working storage, EDMPOOL, and even some ECSA—above the 2 GB bar eliminates the main memory constraints within the DB2 DBM1 address space for most systems.

Increase in the number of users, up to 20,000

By addressing the memory constraint in the overall system, virtual memory is no longer critical allowing five to 10 times more concurrent threads in a single DB2 10 member. This increase in threads removes one key reason for additional DB2 data sharing members and allows some consolidation of LPARs and members previously built for handling more users.

Archived

Improved data set open and close management

DB2 10 takes advantage of a new z/OS 1.12 interface to enable data sets to be opened more quickly and to maintain more open data sets.

Improved online schema changes

Some of the best features within DB2 10 are the online schema change features. Attributes on any DB2 table space, index, or table component are available with the ALTER statement. Within DB2 10, the attributes list is enhanced to handle the most common activities in an online ALTER and then REORG process. Being able to change almost any component of the common database attributes online provides administration flexibility and application availability for changes to your database systems.

You can use the following attributes of universal table spaces with the ALTER statement in DB2 10:

- Data set size
- Table or index page size
- Segment size
- Page size (buffer pool)

You can also migrate old style table space definitions to the new partition-by-growth or partition-by-range universal table spaces.

DB2 10 online schema enhancements improve adoption of the new attributes by recording the ALTER as a pending change and then executing the changes later through an online REORG process. The ALTER and REORG processes leave the database tables available for application activity, while the attribute changes are applied into the system. This method of enhancing the database eliminates downtime and provides relief for mission-critical, very large database availability issues.

The new ALTER and REORG method of applying changes introduces a database advisory state of AREOR to signify that attribute changes are pending. This database AREOR state, along with DB2 catalog tables, and the DROP PENDING CHANGES command, provides full functionality for managing ALTER and REORG processes.

As DB2 databases continue to grow in size and transaction volume, the amount of administration time that is required for database changes continues to be a challenge. A number of DB2 utility improvements further minimize downtime during normal operational using the FlashCopy® capability of the storage systems.

With DB2 10, DBAs can create or rebuild a non-unique index against tables with LOBs without any application impact or locking downtime. This online schema change enhancement can help with newly installed applications where another index can be defined quickly and can improve SQL access or resolve a performance issue. This enhancement alone can help improve performance for any installed application.

Universal range-partitioned table space

DB2 10 also continues to enhance the universal range-partitioned table space. This table space is the updated version of the classic range-partitioned table space with additional segment size specification, universal settings and capabilities. This universal range-partitioned table space is the migration target for the classic range partitioned table space that is deprecated and will be only supported for a few more DB2 releases.

Database administrators are encouraged to use the universal range-partitioned table space instead of the classic range-partition definitions to take advantage of the utility capabilities, availability, and performance features.

DB2 catalog enhancements

The DB2 catalog and directory are restructured removing special structures and links. Through this restructuring, the DB2 catalog now uses the new universal partition-by-growth table spaces, reordered row format, and one table per table space, thus expanding the number of DB2 catalog table spaces. These universal table spaces are defined with DSSIZE 64 MAXPART 1, row level locking, and some CLOB and BLOB data types to handle repeating long strings. These common table space definitions allow you to manage the catalog tables to in a similar manner to the application database with online reorganizations and check utilities.

In addition to these enhancements, the UTSERIAL lock that caused lock contention with older versions of DB2 utility processing is eliminated. This improvement, along with a reduction in log latch contention through compare and swap logic, the new option of readers to avoid waiting for inserters, and improvements in system thread latching serialization, can help reduce many types of DB2 thread processing contention. All of these enhancements help with concurrency of DDL, BIND, utility, and overall processing within application database systems and especially with processes that reference the DB2 catalog.

Another enhancement provides the ability to add a new log into the system inventory while the subsystem is active. The newly added log is available immediately without recycling DB2, which can help recovery procedures and application performance.

MEMBER CLUSTER option

DB2 tries to maintain the table space clustering sequence when data is inserted into the table. For data-sharing systems with robust INSERT processing, maintaining the clustering sequence can cause locking contention within the table space across the different data-sharing members. This contention causes extra CPU cycles to negotiate deadlocks and extended response time to maintain the clustering sequence.

DB2 10 partition-by-range and partition-by-growth table spaces now support the MEMBER CLUSTER parameter that allows DB2 to allocate space for insert disregarding the clustering sequence of the table space. This method relieves the contention, but you need to monitor the clustering sequence of the table space to avoid additional I/O for other high performance applications.

1.4 Application functions

The standard and customizable capabilities that are now available in DB2 10 provide advanced business analytics and fertile ground for customization. Built-in pureXML, LOB, and open SQL scalar and table function interfaces provide many functions and capabilities that can be extended to any type of custom functionality for business requirements.

Temporal tables and their ability to archive historical data automatically provide active data management to help improve performance, audit, and compliance capabilities. Coupled with SQL, you can now use business time or system time SQL parameters to qualify the answer, DB2 10 provides unique, industry-leading database advanced business analytical capabilities that are easy to implement without significant coding effort.

DB2 for z/OS continues to provide functions that can help with the deployment of a data warehouse on System z.

DB2 9 for z/OS provides support for XML data type through the pureXML capabilities and hybrid database engine. DB2 10 for z/OS expands such XML support, including more functionality and performance improvements.

Greater timestamp precision for Java applications

DB2 10 enhances the TIMESTAMP data type with greater precision. The new TIME ZONE sensitive capability provides more compatibility and functionality for all types of applications. The TIMESTAMP precision enhancement supports up to 12 digits of fractional seconds (picoseconds) with the default matching the Java default of 6 digits precision of fractional seconds. The 6 digits default also helps Java functionality and DB2 family compatibility along with SQL Server compatibility. The enhanced CURRENT TIMESTAMP uses a special register so that applications can specify the desired fractional precision for application requirements.

Support for TIMESTAMP WITH TIMEZONE

The TIMESTAMP WITH TIMEZONE data type incorporates the TIMESTAMP 12-digit fractional seconds capabilities and uses the new industry-standard Universal Coordinated Time (UTC), replacing Greenwich Mean Time (GMT). This support provides applications with additional TIMEZONE capabilities to compare business divisions along the exact timeline throughout the world, which is vital for global financial, retail, and banking systems.

Extended indicator variable

DB2 10 introduces an extended indicator variable that provides a way to specify that there is no value provided for an INSERT, UPDATE, and MERGE statement column. As the extended indicator variable name implies, this variable extends the functionality within an indicator variable for providing values within an SQL statement.

For example, a value of -5 within an enabled extended indicator variable specifies the DEFAULT value. If the extended indicator variables are not enabled on the SQL package, then the -5 specifies a NULL value. If the extended indicator value is enabled and given a value of -7, this setting indicates the variable is to be UNASSIGNED, ignored, and treated as though it did not exist within the SQL statement.

These extended indicator variables are typically for Java applications and are quite useful for dynamic statements and variable SQL statement coding where the number of possible host variable parameters is unknown until the transaction logic is completed and the SQL is to be executed.

This capability addresses the application issue that previously required multiple SQL statements coded to match the values that were available for the SQL statements. Now, these multiple SQL statements can be consolidated. When the column value is not known, the host variable value can use the keyword UNASSIGNED for the appropriate column or columns. This feature is especially important for applications that use dynamic statements that are clogging up the system's Dynamic Statement Cache with many copies of essentially the same SQL statements.

Extended support for implicit casting

Implicit casting is the automatic conversion of different types of data to be compatible. DB2 enhances its implicit casting by handling numeric data types that can be cast implicitly to character or graphical string data types. It also supports converting the data in the other direction, from character or graphical string data types to numeric data types.

In previous releases of DB2, you had to do this casting manually, which could be a labor-intensive application process. Now, numeric data, character, and graphical string data can be handled, compared, and assigned implicitly. DB2 for z/OS is more compatible and enhances portability of SQL from other database vendor systems.

Enhanced scalar function support

DB2 10 enhances its compatibility with other database vendors with improvements in SQL scalar and table functions. These built-in functions are used throughout application SQL, making quick work of OLAP functions and calculations, such as SUM, AVG, SIN, COS, and other functions. As in previous releases, these inline functions with their SQL statements return a single value.

Non-inline SQL scalar functions that contain logic provide additional application functionality and flexibility. This flexibility helps DB2 family compatibility and acceptance of data migrations from other database vendors. DB2 also supports multiple versions and source code management of these functions based on their parameter list, routine options, and function body. These functions can be altered or replaced with different versions that are distributed to multiple servers to assist in testing and overall performance. Function version fallback to a previous version occurs instantly without a rebind or recompile when a function version is dropped.

DB2 10 introduces support for SQL user-defined table functions that help ease application migrations from other database vendors. DB2 table functions are flexible because they return a single data table result that is based on the many different type of parameters, such as LOBs, distinct types, and transition tables.

SQL procedural language enhancements

DB2 9 provides native support for the SQL procedural language, eliminating the requirement to generate a C program from the SQL procedure that then executes as an external stored procedure. DB2 9 SQL procedures can be executed natively within the DB2 engine for better runtime execution and stored in the DB2 catalog for better management and version control. Running the native code within the DB2 engine also helps in debugging, deploying, and managing SQL procedural versions throughout multiple servers. Storing the SQL procedures improves the overall change control of this application code so that you can manage it as you do other application developer modules.

The SQL procedural language enhancements include SQL Table functions, nested compound SQL statements within a procedure, and the RETURN statement that can return the result set of a SELECT SQL statement. These stored procedures and other SQL procedural language enhancements allow all types of processing.

The DB2 10 SQL procedural language enhancements provide needed compatibility with other database vendors. The procedure language is enhanced to accept many data types and XML as parameters and to provide limited use of scrollable cursors. The DB2 10 concurrency improvements and SQL procedural language compatibility provide the opportunity to migrate other relational database management solutions to the z/OS environment for a better cost-of-ownership experience, while providing the unique performance, availability, and scalability capabilities that only DB2 for z/OS and System z offer.

Support for OLAP: Moving sums, averages, and aggregates

The OLAP capabilities of moving sums, averages, and aggregates are now built in directly to DB2. Improvements within SQL, intermediate work file results, and scalar or table functions provide performance for these OLAP activities. Moving sums, averages, and aggregates are common OLAP functions within any data warehousing application. These moving sums, averages and aggregates are typical standard calculations that are accomplished using different groups of time-period or location-based data for product sales, store location, or other common criteria.

Having these OLAP capabilities built in directly to DB2 provides an industry-standard SQL process, repeatable applications, SQL function or table functions, and robust performance through better optimization processes.

These OLAP capabilities are further enhanced through scalar, custom table functions or the new temporal tables to establish the window of data for the moving sum, average, or aggregate to calculate its answer set. By using a partition, time frame, or common table SQL expression, the standard OLAP functions can provide the standard calculations for complex or simple data warehouse requirements. Also, given the improvements within SQL, these moving sums, averages, and aggregates can be included in expressions, select lists, or ORDER BY statements, satisfying any application requirements.

Bi-temporal queries and their business advantages

DB2 10 provides temporal data functionality, often referred to as *time travel queries*, through the BUSINESS_TIME and SYSTEM_TIME table period definitions. These period definitions are used for temporal table definitions to provide system-maintained, period-maintained, or bi-temporal (both system and period maintained) data stores. These temporal data tables are maintained automatically, and when the designated time period criterion is met, the data is archived to an associated history table.

The PERIOD SYSTEM_TIME or PERIOD BUSINESS_TIME definition over two columns defines the temporal period for the data within the table:

- The SYSTEM_TIME relates to the time the data was put into the system.
- The BUSINESS_TIME relates to the business transaction or business relevant time period of the data.

These definitions control the criteria for which data exists in the table and when it is migrated to the associated history table. By using both definitions, PERIOD SYSTEM_TIME and PERIOD BUSINESS_TIME, a table has bi-temporal criteria that control the data that exists in the table.

With the BUSINESS_TIME WITHOUT OVERLAPS definition parameter, the temporal tables can make all transaction time stamps unique using the TIMESTAMP picosecond precision (precision 12) enhancements to provide unique transaction time stamps throughout the temporal table. This capability provides an advantage for robust global systems that might have issues with the uniqueness of business timestamp transactions.

Access to currently committed data

With five to 10 times more concurrent threads within a single DB2 member, DB2 10 focuses a significant amount of enhancements on application concurrency. DB2 10 provides an individual package option for managing concurrency within applications. This enhancement provides a DB2 package level BIND parameter to let you choose the way applications handle data concurrency situations. DB2 10 now provides the following parameters and settings:

- DB2 10 introduces the CURRENTACCESSRESOLUTION parameter with the USECURRENTLYCOMMITTED and WAITFOROUTCOME settings. This parameter setting overrides the DB2 subsystem parameters of EVALUNC and SKIPUNCI and helps the application package quickly perform the desired concurrency action.
- The USECURRENTLYCOMMITTED setting instructs the system to ignore rows that are in the process of being inserted and use only currently committed rows. This clause is contingent on the package BIND isolation level settings being either Cursor Stability or Read Stability.

- The WAITFOROUTCOME setting specifies that applicable scans must wait for a commit or rollback operation to complete when data is in the process of being updated or deleted. Rows that are in the process of being inserted are not skipped.

These different settings provide the application with the flexibility to handle highly concurrent web transactions, wait or use uncommitted data. These different parameter settings help provide the desired package level of concurrency and also provide capabilities that mimic some other database vendor's application concurrency settings.

Plan stability, package preservation

The plan stability in DB2 9 offers a way to handle testing of a new version of a DB2 application package. Plan stability offers a way to preserve multiple package copies and allows you to switch back to a previous copy of the bound package. If the access path of the current package is not as efficient, package stability allows the administrator to switch back to a previous version of the package with a simple REBIND SWITCH command to the old version of the package

DB2 10 for z/OS delivers a new framework that is geared towards the support of the existing and future features that pertain to query access path. The access path repository holds query text, query access paths, and optimization options. The repository supports versioning by maintaining copies of access paths and associated options for each query.

pureXML enhancements

Within DB2 10, XML can be used almost anywhere within SQL variables, scalar functions, SQL table functions, and SQL procedures. DB2 10 pureXML incorporates many enhancements that improve overall XML performance, that provide easier XML schema management, and that embrace DB2 family compatibility.

These enhancements start with XML schema validation that is now built in to DB2 10. The XML schema no longer needs to be specified, because DB2 handles XML schema validation more easily through a built-in function that determines the XML schemas from the instance documents. DB2 uses the schema registration timestamp and schema location hints to match the XML document to the correct schema version. This function allows multiple schema versions to coexist and validation of new or older XML documents against their appropriate XML schema versions.

Additional functionality enhancements provide the capability to manipulate any part of an XML document. By using SQL statements with XQuery updating expressions, any single or multiple XML document nodes can be inserted, updated, or deleted or can have their data values updated. This function provides tremendous XML document capabilities, overall performance, and flexibility for any application process.

An additional XML type modifier is also available with DB2 10. The XML type modifier adds constraints on XML column data and enforces and validates the XML column data against the schema definition information. This XML type modifier can be ALTERed onto older XML schemas so that their XML column types can be validated. This process helps to ensure that the XML schema documents stored elements have only the desired XML content.

DB2 10 adds XML date and time types. These data types are supported within XML indexes, and the timestamp is expanded to handle more precision for finer data management. DB2 10 also includes XML time and date arithmetic comparison functions to further support application processing.

DB2 10 improves XML performance with support for binary XML objects. Binary support is better for server and application interaction. It uses pre-tokenized format and length

definitions that can improve overall performance and provide additional ease of use for Java applications.

Binary XML also has additional flexibility features, such as String IDs, for text that represents some or all occurrences of the same text with an integer identifier. This capability can help reduce the size of XML, thus improving application performance.

DB2 10 provides a new LOB_INLINE_LENGTH installation parameter that sets the default number of bytes for storing inline LOBs. Having a minimized LOB length or a predefined standard length provides better I/O capabilities and buffering and optimizes the usage of the inline LOB space.

The administrators now have the option of delaying the definition of the LOB or XML data sets and their indexes, which can help save storage space by defining them into the DB2 catalog and letting application SELECT and FETCH them. The LOB or XML data sets and their indexes are allocated only when the first insert is issued, saving storage and application performance until the data is saved in the database (especially useful for vendor packages).

The administrator also can use the CHECK DATA utility to check the consistency between the XML schema, its document data, and its XML index data.

1.5 Operation and performance enhancements

Technical innovations in operational compliance help with security and enhanced auditing capabilities.

DB2 Utilities Suite for z/OS delivers full support for enhancements in DB2 10 for z/OS as well as integration with the storage systems functions, such as FlashCopy, exploitation of new sort options (DB2 Sort), and backup and restore.

Performance is enhanced with DB2 10 for z/OS, offering reduced regression and opportunities for reduced CPU time. For relational database customers, database performance is paramount. DB2 10 for z/OS can reduce CPU demand 5-10% immediately with no application changes. CPU demand can be further reduced up to 20% when using all the DB2 10 enhancements in new-function mode (NFM). By pushing the performance limits, IBM and DB2 10 continue to lead the database industry with state-of-the-art technology and the most efficient database processing available.

Security and regulatory compliance

DB2 10 also includes security, regulatory compliance, and audit capability improvements. DB2 10 enhances the DB2 9 role-based security with additional administrative and other finer-grained authorities and privileges. This authority granularity helps separate administration and data access that provide only the minimum appropriate authority.

The SECADM authorization level provides the authority to manage access to the tables while being prohibited from creating, dropping, or altering the tables. The enhanced DBADM authority provides an option to have administration capabilities without data access or without access control. These authority profiles provide better separation of duties while limiting or eliminating blanket authority over all aspects of a table and its data.

In addition, DB2 10 embraces audit and regulatory compliance through a new audit policy that provides a set of criteria for auditing for the possible abuse and overlapping of authorities within a system. This feature helps management, administrators, and the business community understand, configure, and audit security policies and data access quickly for any

role or user. Many audit policies can be developed to quickly verify audit and document the security compliance across the environment's critical data resources and application users.

Support for row and column access control

DB2 10 also enhances security through its row and column access control. This access control lets administrators enable security on a particular column or particular row in the database. Such a security method complements the privilege model. After you have enforced row or column level access control for a table, any SQL DML statement that attempts to access that table is subject to the row and column access control rules that you defined for that table. During table access, DB2 transparently applies these rules to every user.

This capability allows you to define fine-grained security against any data. The role-based security model, combined with the row and column access control and masking or encryption of sensitive information, enhances the ultimate secure database environment for your business. These features provide tighter controls, allow more security flexibility, and provide tremendous regulatory and audit compliance capabilities.

Real time statistics stored procedures

Because the optimizer access paths improve performance, up-to-the-moment statistics are vital. DB2 10 includes a set of stored procedures to monitor and collect table and index statistics. These procedures monitor the current statistics, determine whether statistics need to be collected, and then perform the collection autonomically to ensure good access path optimization.

These procedures especially help volatile environments and can dynamically improve access path optimization by getting index filtering statistics for SQL WHERE clause predicates to make the best access path decisions. By gathering statistics for you, these DB2 10 stored procedures take the burden off the administrators for large and especially dynamically created objects to help ensure overall application performance.

Skip level migration

DB2 10 for z/OS provides the capability to migrate not only from DB2 9 NFM but also directly from DB2 V8 NFM. The process of migrating directly from DB2 V8 NFM to DB2 10 CM is called *skip level migration*. This capability affords you greater flexibility as to how and when you can migrate to DB2 10. The actual skip level migration process itself is rather simple and allows you to jump directly from DB2 V8 NFM to DB2 10 CM without having any intermediate steps; however, the planning phase and learning curve must take into account new functions and restrictions for both releases.

Performance is pervasive throughout the product

IBM DB2 10 for z/OS with its CPU reduction for applications and many new-function mode enhancements that are ready for immediate use delivers the best performance improvements since DB2 Version 2.1. DB2 10 emphasis on performance is pervasive throughout the features and enhancements. Availability, scalability, security, compliance, application integration, XML and SQL all contain performance improvements. These enhancements provide an improved operational environment, making administration easier and reducing the total cost of ownership for the business.

Performance is the major emphasis of DB2 10. Many enhancements make direct reduction in CPU time and suspension time for applications. By migrating to the new version and deploying and binding your applications within the environment, your applications can take advantage of many of the performance enhancements without changing the application. DB2 10 uses the full 64-bit architecture, optimization of new access paths and provides performance improvements that are independent of access path choice.

Rebinding your applications is now even easier. Concerns about regression are easily addressed with plan stability, introduced in DB2 9 and further enhanced in DB2 10. DB2 10 plan stability with fallback, and package level DSNZPARM settings provide administrators the ability to rebind and reduce performance risk.

Optimizer access path range-list index scan

DB2 10 offers a new application access path called range-list index scan. This access path improves SQL processing against an index when multiple WHERE clauses can all reference the same index. For example, when the SQL has multiple OR conditions that reference the same index, the optimizer now recognizes it and scans the index only once, maintaining the index order, instead of multiple index access that requires a final sort to reinstate order. This process cuts down the number of RID list entries for the processing, which can improve I/O and CPU performance.

You can use this access path when all the SQL WHERE clauses with multiple OR statements reference the same table, every OR predicate has at least one matching predicate and is mapped to the same index. This type of SQL WHERE clause with multiple OR statements is typical for many types of applications, especially searching, scrolling, or pagination application processes.

Optimizer uses more parallelism

DB2 10 also improves several existing access paths through parallelism. These specifically designed enhancements eliminate some previous DB2 restrictions, increase the amount of work redirected to the zIIP processors, and distribute work more evenly across the parallel tasks. These enhancements provides additional reasons to enable parallelism within your environment.

Parallelism improves your application performance and DB2 10 can now take full advantage of parallelism with the following types of SQL queries:

- Multi-row fetch
- Full outer joins
- Common table expressions (CTE) references
- Table expression materialization
- A table function
- A CREATE GLOBAL TEMPORARY table (CGTT)
- A work file resulting from view materialization

These new DB2 10 CP parallelism enhancements are active when the SQL Explain PLAN_TABLE PARALLELISM_MODE column contains a “C.”

The new parallelism enhancements can also be active during the following specialized SQL situations:

- When the optimizer chooses index reverse scan for a table
- When a SQL subquery is transformed into a join
- When DB2 chooses to do a multiple column hybrid join with sort composite
- When the leading table is sort output and the join between the leading table and the second table is a multiple column hybrid join

Additional DB2 10 optimization and access improvements also help many aspects of application performance. In DB2 10, index lookaside and sequential detection help improve referencing parent keys within referential integrity structures during INSERT processing. This process is more efficient for checking the referential integrity dependent data and reduces the overall CPU required for the insert activity.

List prefetch improves index access

List prefetch is used more within DB2 10 to access index leaf and non-leaf pages. In previous versions of DB2, when the index became disorganized and had large gaps between non-leaf pages, accessing index entries through sequential reading of non-leaf pages became degraded by huge numbers of synchronous I/Os. DB2 10 improvements use non-leaf page information to perform a list prefetch of the leaf pages. This function eliminates most of the synchronous I/Os and the I/O waits that are associated to the large gaps in the non-leaf pages during the sequential read. This list prefetch processing especially helps long running queries that are dependent on non-leaf page access and also helps all the index-related utilities, such as partition-level REORG, REORG INDEX, CHECK INDEX, and RUNSTATS.

Improved sequential detection

When DB2 uses an index that has a high cluster ratio and has to then fetch the rows that are qualified by the index scan, it typically uses dynamic prefetch on the data. Dynamic prefetch has a sequential detection algorithm whose objective is to avoid synchronous I/Os for clustered pages. DB2 10 makes this algorithm more robust, resulting in fewer synchronous I/Os as the cluster ratio degrades below 100%.

Improved log I/O

DB2 10 reduces the number of log I/Os and provides better I/O overlap when using dual logging.

Optimizer no longer likely to change access path because of RID Pool overflows

DB2 10 also improves the handling of SQL statements that reference a large amount of data through list processing. This list processing uses a large number of record IDs (RIDs) and sometimes overflows the RID pool. In previous versions of DB2, the RID pool overflow caused DB2 to change the SQL access method to a table space scan. Now, when the RID pool resources are exhausted, the RID list is written to work file resources and processing continues. This improvement helps avoid the table space scan with its associated elapsed time, locking impact and performance overhead.

Optimizer does more during stage 1 SQL evaluation

The DB2 10 optimizer can now evaluate scalar functions and other stage 2 predicates during the first (stage 1) evaluation of the SQL access path. For indexed columns, this means the optimizer can apply these stage 2 predicates as non-matching screening predicates to potentially reduce the number of data rows that need to be accessed. This process eliminates or reduces the amount of data evaluated in stage 2, thus improving query elapsed time and overall query performance.

Optimizer determines which access is more certain

In previous versions of DB2, the optimizer evaluates the SQL WHERE predicate, the table indexes available, and various statistics to determine the most efficient access path. With enhancements in DB2 10, the optimizer analyzes additional filter factor variables evaluating the SQL range predicate, the non-uniform distribution of the data, usage of parameter markers, host variables, or literals where values are unknown.

When choosing between two different indexes there are situations where these unknown filter factor variables make the cost estimate between two different access paths close. Analyzing which of the different filter factor variables are known, DB2 determines which of the index access paths has a higher degree of certainty. DB2 can choose an index access with a slightly higher cost that provides a more certain runtime performance. This analysis is especially important to provide the best consistent reliable performance with the different

types of programming languages and the application diversity of the parameter markers, literals, and host variables available.

Dynamic statement cache ATTRIBUTES improvements

One of the most important DB2 10 system improvements is the ability to combine some variations of SQL within the dynamic statement cache. Using the new ATTRIBUTES clause within the PREPARE SQL statement, DB2 10 can recognize that the SQL is the same except for the WHERE clause literal values. This process helps DB2 to recognize that these statements already exist within the cache and to reuse the cache resources that were generated previously for the SQL statement, which can help to avoid additional DB2 catalog activity, such as object verification and access path creation for another SQL statement. This process also helps free more cache space for other SQL statements for reuse, which can improve performance and transaction response time.

Improved DDF transaction flow

Application performance and network transaction traffic is optimized when SELECT statements are coded using the FETCH 1 ROW ONLY clause. DB2 now recognizes this SELECT statement FETCH 1 ROW ONLY clause and combines the API calls and messages, reducing the number of messages flowing across the network through the system.

The change to the FETCH 1 ROW ONLY clause also improves the performance of the JDBC and CLI APIs. After the query data is retrieved, the FETCH 1 ROW ONLY clause causes the API's default action for DB2 to close the resources. DB2 closes the resources regardless of whether a CURSOR WITH HOLD was declared and notifies the API driver to cancel any additional FETCH or CLOSE statement requests. This process reduces the number and amount of transaction network messages transmitted, improves DB2 performance, and minimizes locking contention.

Hash space and access method

DB2 10 also introduces an access type called *hash access*, which is supported by a hash space. DB2 uses an internal hash algorithm with the hash space to reference the location of the data rows. In some cases, this direct hash access reduces data access to a single I/O, decreasing the CPU workload and speeding up application response time. Queries that use full key equal predicates, such as customer number or product number lookups, are good candidates for hash access. You can create additional indexes to support other range, list, or keyed access types.

The definition of the hash space requires column or columns for the direct hash access keys. Each table with hash access has an associated hash space or hash space partitions. Hash access requires some additional storage space to reduce access CPU workload.

There are tradeoffs for using hash access. Parallelism is not available, and traditional clustering is not allowed for the hash data. Nevertheless, hash access can be beneficial for database designs where unique keys are already using equal predicates on product or customer IDs, object IDs, XML document IDs, and other direct key retrievals.

Smarter insert page candidate selection algorithm

DB2 10 modifies the algorithm to choose a page to insert a new row when a key does not match any existing key in the cluster index. Instead of choosing the page of the next highest key, DB2 10 chooses the page of the next lowest key. When inserting a series of rows with a sequentially increasing key at the same insertion point of the cluster index, the new algorithm leads DB2 to pick the same page where free space was previously found. This algorithm tends to reduce the number of getpages needed to locate free space while at the same time maintaining good clustering.

INCLUDE non-unique columns within a unique index

One of the schema changes that your applications need right away is the ability to include more columns into unique indexes. This DB2 10 feature allows you to include non-unique columns in the definition of a unique index definition. Before this enhancement, you needed one index for the unique constraint and another index for the non-unique columns. Using the new CREATE or ALTER INCLUDE clause, a unique index definition can include additional columns in the definition of the index. This capability eliminates all the extra I/O spent maintaining the other index along with the additional storage needed for multiple index definitions with similar columns and improves performance for all access to the table.

Parallel inserts into multiple indexes

DB2 10 also improves insert performance by using more parallelism. When INSERT SQL modifies a table with multiple indexes, DB2 10 does the prefetch of multiple indexes in parallel. By initiating parallel I/Os for the multiple indexes, the process is not waiting for the synchronous I/Os, reducing the overall insert process time. This method reduces the time frame of possible contention within the system and improves performance of all applications.

Utilities enhancements

Utilities support for RECFM=VBS and a more widespread and integrated use of FlashCopy technology offer better performance and more availability.

Data compression

A new DSNZPARM provides the ability to compress DB2 SMF trace data. This function helps minimize the cost of disk space that is needed to save valuable accounting information.

There is no need to run REORG to create a dictionary to apply compression to table spaces. Insert can do that dynamically.

Archived

Archived

Part 1

Subsystem

In this part, we describe functions generally related to the DB2 subsystem and the z/OS platform.

This part includes the following chapters:

- Chapter 2, “Synergy with System z” on page 3
- Chapter 3, “Scalability” on page 51
- Chapter 4, “Availability” on page 67
- Chapter 5, “Data sharing” on page 109

© Copyright IBM Corp. 2010. All rights reserved.

Archived

2

Synergy with System z

As with all previous versions, DB2 10 for z/OS takes advantage of the latest improvements in the platform. DB2 10 increases the synergy with System z hardware and software to provide better performance, more resilience, and better function for an overall improved value.

DB2 benefits from large real memory support, faster processors, and better hardware compression. DB2 uses the enhanced features of the storage servers, such as the IBM System Storage® DS8000®. FlashCopy is used by DB2 utilities, allowing higher levels of availability and ease of operations. DB2 makes unique use of the z/Architecture® instruction set for improvements in reliability, performance, and availability. DB2 continues to deliver synergy with hardware data compression, Fibre Channel connection (FICON®), channels, disk storage, advanced networking function, and Workload Manager (WLM).

In this chapter, we discuss the following topics:

- Synergy with System z in general
- Synergy with IBM System z and z/OS
- Synergy with storage
- z/OS Security Server
- Synergy with TCP/IP
- WLM
- Using RMF for zIIP reporting and monitoring
- Warehousing on System z
- Data encryption
- IBM WebSphere DataPower
- Additional zIIP and zAAP eligibility

© Copyright IBM Corp. 2010. All rights reserved.

2.1 Synergy with System z in general

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

In this chapter, we describe the efforts that have been made in DB2 10 to more fully exploit synergy potential between the System z hardware and software to remove constraints for growth, to improve reliability and availability, to continue to improve total cost of ownership, and to improve performance across the board. We furthermore outline features and functions of the IBM zEnterprise platform and z/OS V1R12 that are expected to benefit DB2 for z/OS.

2.2 Synergy with IBM System z and z/OS

In this section, we discuss interfaces that are passively or actively used by DB2 10 to fully exploit the synergy potential between the System z hardware and the z/OS operating system software.

2.2.1 DBM1 64-bit memory usage and virtual storage constraint relief

For many years, virtual storage has been the most common constraint for large customers. Prior to DB2 10, the amount of available virtual storage below the bar potentially limited the number of concurrent threads for a single data sharing member or DB2 subsystem. DB2 8 provided the foundation for virtual storage constraint relief (VSCR) below the bar and moved a large number of DB2 control blocks and work areas (buffer pools, castout buffers, compression dictionaries, RID pool, internal trace tables, part of the EDM pool, and so forth) above the bar. DB2 9 provided additional relief of about 10-15%. Although this level of VSCR helped many customers to support a growing number of DB2 threads, other customers had to expand their environments horizontally to support workload growth in DB2, by activating data sharing or by adding further data sharing members, which added complexity and further administration to existing system management processes and procedures.

DB2 10 for z/OS provides a dramatic reduction of virtual private storage below the bar, moving 50-90% of the current storage above the bar by exploiting 64-bit virtual storage. This change allows as much as 10 times more concurrent active tasks in DB2. Customers need to perform much less detailed virtual storage monitoring. Some customers can have fewer DB2 members and can reduce the number of logical partitions (LPARs). The net results for DB2 customers are cost reductions, simplified management, and easier growth.

Archive

Figure 2-1 illustrates the virtual storage constraint relief that is provided by DB2 10.

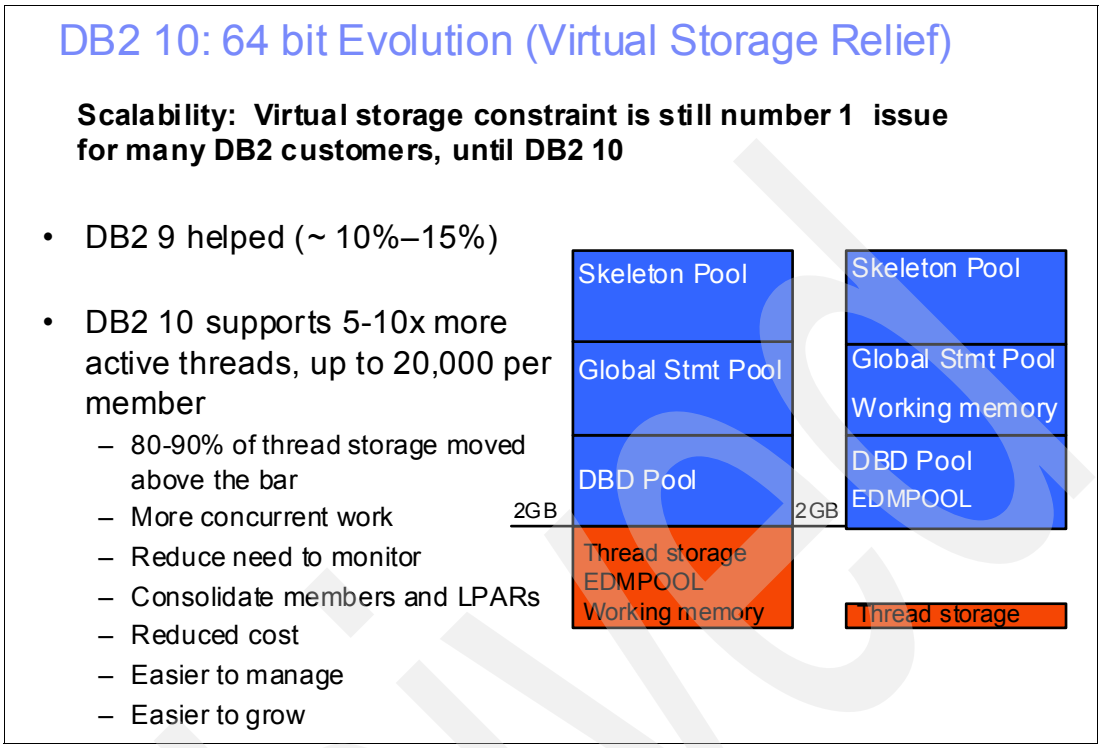


Figure 2-1 DB2 9 and DB2 10 VSCR in the DBM1 address space

For more information about VSCR in DB2 10, refer to 3.1, “Virtual storage relief” on page 52.

2.2.2 ECSA virtual storage constraint relief

Because of the DBM1 VSCR delivered in DB2 10 (see 2.2.1, “DBM1 64-bit memory usage and virtual storage constraint relief” on page 4), it becomes more realistic to think about data sharing member or LPAR consolidation. DB2 9 and earlier versions of DB2 use 31-bit extended common service area (ECSA) to share data across address spaces. If several data sharing members are consolidated to run in one member or DB2 subsystem, the total amount of ECSA that is needed can cause virtual storage constraints to the 31-bit ECSA.

To provide virtual storage constraint relief (VSCR) to that situation, DB2 10 uses more often 64-bit common storage instead of 31-bit ECSA. For example, in DB2 10 the instrumentation facility component (IFC) uses 64-bit common storage. When you start a monitor trace in DB2 10, the online performance (OP) buffers are allocated in 64-bit common storage to support a maximum OP buffer size of 64 MB (increased from 16 MB in DB2 9).

Other DB2 blocks were also moved from ECSA to 64-bit common. Installations that use a lot of stored procedures (especially nested stored procedures) might see a reduction of several megabytes of ECSA.

2.2.3 Increase of 64-bit memory efficiency

Every access to an address space memory frame must go through a virtual-to-real mapping contained in page table entries (PTEs). If the mapped address is cached on the translation look-aside buffer (TLB), which is a hardware cache on the processor, the mapping process is

efficient. If there is a TLB cache miss, processing must stop and wait for a search in main memory to search for the entry and place it on the TLB.

Figure 2-2 illustrates the address translation for 4 KB pages.

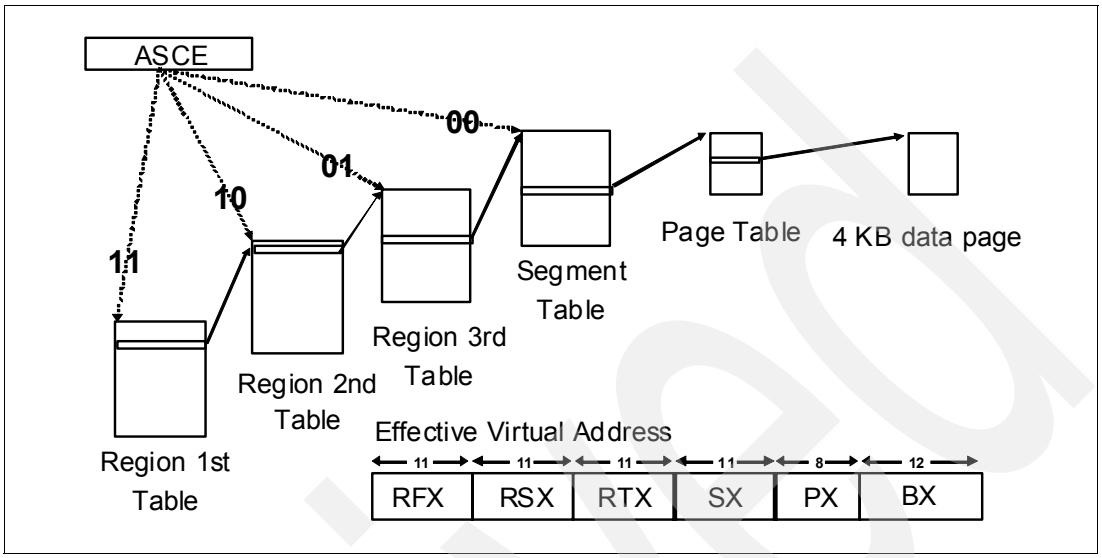


Figure 2-2 Address translation for 4 KB pages

The address space control element (ASCE) describes an address space with Region Table or Segment Table Origin or Real Space Token as follows:

- A virtual space that is described by translation tables
- The Real Space for which virtual addresses are translated to the same identical real address with no translation tables

In this case, the effective virtual address can be the result of up to the following levels of dynamic address translation tables searches:

- Region First (indexed by RFX)
- Region Second (indexed by RSX)
- Region Third (indexed by RTX)
- Segment (indexed by SX)
- Page (indexed by PX) Translation can start at R1T, R2T, R3T, or SGT

The starting point of the translation is designated in the Address Space Control Element.

Recently, application memory sizes have increased dramatically due to support for 64-bit addressing in both physical and virtual memory. However, translation look-aside buffer (TLB) sizes have remained relatively small due to low access time requirements and hardware space limitations. A z10 hardware can cache 512 TLB entries per processor. The TLB coverage in today's applications represents a much smaller fraction of an application's working set size leading to a larger number of TLB misses. An application can suffer a significant performance penalty, resulting from an increased number of TLB misses and the increased cost of each TLB miss.

The workload characteristics where large pages are beneficial must include:

- A large DBM1 address space (for example, large buffer pools)
- High user concurrency
- A high GETPAGE rate

In addition to these characteristics, to see the benefits of large pages, the workload should be well-tuned and mainly CPU-bound (not I/O bound); otherwise, potentially bigger performance issues can overshadow any benefits of making virtual memory management more efficient.

z/OS V1R10 introduced 1 MB real memory frames. DB2 10 uses 1 MB real memory frames for buffer pools that are defined with the PGFIX(YES) attribute. Using 1 MB memory frames for PGFIX(YES) buffer pools is a perfect match with the 1 MB page frames attribute because 1 MB frames are not paged out by z/OS.

Using 1 MB memory frames is intended to reduce the burden of virtual to real mapping memory management in z/OS by reducing the number of TLB entries that are required for DB2 buffer pool pages. The use of 1 MB pages in DB2 for z/OS increases the TLB coverage without proportionally enlarging the TLB size. This results in better performance by decreasing the number of TLB misses that DB2 might incur.

To illustrate the difference between using 4 KB and 1 MB memory page frames, let us assume that we have defined a 5 GB address space for DBM1, where the buffer pool is allocated. With 4 KB page frames, there are 1.3 million mapping entries. If z/OS uses 1 MB pages, there are only 5,120 entries.

Figure 2-3 illustrates the address translation for 1 MB pages.

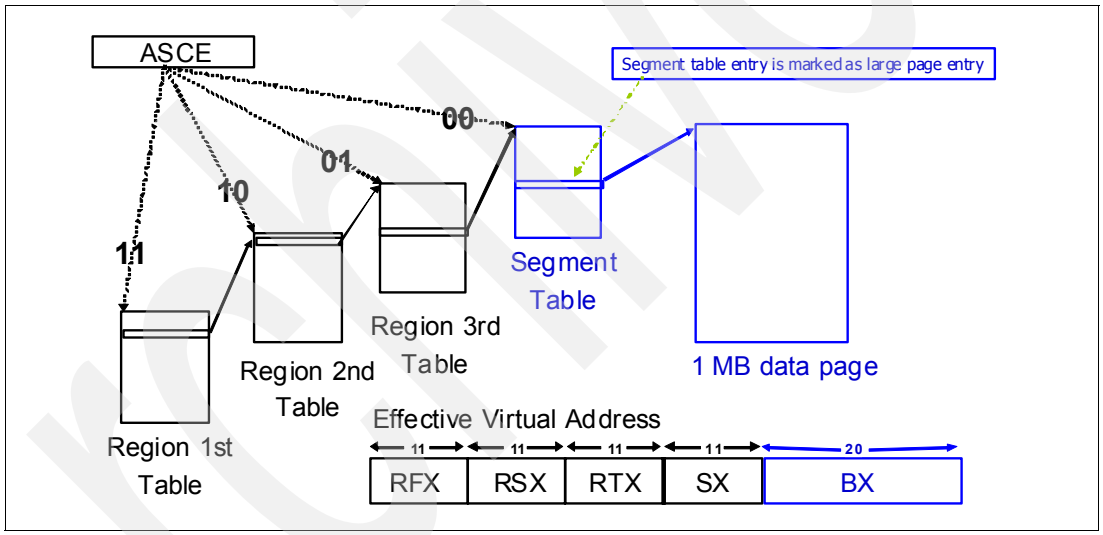


Figure 2-3 Address translation for 1 MB pages

For more information about how DB2 10 uses 1 MB page frames for PGFIX(YES) buffer pools, refer to 13.8.1, “Buffer storage allocation” on page 561.

2.2.4 Improved CPU cache performance

In the past, System z hardware and software have remained relatively independent of each other. The use of faster processors and large 64-bit memory require a closer cooperation between hardware and software. With the System z10 architecture, DB2 for z/OS operates in an environment that supports more and faster processors with larger amounts of real and virtual storage.

In such a fast processing environment, CPU cache misses can occur and can cause an increased memory and CPU cache latency for moving critical data structures from real storage into the level 2 (L2) CPU cache. To reduce memory and cache latency, DB2 10

extensively uses the prefetch data hardware instruction available in z10 and z196 to prefetch critical data structures ahead of time from real storage to the L2 CPU cache.

Some of DB2's most frequently referenced internal structures are also rearranged for cache alignment to improve performance.

Figure 2-4 illustrates for a z10 the different CPU caches, their memory and cache latencies (expressed in machine cycles), and the data flow between CPU caches and real memory. Processors access instructions or data that reside in the L1 cache. Instructions or data are loaded as needed from L2 cache into the L1.5 cache and then into the L1 cache. There is a separate L1 cache for instructions and for data.

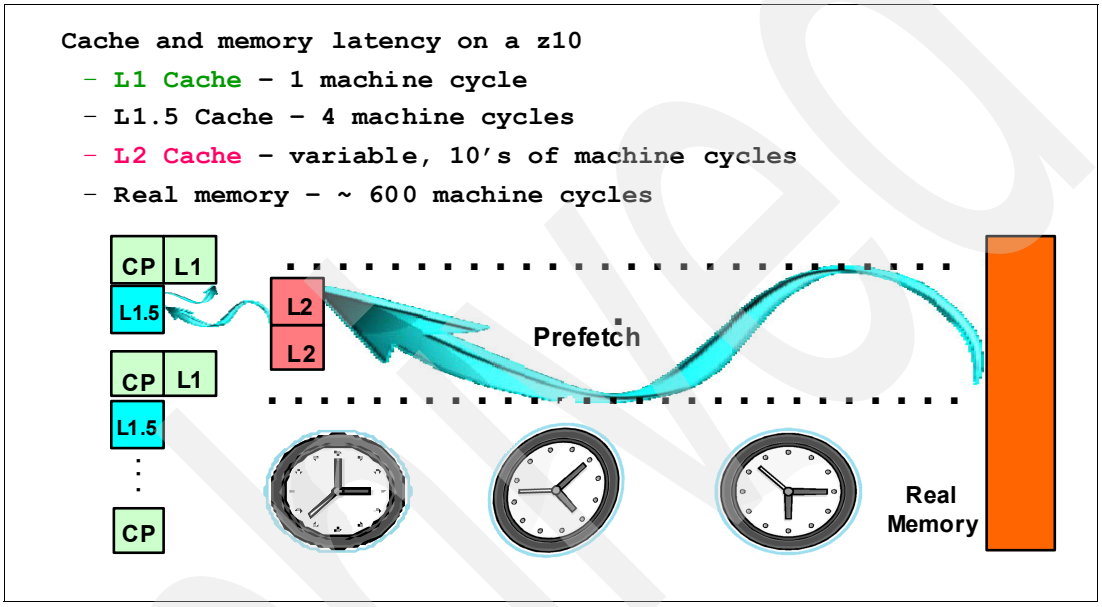


Figure 2-4 Memory and CPU cache latencies for z10

2.2.5 HiperDispatch

z/OS workload management and dispatching are enhanced to take advantage of the The IBM System z10 hardware design. The System z10 processor supports HiperDispatch, which provides a new mode of dispatching that increases the system capacity by up to 10%. The amount of improvement varies according to the system configuration and workload.

HiperDispatch is a combination of hardware features, z/OS dispatching, and the z/OS Workload Manager that increases system capacity by increasing the probability of cache hits when executing z/OS instructions. Each CPU has its own level 1 (L1) and level 1.5 (L1.5) cache. This level of cache is the best place to find data, because L1 and L1.5 cache requires the fewest machine cycles to access the data. CPUs are grouped at the hardware level in books.

All CPUs in the same book share a common level 2 (L2) cache, which is the next best place to access data. A CPU can also access the L2 cache of other books, but this access requires more machine cycles. The difference in machine cycles required to access a piece of data found in the L1 cache versus the same book L2 cache is relatively small. However, there is a significant difference in the number of machine cycles to access a piece of data in the same book L2 cache versus a separate book L2 cache. To optimize for same book L2 cache hits, a unit of work must run on a subset of the available processors in the same book.

You can take advantage of CPU cache optimization by activating HiperDispatch. When activated in z/OS, HiperDispatch provides the following capabilities:

- Consistently dispatches work on the same physical CPU to optimize for L1 and L1.5 CPU cache access.
- Dispatches work on a set of physical processors in the same book to optimize cache efficiency across the suspend/resume.

For more information about how to activate and use HiperDispatch, refer to *z/OS Version 1 Release 11 Implementation*, SG24-7729.

2.2.6 XML virtual storage constraint relief

In z/OS V1R11, XML code page dependent tables that are used for z/OS XML System Services processing are moved above the bar, which frees up common storage below the bar that is currently used.

2.2.7 XML fragment validation

In z/OS V1R12, XML System Services validating parsing performance is anticipated to improve. A revolutionary XML fragment validation capability enables you to validate only a portion of a document. For example, by revalidating only the single fragment being updated, DB2 10 for z/OS pureXML can avoid the costly revalidation of the entire document, which without this function can take many times longer.

2.2.8 Improved DB2 startup times and DSMAX with z/OS V1R12

DB2 startup times can be time consuming, especially when DB2 for z/OS has to go through lengthy data set allocation processing to open thousands of data sets during start. z/OS V1R12 delivers functions that allow for significant data set allocation elapsed time reductions. These improvements can result in significant DB2 start time reductions.

DB2 10 exploits z/OS 1.12 new allocation functions to improve the performance of allocation, deallocation, open, and close of the data sets in DB2 page sets. These functions improve the performance when opening a large number of data sets concurrently, especially for DB2 users with a high value of DSMAX. Also significant reduction in elapsed time has been observed by DB2 performance tests with a DSMAX of 100,000.

DB2 APARs PM00068, PM17542, and PM18557 enable the improvements for DB2 V8 and DB2 9.

2.2.9 CPU measurement facility

Enabling the gathering of hardware performance data on customer systems with Hardware Instruction Services (HIS) facilitates the improvement of z/OS-based software during:

- Problem determination
- CPU measurement facility (CPU MF, System z10 GA2)
- Supported by z/OS Hardware Instrumentation Facility (HIS)
- SMF type 113
- Always active in z/OS V1R12
- To be separately activated in z/OS V1R10 and z/OS V1R11
- Provides information about CPU cache

Archived

2.3 Synergy with IBM zEnterprise System

Today, many clients deploy their multitier workloads on heterogeneous infrastructures. For example, their mission-critical workloads and data serving need the availability, resiliency, security, and scalability strength of the mainframe. Meanwhile, other workloads, such as those handling intensive computations or low-cost, non-mission-critical transactions, might be better suited to run on UNIX or x86 architectures. Creating and managing these multiple and heterogeneous workloads, implemented on many physically discrete servers, can lead to inefficient and ineffective solutions.

The zEnterprise System provides an architecture that consists of heterogeneous virtualized processors that work together as one infrastructure. The system introduces a revolution in the end-to-end management of heterogeneous systems and offers expanded and evolved traditional System z capabilities.

The zEnterprise System consists of the following components:

- IBM zEnterprise 196 (z196)
- IBM zEnterprise BladeCenter® Extension (zBX)
- IBM zEnterprise Unified Resource Manager (Unified Resource Manager)

Figure 2-5 summarizes these components.

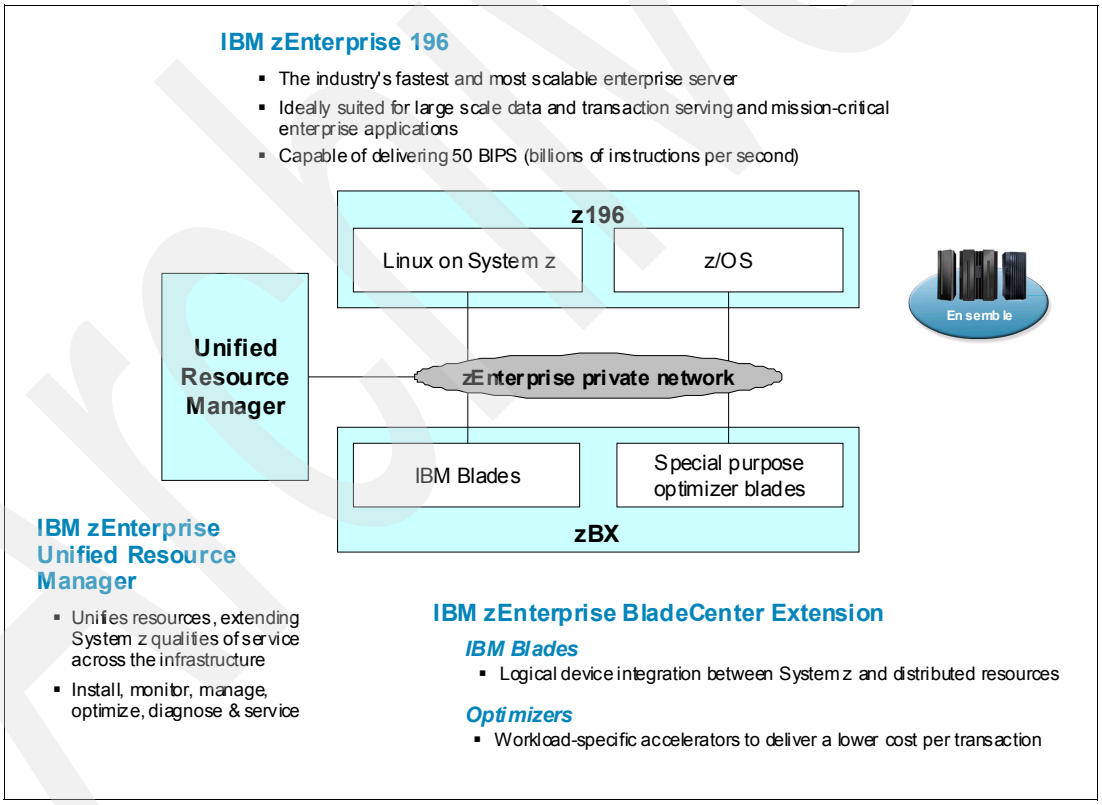


Figure 2-5 IBM zEnterprise System components

The existing capacities for zEnterprise are improved and additional capacities added using heterogeneous technology and Unified Resource Manager. See Figure 2-6.

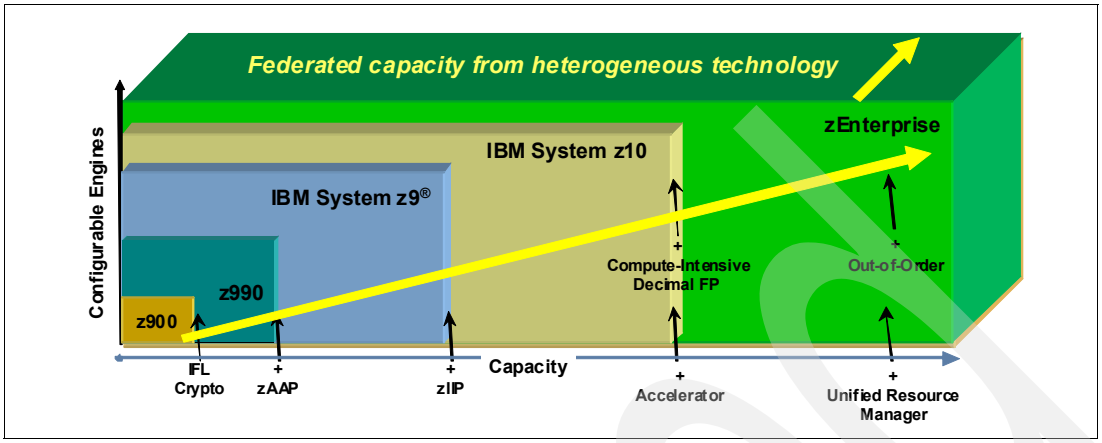


Figure 2-6 IBM zEnterprise System: Capacity and scalability

Here is a a closer look at each component and its capabilities:

- High-end mission-critical platform: z196

  The z196 is the industry's fastest and most scalable enterprise server, and it improves upon the capabilities of its predecessor, the System z10 Enterprise Class (System z10 EC). The z196 not only has the capabilities and scalability of physical resources (for example, processor, memory, I/O, and so on), but also offers better reliability, availability, and serviceability (RAS). It is the ideal platform for mission-critical enterprise workloads.

- Traditional workloads and data serving: z/OS

  z/OS offers extremely high scalability and performance for applications and data serving, and high availability and cross-system scalability enabled by Parallel Sysplex® and GDPS® solutions. z/OS provides a highly optimized environment for application integration and data management, with an additional performance benefit if both the application and the data are hosted on z/OS. It provides the ideal environment for both traditional application workloads and leading-edge technologies, as well as large scalable data serving, especially for mission-critical workloads.

- Mission-critical scale-out workload: z/VM® and Linux

  The z196 offers software virtualization through z/VM. The extreme virtualization capabilities provided by z/VM enable the high virtualization of thousands of distributed servers on Linux on System z. Linux on System z is an ideal platform for mission-critical scale-out workloads such as web applications, BI applications, and more.

- Special purpose optimizer blades

  zEnterprise provides an architecture that allows you to attach IBM special purpose optimizer blades. The first of this kind is IBM Smart Analytics Optimizer for DB2 for z/OS V1.1, which can accelerate certain data warehouse queries for DB2 for z/OS running on the z196, thereby reducing operational costs and improving the performance of BI processes.

- zEnterprise Unified Resource Manager

  The zEnterprise Unified Resource Manager is Licensed Internal Code (LIC), also known as *firmware*, that is part of the Hardware Management Console (HMC). Unified Resource Manager is a key component of zEnterprise. It provides integrated management across all elements of the system. Unified Resource Manager can improve your ability to integrate, monitor, and dynamically manage heterogeneous server resources as a single, logical

virtualized environment, while contributing to cost reduction, risk management, and service improvement.

- zEnterprise ensemble

  A zEnterprise *ensemble* is a collection of up to eight nodes, each composed of a z196 and optionally a zBX. The physical resources of servers are managed as a single virtualized pool by the Unified Resource Manager using the Hardware Management Console.

- Private networks

  Two new internal, secure networks are introduced for the zEnterprise ensemble. These networks are the *intraensemble data network* (IEDN) and the *intranode management network* (INMN). Existing external networks are supported as well. An IEDN is used for application data communications. An INMN is used for platform management within a zEnterprise. These networks have enough bandwidth for their purposes (10 Gbps for IEDN and 1 Gbps for INMN).

- Hypervisors

  The IBM POWER7™ blade provides PowerVM™ for IBM POWER7. PowerVM offers industry-leading virtualization capabilities for AIX®. This hypervisor is managed, along with the hypervisors of z196 (PR/SM™ and z/VM), by a single point of control using Unified Resource Manager.

From the DB2 subsystem point of view, we expect synergy with DB2 10. Faster CPUs, more CPUs, and more memory mean better DB2 performance, scalability.

- Combined with DB2 10 improvements in buffer pool management, virtual storage constraint relief and latch contention reduction, DB2 applications can observe significant cost reductions and scalability improvements on zEnterprise.
- Compression hardware improvements expected to improve DB2 data compression performance.
- 192 MB L4 Cache expected to benefit DB2 workloads.
- TLB changes expected to improve DB2 10 performance for 1 MB page sizes.
- Hybrid architecture to open opportunities for DB2 query performance acceleration.
- IBM Smart Analytics Optimizer for DB2 for z/OS V1.1 requires z196 and DB2 9 for z/OS.

Preliminary measurements currently available with DB2 9 show (see Figure 2-7):

- DB2 OLTP workloads observing 1.3 to 1.6 times DB2 CPU reduction compared to System z10 processors
- Higher DB2 CPU reduction can be achieved as the number of processors per LPAR increases
- With DB2 10 and zEnterprise, CPU reduction can be up to 1.8 times compared to DB2 9 and System z10 with large number of processors per LPAR

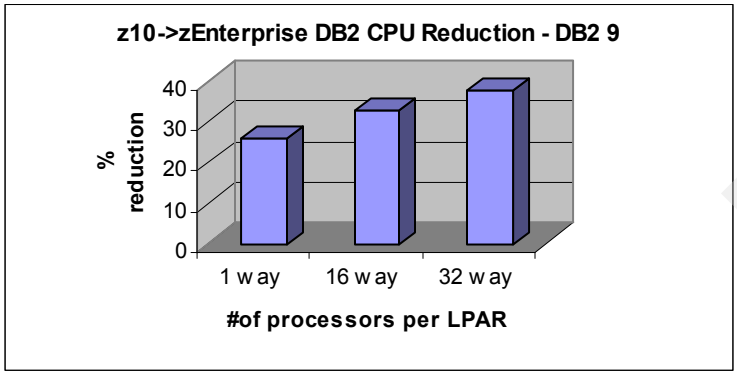


Figure 2-7 zEnterprise CPU reduction with DB2 9

2.4 Synergy with storage

Many DB2 for z/OS database systems have grown rapidly to accommodate terabytes of business critical information. Fast growing data volumes and increasing I/O rates can introduce new challenges that make it difficult to comply with existing service level agreements (SLA). For example, recovery times are expected to stay within SLA boundaries or must not cause outages of business critical applications.

While I/O rates are increasing, existing applications have to perform according to SLA expectations. To support existing SLA requirements in an environment of rapidly increasing data volumes and I/O rates, DB2 for z/OS uses features in the Data Facility Storage Management Subsystem (DFSMS) that help to benefit from performance improvements in DFSMS software and hardware interfaces.

2.4.1 Extended address volumes

The track addressing architecture of traditional volumes allows for relatively small gigabyte (GB) capacity volumes, which put pressure on the 4-digit device number limit. The largest traditional 3390 model volume (3390-54) has a capacity of 65,520 cylinders or approximately 54 GB. In a Parallel Sysplex you can define up to 65280 I/O devices such as volumes, tapes, cartridges, printers, terminals, network interfaces, and other devices. Due to rapidly growing data, more of these relatively small GB capacity volumes are required, making it more likely to hit the 4-digit device number limit.

To address this issue, z/OS V1R10 introduces extended address volumes (EAV) to provide larger volume capacity in z/OS to enable you to consolidate a large number of small GB capacity volumes onto a significantly smaller number of large EAV (see Figure 2-8).

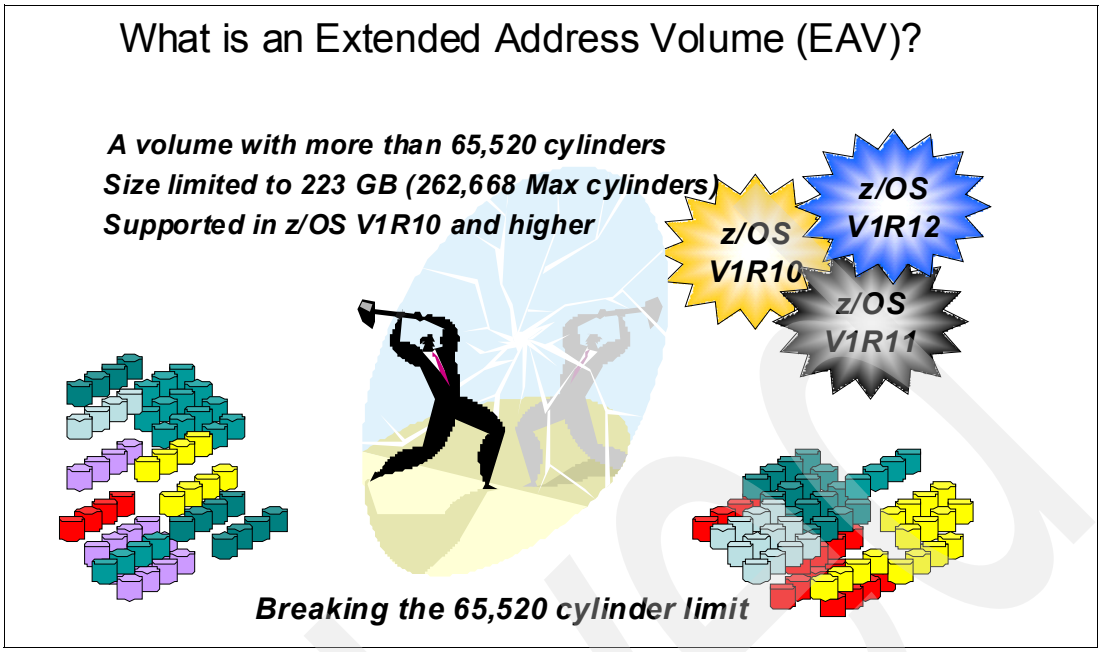


Figure 2-8 EAV breaking the limit

EAV implement an architecture that provides the architectural capacity of 100s of terabytes of data for a single volume. For example, the maximum architectural capacity of an EAV is 268,434,453 cylinders. However, the current EAV implementation is limited to a capacity for a single volume of 223 GB or 262,668 cylinders (see the virtualization shown in Figure 2-9 on page 14).

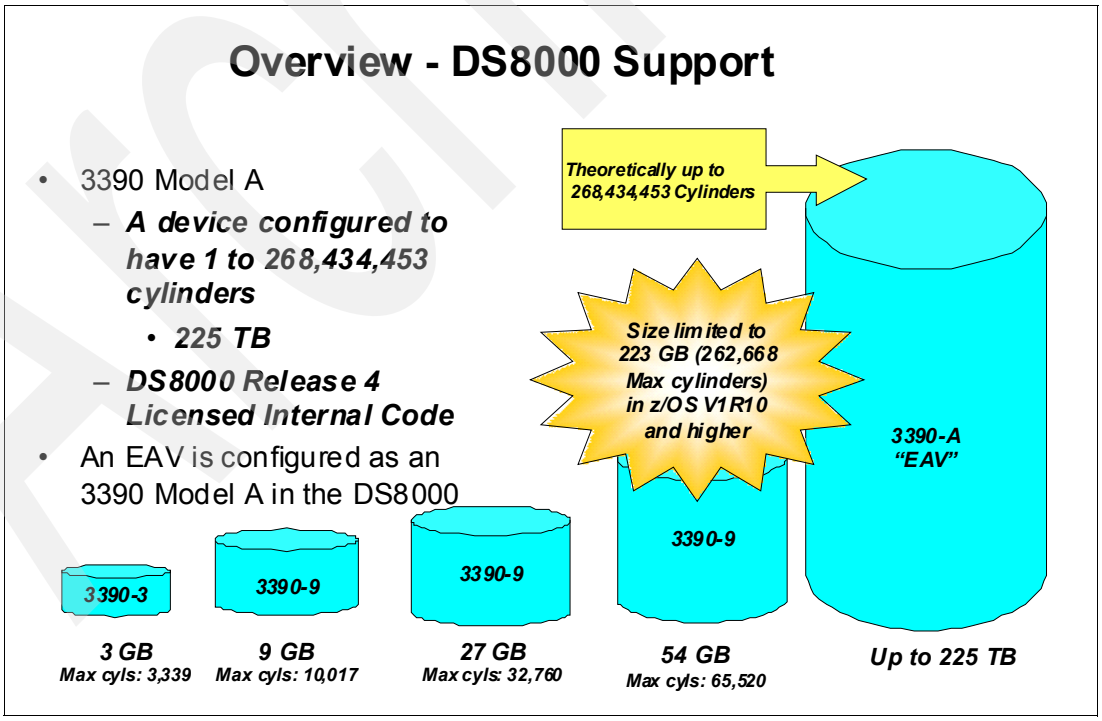


Figure 2-9 Overview DS8000 EAV support

When you consolidate big numbers of data sets that reside on multiple volumes onto bigger EAV, you also increase the I/O density of these EAV. You concurrently access the data sets that previously resided on multiple small GB capacity volumes through just one single large EA volume, which can increase the disk I/O response times due to disk queuing. To solve this issue, you need to configure the EAV to use parallel access volumes (PAVs). The PAV capability represents a significant performance improvement by the storage unit over traditional I/O processing. With PAVs, your system can access a single volume from a single host with multiple concurrent requests.

When you PAV-enable a logical volume, you statically assign a fixed number of PAVs to that volume. If one single volume becomes over used by the workload, the number of PAVs that are assigned to that volume might still not be sufficient to handle the I/O density of the workload that is flooding the volume. To address this situation, the DS8000 series of disk subsystems offers enhancements to PAV with support for HyperPAV, which enables applications to achieve equal or better performance than PAVs alone, while also using the same or fewer operating system resources.

For more information about PAV and HyperPAV technology, refer to *DB2 9 for z/OS and Storage Management*, SG24-7823.

For additional information about EAV, refer to 3.7, “Support for Extended Address Volumes” on page 62.

2.4.2 More data set types supported on EAV

Extended addressing space (EAS) eligible data sets are defined to be those that can be allocated in the extended addressing space of an EA volume. They can reside in track or cylinder-managed space and can be SMS-managed or non-SMS managed, sometimes referred to as *cylinder-managed space*.

EAS-eligible data set types include VSAM, sequential, direct and partitioned data sets. As illustrated in Figure 2-10, VSAM files became EAS eligible in z/OS V1R10, extended format sequential data sets were added in z/OS V1R11, basic and large format sequential, BDAM, PDS and PDSE, VSAM volume data sets (VVDS) and basic catalog structure (BCS) data sets have become EAS eligible in z/OS V1R12.

With the new support provided in z/OS V1R12, you can define and use integrated catalog facility (ICF) basic catalog structure (BCS) with EA, allowing catalogs to grow larger than 4 GB. The maximum size the ICF catalog is limited by the size of the ICF catalog volume. For example, if the ICF catalog resides on a 223 GB volume, the catalog cannot grow beyond that volume size.

Using this feature, you can store a DB2-related ICF catalog in an EAS of an EA volume, which simplifies ICF catalog administration and additionally contributes to high availability because the ICF catalog can grow bigger and is unlikely to run out of space.

Archived

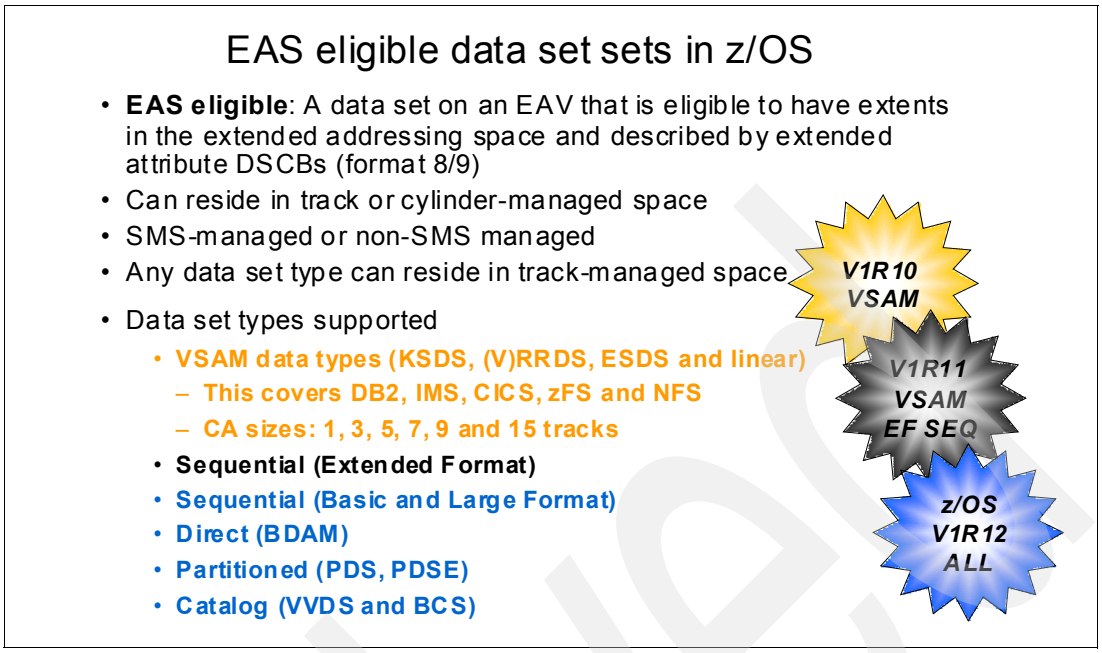


Figure 2-10 EAV support for ICF catalogs and VVDS

2.4.3 Dynamic volume expansion feature

The IBM System Storage DS8000 series supports dynamic volume expansion, which increases the capacity of existing system volumes, while the volume remains connected to a host system. This capability simplifies data growth by allowing volume expansion without taking volumes offline. Using dynamic volume expansion significantly reduces the complexity of migrating to larger volumes (beyond 65,520 cylinders).

Dynamic volume expansion is performed by the DS8000 Storage Manager and can be requested using its web interface. You can increase the 3390 volumes in size, for example from a 3390 model 3 to a model 9 or a model 9 to a model A (EA volume). z/OS V1R11 introduces an interface that can be used to make requests for dynamic volume expansion of a 3390 volume on a DS8000 from the system.

For more information about the Dynamic Volume Expansion feature, refer to *DB2 9 for z/OS and Storage Management*, SG24-7823.

2.4.4 SMS data set separation by volume

When allocating new data sets or extending existing data sets to new volumes, SMS volume selection frequently calls SRM to select the best volumes. Unfortunately, SRM might select the same set of volumes that currently have the lowest I/O delay. Poor performance or single point of failure (SPOF) can occur when a set of functional-related critical data sets are allocated onto same volumes. In z/OS V1R11, DFSMS provides a function to separate critical data sets, such as DB2 partitions, active log, and boot strap data sets, onto different volumes to prevent DASD hot spots, to reduce I/O contentions, and to increase data availability by putting critical data sets behind separate controllers.

This function provides a solution by expanding the scope of the data set separation function currently available at PCU level to the volume level. To use this function, you need to define the volume separation groups in the data set separation profile. During data set allocation,

SMS attempts to separate data sets that are specified in the same separation group onto different extent pools and volumes.

This function provides a facility for the installation to separate functional-related critical data sets onto different extent pools and volumes for better performance and to avoid SPOFs.

For more information about SMS data set separation by volume, refer to *DB2 9 for z/OS and Storage Management*, SG24-7823.

2.4.5 High Performance FICON

Higher CPU capacity requires greater I/O bandwidth and efficiency. High Performance FICON (zHPF) enhances the z/Architecture and the FICON interface architecture to provide I/O optimizations for OLTP workloads. zHPF is a data transfer protocol that is optionally employed for accessing data from an IBM DS8000 storage subsystem. Data accessed by the following methods can benefit from the improved transfer technique:

- DB2
- IMS™ Fast Path
- Partitioned data set extended (PDSE)
- Virtual Storage Access Method (VSAM)
- zSeries® File System (zFS)
- Hierarchical file system (HFS)
- Common VTOC access facility (CVAF)
- ICF Catalog
- Extended format sequential access method (SAM).

Performance tests with zHPF have shown performance improvements as illustrated in Figure 2-11.

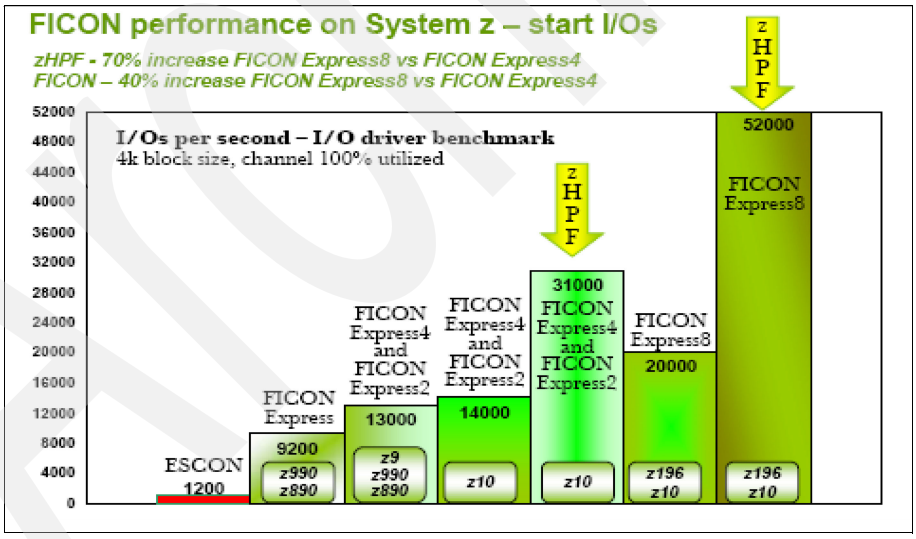


Figure 2-11 zHPF performance

IBM introduced High Performance FICON, known as zHPF, with the z10 processor. zHPF introduces a more efficient I/O protocol that significantly reduces the back and forth communication between devices and channels. zHPF has been further enhanced by the new channel technology provided with the IBM z196. The FICON Express8 channels on a z196 support up to 52,000 zHPF 4 KB IOPS or up to 20,000 FICON 4 KB IOPS. When comparing to the FICON Express4 channels, this is an improvement of about 70% for zHPF protocols

and 40% for FICON protocols. The maximum zHPF 4 KB IOPS measured on a FICON Express8 channel is 2.6 times the maximum FICON protocol capability.

Not all media manager I/Os can use zHPF. I/Os that access discontiguous pages are ineligible and format-writes are ineligible. On the z10, I/Os that read or write more than 64 KB are ineligible, but this restriction is removed on the zEnterprise system.

The protocol simplification introduced by zHPF is illustrated in Figure 2-12. This example includes a 4 KB read FICON channel program where three information units (IUs) are sent from the channel to the control unit and plus IUs from the control unit to the channel. zHPF cuts in half the total number of IUs sent using only one IU from the channel to the control unit and two IUs from the control unit to the channel.

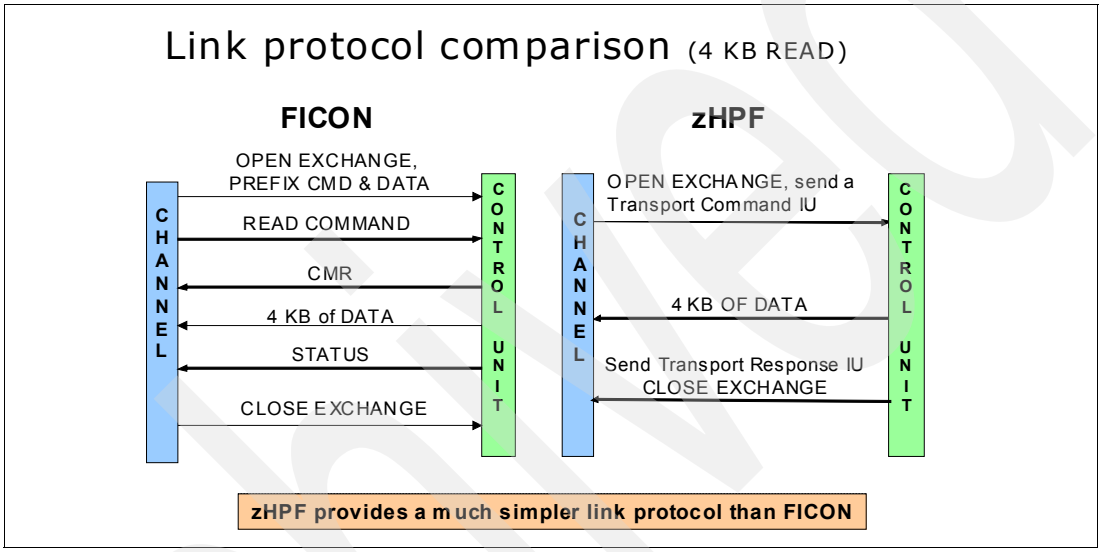


Figure 2-12 zHPF link protocol comparison

For more information about zHPF, refer to *IBM zEnterprise 196 I/O Performance Version 1,* which is available from:

ftp://public.dhe.ibm.com/common/ssi/ecm/en/zsw03169usen/ZSW03169USEN.PDF

DB2 for z/OS prefetch I/O

A set of measurements focused on prefetch operations performed by DB2 for z/OS. DB2 supports three types of prefetch, namely sequential prefetch, dynamic prefetch, and list prefetch. In most cases, these prefetch types were not eligible for zHPF prior to the z196, because they read more than 64 KB bytes, but the z196 makes sequential prefetch and dynamic prefetch both zHPF-eligible.

Starting with DB2 9, sequential prefetch can read up to 256 KB for SQL and 512 KB for utility per I/O, depending on how many buffers are available for prefetch. In contrast, dynamic prefetch continues to read 128 KB per I/O, but unlike sequential prefetch, it can do two parallel prefetch I/Os. Thus, dynamic prefetch achieves better I/O performance than sequential prefetch.

The prefetch measurements using 4 KB page sizes show that zHPF increases the throughput in case of reading from disk as well as reading from cache. The best case is DB2's dynamic prefetch when the disks are not involved. Remember that dynamic prefetch can generate two concurrent streams on the link, even though DB2 is reading only one data set.

DB2 logging and deferred writes

If the number of 4 KB log buffers is less than or equal to 16, these writes are eligible for zHPF on a z10. If the number of buffers is greater than 16, the z10 cannot use zHPF, but the z196 can. Measurements show that DB2 10 can achieve a slightly higher log throughput with zHPF compared to FICON.

DB2 deferred writes write up to 128 KB for SQL (exceptions are 256 KB writes for LOBs and 512 KB for format-write in utilities), but each write I/O is limited to a range of 180 pages. Thus, whether or not the update write I/Os that are generated by deferred writes are zHPF-eligible depends on the density of the modified pages. The z196 enhancement enables larger size update writes (over 64 KB) to use zHPF. This enhancement applies to sequential Inserts, which is associated with table spaces growing. However, if a table space is growing, it has to be preformatted. The preformat I/Os are usually more of a bottleneck than the sequential deferred write I/Os and they are not eligible for zHPF.

As stated previously, zHPF can help increase the DB2 log throughput by 3%. Although 3% might not be viewed as a large number, the cumulative effect of making more DB2 I/Os eligible for zHPF is to lower the overall utilization of the channels and host adapters and these lower utilizations enable more dramatic performance effects.

DB2 utilities

DB2 utilities use a variety of different types of I/Os. The z196 helps with sequential reads. The sequential reads, from a table space or index, are now zHPF-eligible. Format writes and list prefetch I/Os are not eligible. The sequential reads from DSORG=PS data sets are zHPF-eligible if the data set has extended format.

Table 2-1 summarizes all of the different types of I/Os that DB2 uses. It shows which I/Os are eligible for zHPF on the z10 and on the z196.

Table 2-1 Eligibility for zHPF of DB2 I/O types

Type of DB2 I/O	z10	z196
Random single 4 KB read/write	YES	YES
Sequential prefetch and dynamic prefetch	NO	YES
DB2 work files, reads and update writes	YES	YES
List prefetch	NO	NO
Log writes <=64 K	YES	YES
Log writes > 64 K	NO	YES
Log reads (with DB2 9 or 10)	NO	YES
Sequential update writes	NO	YES
Update writes with a discontinuity	NO	NO
Random update writes without a discontinuity	YES	YES
Format and preformat	NO	NO
Utility table space scans (sequential prefetch)	NO	YES
Sequential reads from DSORG=PS, EF data sets	NO	YES
Sequential reads from DSORG=PS, non-EF data sets	NO	NO
Sequential writes to DSORG=PS	NO	NO

2.4.6 IBM System Storage DS8800

The DS8800 high-performance flagship model features IBM POWER6+™ processor technology to help support high performance. Compared to the performance of the previous DS8000 system (DS8300) the new processor, along with 8 Gbps Host Adapters and 8 Gbps Device adapters, helps the DS8800 to improve sequential read and write performance. The DS8800 is the first IBM enterprise class control unit to use 2.5” serial-attached SCSI (SAS) drives in place of 3.5” Fibre Channel drives, which allows the footprint in terms of space and energy consumption to be reduced. A single frame of the DS8300 holds up to 144 drives. The DS8800 holds up to 240 drives.

The performance can improve in the following ways:

- Whereas the speeds of the host adapters and disk adapters in the DS8300 are 4 Gbps, the speeds of the host adapters and device adapters of the DS8800 are 8 Gbps. Together with FICON Express 8, the DS8800 enables 8 Gbps speeds to be achieved from the host across the channel links and all the way to the disks.
- A single frame of the DS8800 has twice as many device adapters as the DS8300 (which is important to support the extra drives) and can, therefore, achieve higher throughput for parallel sequential I/O streams.

Measurements were done of a DB2 table scan using 4 KB pages on a DS8300 and a DS8800 attached to a z196 processor, which enables DB2 prefetch I/Os to become zHPF eligible. The buffer pool is defined large enough to enable DB2 to read 256 KB per I/O. See Figure 2-13.

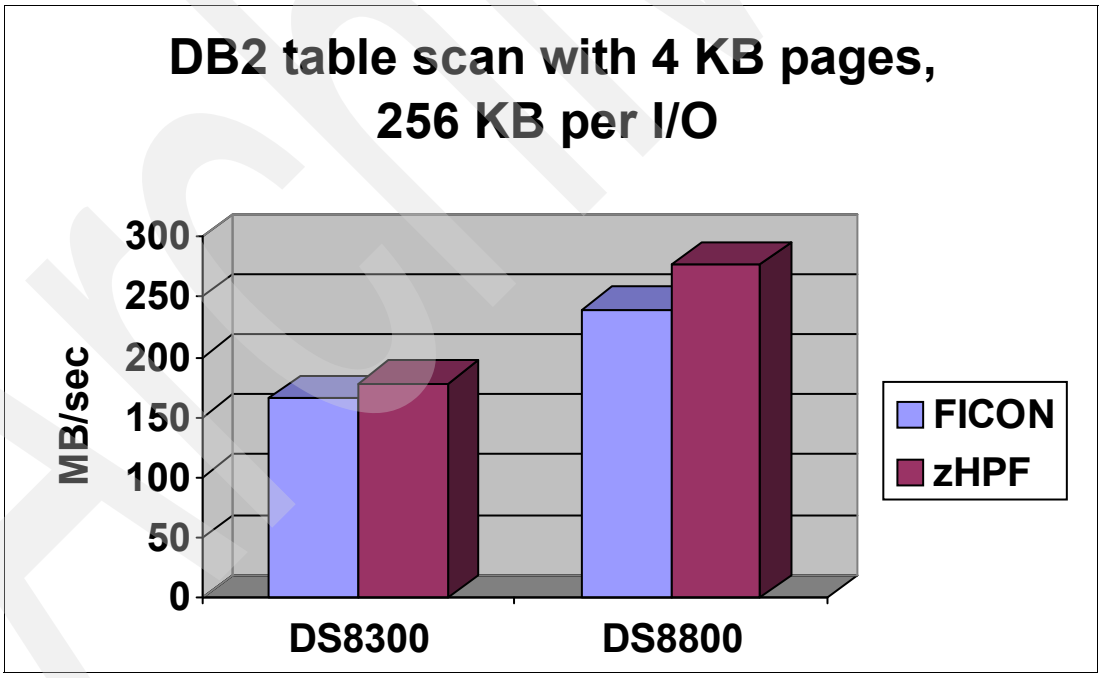


Figure 2-13 DS8000 versus DS8300 for DB2 sequential scan

The results show that using FICON, the DS8800 throughput is 44% higher than the DS8300. Using zHPF, the DS8800 throughput is 55% higher than the DS8300. And, if we compare the DS8800 with zHPF to the DS8300 with FICON, the throughput increases 66%.

Table scans are not the only type of DB2 workload that can benefit from faster sequential I/O. Dynamic prefetch, sequential inserts, and DB2 utilities, all become much faster. Synchronous

DB2 I/O might also benefit from 8 Gbps host adapters, but not to the degree that sequential I/O does.

Another recent set of measurement was performed on DB2 logging. As the DB2 log buffer queues build up, the size of the DB2 log I/Os becomes large and it requires fast channels. The FICON Express 8, in combination with the fast host adapters in the DS8800, helps boost the throughput.

Figure 2-14 shows that the DS8800 improves the maximum throughput by about 70% when the log buffer queues become large. These measurements were done on a z196 which enables the log I/Os exceeding 64 KB to be zHPF eligible, providing an additional boost in performance. Figure 2-14 also shows that DB2 10 achieves higher throughput than DB2 9 for the reason described in 13.14, “Logging enhancements” on page 574.

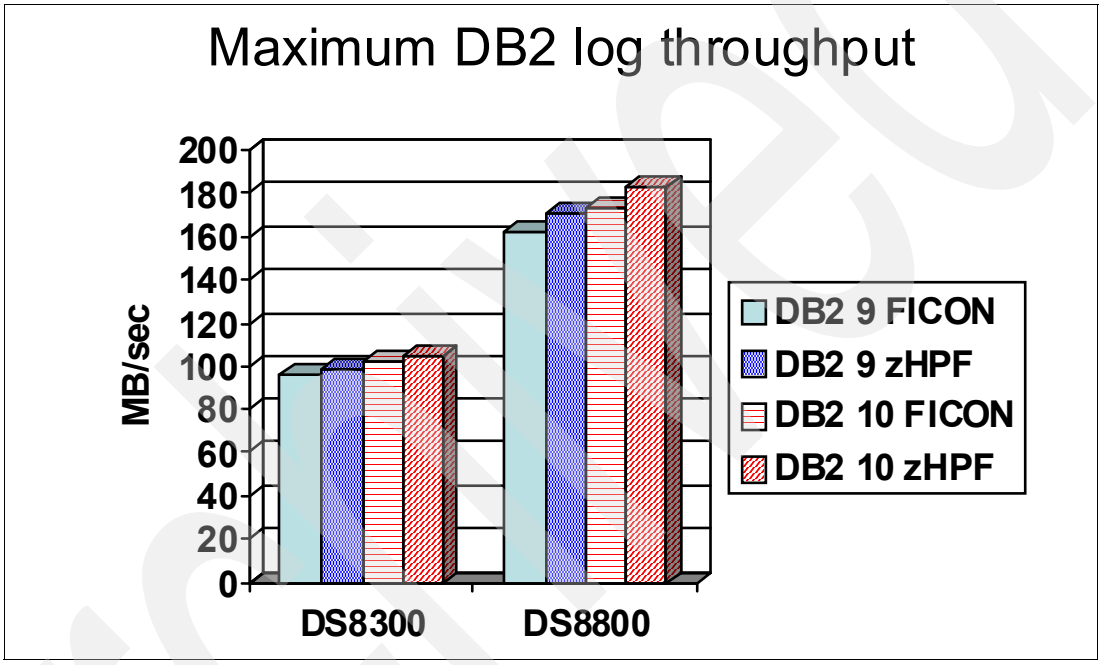


Figure 2-14 Log writes throughput comparisons

For details about the storage systems, see:

http://www.ibm.com/systems/storage/news/center/disk/enterprise/

2.4.7 DFSMS support for solid-state drives

Solid®-state drives are high I/O operations per second (IOPS) enterprise-class storage devices that are targeted at business-critical production applications, which can benefit from a high level of fast-access storage to speed up random I/O operations. In addition to better IOPS performance, solid-state drives offer a number of potential benefits over electromechanical hard disk drives (HDDs), including better reliability, lower power consumption, less heat generation, and lower acoustical noise.

In z/OS V1R11, DFSMS allows SMS policies to help direct new data set allocations to solid-state drive volumes. Tooling also helps to identify existing data that might perform better when placed on solid-state drive volumes. A new device performance capability table is available for volumes that are backed by solid-state drives. Selection preference toward solid-state drive or non-solid-state drive volumes can be achieved easily by setting the Direct Millisecond Response Time and Direct Bias values for the storage class.

For more information about solid-state drives, refer to the following resources:

- z/OS HOT TOPICS Newsletter, issue 20th March 2009, *Stop spinning your storage wheels*, which is available from:

  http://publibz.boulder.ibm.com/epubs/pdf/e0z2n191.pdf

- *DB2 9 for z/OS and Storage Management*, SG24-7823
- *Ready to Access DB2 for z/OS Data on Solid-State Drives*, REDP-4537
- Washington Systems Center Flash WP101466, *IBM System Storage DS8000 with SSDs An In-Depth Look at SSD Performance in the DS8000*

2.4.8 Easy Tier technology

IBM DS8700 disk storage system now includes a technology invented by IBM Research that can make managing data in tiers easier and more economical. The IBM System Storage Easy Tier feature uses ongoing performance monitoring to move only the most active data to faster solid-state drives (SSD), which can eliminate the need for manual storage tier policies and help reduce costs.

By automatically placing the clients' most critical data on SSDs, Easy Tier provides quick access to data so it can be analyzed for insight as needed, to provide competitive advantage. The Easy Tier technology in that respect potentially offers an alternative to the DFSMS policy based SSD volume data set placement that we described in 2.4.7, "DFSMS support for solid-state drives" on page 21.

For more information about the Easy Tier technology, refer to *DB2 9 for z/OS and Storage Management*, SG24-7823.

2.4.9 Data set recovery of moved and deleted data sets

With versions of DFSMShsm prior to V1R11, Fast Replication recover (FRRECOVER) of individual data sets was not applicable to data sets that were deleted or moved to a different volume after the volume FlashCopy backup was taken.

Starting with DFSMShsm V1R11, you can configure the SMS copypool to use the CAPTURE CATALOG option. With this option enabled, DFSMShsm collects additional information during FlashCopy volume backup that enables the FRRECOVER process to restore a data set on its original source volume, even in situations in which the data set to restore was deleted or moved to a different volume.

Considerations when using DB2 9

In DB2 9, you can generally recover an individual table space from a DB2 system level backup (SLB). In a z/OS V1R11 environment, this capability also supports the recovery of deleted or moved DB2 data sets. However, DB2 9 does not allow you to recover a table space from an SLB in the following situations:

- The table space was moved by DB2 9 (for example, by a DB2 utility) after the SLB was taken.
- A RECOVER, REORG, REBUILD INDEX utility or a utility with the NOT LOGGED option was run against the table space after the SLB was taken.

Enhancements with DB2 10

In DB2 10, these DB2 9 RECOVER considerations are removed. For example, if a REORG utility runs after an SLB is created, DB2 restores the data set to its original name and

subsequently renames it to its current name to deal with the I and J instances of the data set name. DB2 10 can handle that situation even if multiple REORG utilities have been executed since the SLB was created.

2.4.10 Synergy with FlashCopy

The use of FlashCopy for data copy operations can reduce elapsed and CPU times spent in z/OS. A z/OS FlashCopy client, such as the DFSMSdss ADRDSSU fast replication function, waits only until the FlashCopy operation has logically completed, which normally takes a much shorter time than the actual physical FlashCopy operation spends inside the DS8000 disk subsystem. No CPU is charged to z/OS clients for the DS8000 physical data copy operation.

DB2 for z/OS and FlashCopy use

FlashCopy use in DB2 for z/OS began with Version 8 with the use of disk volume FlashCopy for the DB2 SYSTEM BACKUP and RESTORE utility, which helped to reduce the z/OS elapsed time that is required to backup and restore the entire DB2 system. For example, backing up a few terabytes of data consisting of active logs, Bootstrap data sets, catalog, directory, and user table and index spaces can become a matter of just a few seconds to logically complete in z/OS without application unavailability.

In DB2 9, the BACKUP and RESTORE SYSTEM utilities were enhanced to support functions that are available with DFSMShsm V1R8. For example, in DB2 9, you can keep several SYSTEM BACKUP versions on DASD or on tape, you can use the SYSTEM BACKUP utility to perform incremental track level FlashCopy operations, and you can use a SYSTEM BACKUP utility created backup to recover individual table spaces or index spaces. Also in DB2 9, the CHECK INDEX SHRLEVEL CHANGE utility performs consistency checks on a table and index space shadow that the utility creates using the DFSMSdss ADRDSSU fast replication function which implicitly uses FlashCopy.

For more information about FlashCopy usage in DB2 for z/OS, refer to *DB2 9 for z/OS and Storage Management*, SG24-7823.

FlashCopy use by DB2 COPY utility

In DB2 10, the COPY utility is enhanced to provide an option to use the DFSMSdss fast replication function for taking full image copies by the COPY utility or the inline COPY function of the REORG and LOAD utilities. The DFSMSdss fast replication function invokes FlashCopy to perform the physical data copy operation, which in turn offloads the physical data copy operation to the DS8000 disk subsystem. As a result, no data pages need to be read into the table space buffer pool, which by itself reduces CPU usage that is normally caused by buffer pool getpage processing. For more information about FlashCopy use by the DB2 10 COPY utility, refer to 11.1, “Support FlashCopy enhancements” on page 426.

To illustrate the performance benefit that the new FlashCopy use can provide, we created and populated a sample table with 100000 rows in a table space. We defined the table space with MAXROWS 1 to force DB2 to allocate 100000 data pages with one row per page. We then performed two COPY utility executions, one with and one without using FlashCopy, to compare COPY utility performance.

Figure 2-15 shows the accounting report highlights of the utility execution using the FLASHCOPY NO COPY utility option. In the buffer pool activity section, DB2 reads all data pages into the local buffer pool for image copy processing.

```
HIGHLIGHTS
--------------------
PARALLELISM: UTILITY

TIMES/EVENTS  APPL(CL.1)  DB2 (CL.2) TOTAL  BPOOL ACTIVITY   TOTAL
------------  ----------  ---------- ---------------------  ------
ELAPSED TIME   11.381112    0.047218 GETPAGES              100299
CP CPU TIME     0.981997    0.472171 BUFFER UPDATES            58
 AGENT          0.022562    0.006620 SEQ. PREFETCH REQS      1564
 PAR.TASKS      0.959435    0.465551 PAGES READ ASYNCHR.    100093
```

Figure 2-15 FLASHCOPY NO COPY utility accounting report

Figure 2-16 shows the accounting report highlights of the utility execution using the FLASHCOPY YES COPY utility option. In the buffer pool activity section, DB2 does not read the data pages that are to be processed into the local buffer pool. Instead, DB2 invokes the DFSMSdss ADRDSSU fast replication function, which in this particular situation results in a 97% z/OS CPU time and a 94% elapsed time reduction compared to the COPY utility execution, as illustrated in Figure 2-15.

```
HIGHLIGHTS
--------------------
PARALLELISM: NO

TIMES/EVENTS  APPL(CL.1)  DB2 (CL.2) TOTAL  BPOOL ACTIVITY   TOTAL
------------  ----------  ---------- ---------------------  ------
ELAPSED TIME    0.731253    0.034548 GETPAGES                 103
CP CPU TIME     0.020734    0.004625 BUFFER UPDATES            25
 AGENT          0.020734    0.004625 SEQ. PREFETCH REQS         0
 PAR.TASKS      0.000000    0.000000 PAGES READ ASYNCHR.        0
```

Figure 2-16 FLASHCOPY YES COPY utility accounting report

CPU and elapsed time savings: The CPU and elapsed time savings that you can achieve using the COPY utility FlashCopy exploitation can vary, depending on I/O configuration and table space size.

2.4.11 DB2 catalog and directory now SMS managed

In DB2 10 the catalog and directory table and index spaces are defined as extended attribute (EA) data sets and can reside on EAV.

Note that if you plan to use the DB2 Backup System utility for backing up and recovering your DB2 subsystem data, this prerequisite must also be met. Thus, it makes sense to have your overall DB2 backup strategy in mind when you plan the installation of or the migration to DB2 10 for z/OS and if your catalog and directory VSAM data sets are not SMS managed. Careful planning makes it possible to get your catalog and directory SMS configuration correct for DB2 10 and for your backup and recovery strategy.

With the DB2 catalog and directory data set SMS managed, DB2 passively exploits SMS features that are only available to SMS managed data sets. Some of the key features are:

- Data set attributes, performance characteristics and management rules of data sets are transparently defined in SMS using data classes (DATACLAS), storage classes (STORCLAS) and management classes (MGMTCLAS).
- The assignment of DATACLAS, STORCLAS, MGMTCLAS can be externally provided (for example in the IDCAMS DEFINE CLUSTER command) and enforced through SMS policies, also known as automatic class selection (ACS) routines.
- DASD volumes are grouped into SMS storage groups (STORGRP). During data set creation, the STORGRP assignment is transparently enforced by the SMS policy through the STORGRP ACS routine. Within a selected STORGRP, SMS places the data set that is to be created onto one of the volumes within that SMS STORGRP. To prevent a SMS STORGRP from becoming full, you can provide overflow STORGRPs, and you can define utilization thresholds that are used by SMS to send alert messages to the console in case the STORGRP utilization threshold is exceeded. Monitoring overflow STORGRPs and automating SMS STORGRP utilization messages provide strong interfaces for ensuring DASD capacity availability. You can also use the information provided by IDCAMS DCOLLECT for regular STORGRP monitoring and capacity planning.
- Data set attributes are transparently assigned through SMS data classes (DATACLAS) during data set creation. New SMS data set attributes that are required to support better I/O performance and data set availability can transparently be activated by changing online the DATACLAS in SMS. For example, you can change online the DATACLAS volume count which becomes active immediately for all data sets using that DATACLAS, There is no need for you to run an IDCAMS ALTER command to add volumes to the data set as you would have to if the data sets were non-SMS managed.
- The use of SMS storage classes (STORCLAS) allow you to assign performance attributes on data set level, because a SMS STORCLAS is assigned during data set creation. With non-SMS managed data sets storage related performance attributes are normally only available on volume level affecting all data sets residing on that volume. For example, you can support a minimum disk response time for a particular data set by assigning a particular SMS STORCLAS that supports that performance requirement.
- Some data set attributes (for example, the volume count) can simply be adjusted by changing the SMS data class. For example, instead of using the ALTER command to add volumes to an existing DB2 VSAM LDS data set, you can simply increase the volume count in the data class definition that is used by the DB2 VSAM LDS data set

We discuss SMS managed DB2 catalog and directory data sets in 12.5.1, “SMS-managed DB2 catalog and directory data sets” on page 490.

2.5 z/OS Security Server

DB2 10 introduces security enhancements to support z/OS Security Server (RACF®) features that were introduced in z/OS V1R10 and z/OS V1R11. These features, now supported in DB2 10, are RACF password phrases and RACF identity propagation.

2.5.1 RACF password phrases

In z/OS V1R10, you can set up a RACF user to use a password *phrase* instead of a textual password. A textual passwords can be up to eight characters in length and in most cases must not contain special characters other than numbers and letters. For example, you cannot

use a blank in a textual password. Textual passwords are considered weak passwords, because they are easier to guess or to crack. With a password phrase, you can use a memorable phrase to authenticate with the database. You can use uppercase or lowercase letters, special characters, or even spaces. Due to its complexity (up to 100 characters in z/OS), it is impossible to guess, unlikely to crack, and easier to remember.

For more information about how DB2 10 supports the use of password phrases, refer to 10.6.1, "z/OS Security Server password phrase" on page 411.

2.5.2 RACF identity propagation

In today's business world, a user authenticates to the enterprise network infrastructure using an enterprise identity user ID (for example, the user ID that you use when you log in to an enterprise network from your workstation). After successful authentication, users want to be able to use the same enterprise identity user ID to authenticate to application systems and database servers that belong to the enterprise infrastructure, which in our case includes a DB2 for z/OS database server.

Due to z/OS Security Server (RACF) specific restrictions or limitations, an enterprise identity user ID often cannot be used to authenticate to a z/OS server. For example, a RACF user ID can be up to eight characters in length and might not support all the characters that you can use in an enterprise identity user ID. Additionally, for the enterprise identity user ID to work, you have to define one RACF user for each enterprise identity user, which causes additional administration effort.

This case is where RACF Enterprise Identity Mapping (EIM) is most useful. With EIM support activated in RACF, you can authenticate to DB2 for z/OS using your enterprise identity user ID. DB2 for z/OS then uses a RACF service to map that enterprise identity user ID to an existing RACF user ID that is used by DB2 for further authentication and security validation.

In z/OS Security Server V1R11, the EIM function is described as z/OS *identity propagation*. z/OS identity propagation is fully integrated into the RACF database and has no external dependency to an LDAP server infrastructure.

DB2 10 for z/OS supports z/OS identity propagation for Distributed Relational Database Architecture™ (DRDA®) clients that submit SQL to DB2 through trusted context database connections. You can audit SQL activities performed by enterprise identity users by using the DB2 10 policy based auditing capability.

For more information about how DB2 10 supports z/OS identity propagation and how to audit DB2 activities performed by enterprise identity users, refer to 10.6.2, "z/OS Security Server identity propagation" on page 416.

2.6 Synergy with TCP/IP

The DB2 for z/OS server uses TCP/IP functions in many places. As a consequence, performance improvements in TCP/IP can help to improve DB2 for z/OS performance in cases where the TCP/IP improvement provided affects a TCP/IP interface or infrastructure that is also used by DB2.

Synergy with TCP/IP affects DB2 for z/OS as a server. In addition, DB2 client processes can benefit from improvements in TCP/IP. For example, you can take advantage of the FTP named pipe support introduced by z/OS V1R11 to load data into a DB2 table while the pipe is

still open for write and while an FTP client simultaneously delivers data through the UNIX System Services named pipe.

2.6.1 z/OS V1R10 and IPv6 support

Industry sources highlight the issue of IPv4 running out of addresses. For more information, see:

http://www.potaroo.net/tools/ipv4/index.html

To address this issue, support for IPv6 IP-addresses is added in DB2 9 for z/OS. DB2 10 for z/OS runs on z/OS V1R10 or later releases of z/OS. z/OS V1R10 is IPv6 certified and, therefore, is officially enabled to support DB2 10 for z/OS workloads that use IPv6 format IP addresses.

For details about the z/OS V1R10 IPv6 certification, refer to “Special Interoperability Test Certification of the IBM z/OS Version 1.10 Operating System for IBM Mainframe Computer Systems for Internet Protocol Version 6 Capability”, US government, Defense Information Systems Agency, Joint Interoperability Test Command, which is available from:

http://jitc.fhu.disa.mil/adv_ip/register/certs/ibmzosv110_dec08.pdf

2.6.2 z/OS UNIX System Services named pipe support in FTP

Named pipes are a feature that is provided by z/OS UNIX System Services.

UNIX System Services named pipes are similar to non-persistent message queues, allowing unrelated application processes to communicate with each other. Named pipes provide a method of fast interprocess communication and also provide first in, first out (FIFOs) so that applications can establish a one way flow of data. The named pipe has to be explicitly deleted by the application when it is no longer needed. After you create a named pipe using the **mkfifo** command, the named pipe is visible in the UNIX System Services file system as shown in Figure 2-17 (size 0, file type = p).

```
mkfifo npipe; ls -l npipe
prw-r--r--   1 DFS      DFSGRP         0 Sep 24 16:11 npipe
```

Figure 2-17 UNIX System Services file system information for a UNIX System Services named pipe

Unlike a regular file, you can open a named pipe for write and read at the same time, allowing for overlapping processing by the pipe writer and the pipe reader. The contents of a named pipe resides in virtual storage. UNIX System Services does not write any data to the file system. If you read from a named pipe that is empty, the pipe reader is blocked (put on wait) until the pipe writer writes into the named pipe. UNIX System Services named pipes are read and written strictly in FIFO order. You cannot position a file marker to read a named pipe other than in FIFO order. UNIX System Services removes data from a named pipe upon read. You cannot go back and reread data from a named pipe. All writes to a named pipe are appended to whatever is currently in the named pipe. You cannot replace the contents of a named pipe by writing to it.

FTP support for UNIX System Services named pipes

When you use FTP to transfer data into a regular data set to load that data set into a DB2 table, you cannot start the DB2 load utility until the FTP data transfer of that data set is complete. For huge data volumes, the time needed for the FTP operation to complete can take many hours delaying the start of the DB2 load utility for that amount of time.

In z/OS V1R11, you can use FTP to send your data directly into a UNIX System Services named pipe while the DB2 load utility is simultaneously processing the same named pipe for read. If the named pipe is empty the load utility is put on wait until more data arrives through the named pipe writer. Figure 2-18 illustrates the data flow of this process.

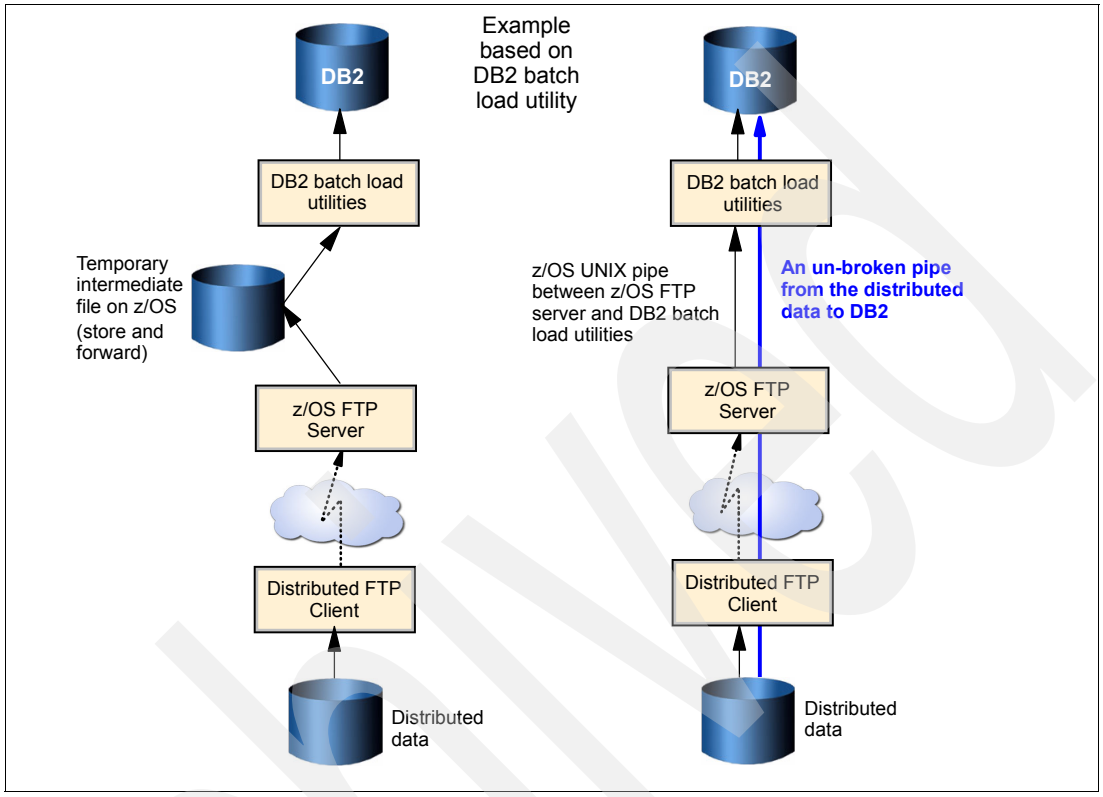


Figure 2-18 How FTP access to UNIX named pipes works

Using FTP, UNIX System Services named pipes in context with DB2 utilities (for example the DB2 LOAD and UNLOAD utilities) are just some of the use cases for which UNIX System Services named pipes can be useful. You can determine when use UNIX System Services named pipes when doing so is useful for solving a business problem.

For example, you can use the FTP named pipe support or write your own UNIX System Services named pipe writer application to continuously and asynchronously collect and ship information from multiple sources on z/OS for further processing to an application that on the other end implements the UNIX System Services named pipe reader application. One pipe reader application is the DB2 LOAD utility. However, you can use named pipes from your own application programs. Using named pipes from a program is easy, because you can access named pipes as you do normal UNIX files.

For more information about FTP access to UNIX System Services named pipes, refer to *IBM z/OS V1R11 Communications Server TCP/IP Implementation Volume 2: Standard Applications*, SG24-7799.

2.6.3 IPSec encryption

You can use z/OS Communication Server IP Security (IPSec) to provide end-to-end encryption entirely configured in TCP/IP transparent to DB2 for z/OS client and server. Using IPSec in a DB2 for z/OS server environment is usually recommended when all traffic needs to

be encrypted. If only a few of your applications need to access secure data, then use AT-TLS and DB2 SSL for data encryption. Figure 2-19 illustrates the general z/OS IPSec architecture.

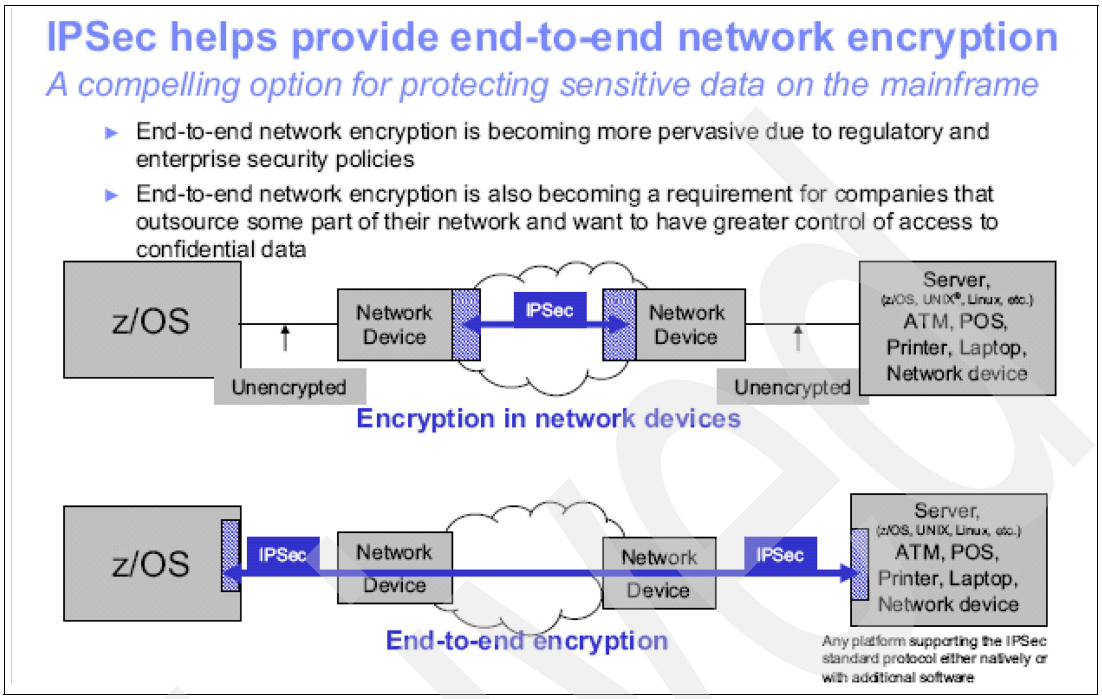


Figure 2-19 z/OS IPSec overview

To make IPSec solutions on z/OS more attractive, the z/OS Communications Server is designed to allow IPSec processing to take advantage of IBM System z Integrated Information Processors (zIIP). The zIIP IPSecurity design allows Communication Server to interact with z/OS Workload Manager to have a portion of its enclave Service Request Block (SRB) work directed to zIIP. Beginning with z/OS Communication Server V1R8, most of the processing related to security routines (Encryption and Authentication algorithms) run in enclave SRBs that can be dispatched to run on available zIIP processors.

In IPSec, the following work is eligible to run on zIIP processors:

- Inbound traffic is dispatched to run in enclave SRBs and, therefore, is 100% zIIP eligible.
- For outbound traffic, zIIP eligibility depends on the size of the messages being sent. Each message sent starts out on the application TCB. If the message is short, then all of the data is sent in a single operation under that TCB. However, if the message size requires segmentation, then all subsequent segments will be processed in an enclave SRB which is eligible to be dispatched on a zIIP processor.

Figure 2-20 summarizes the zIIP eligibility of the IPSec workload.

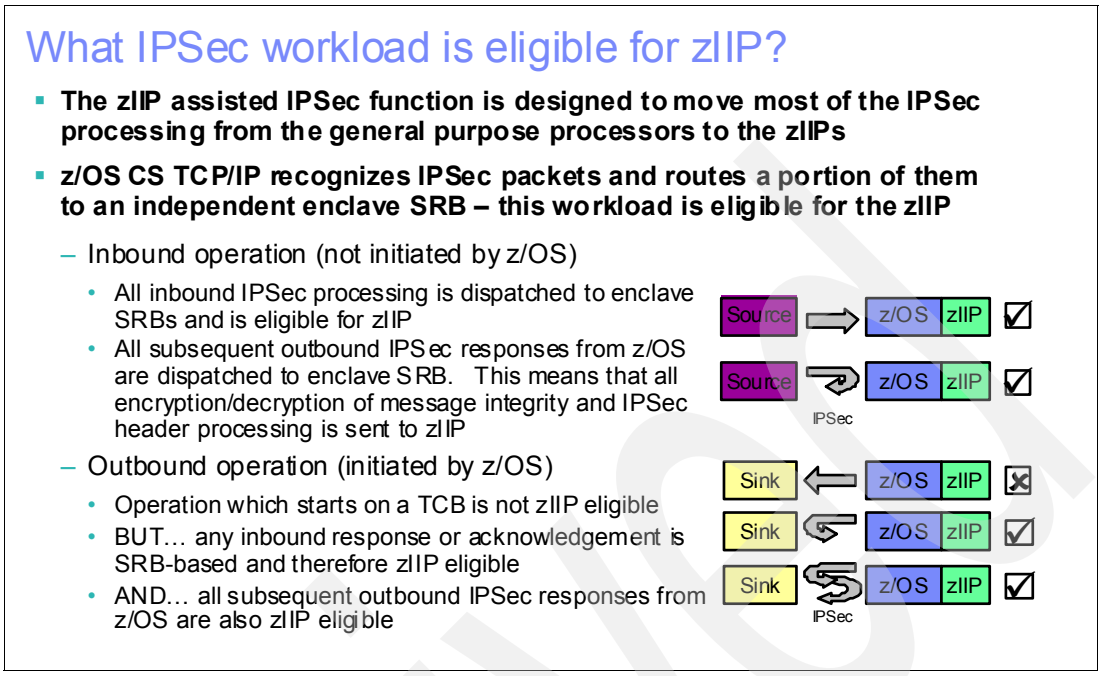


Figure 2-20 IPSEC zIIP eligibility

2.6.4 SSL encryption

DB2 for z/OS provides secure SSL encrypted communication for remote inbound and outbound connections by exploiting TCP/IP Application Transparent Transport Layer Security (AT-TLS), which is also known as *z/OS System SSL*.

In z/OS V1R12, AT-TLS is improved to consume up to 30% less CPU compared to z/OS V1R11 and earlier releases of z/OS. The improvement is achieved by eliminating one complete data copy from the overall instruction path. Figure 2-21 explains the z/OS V1R12 AT-TLS change and the results that were observed during our testing.

Archive

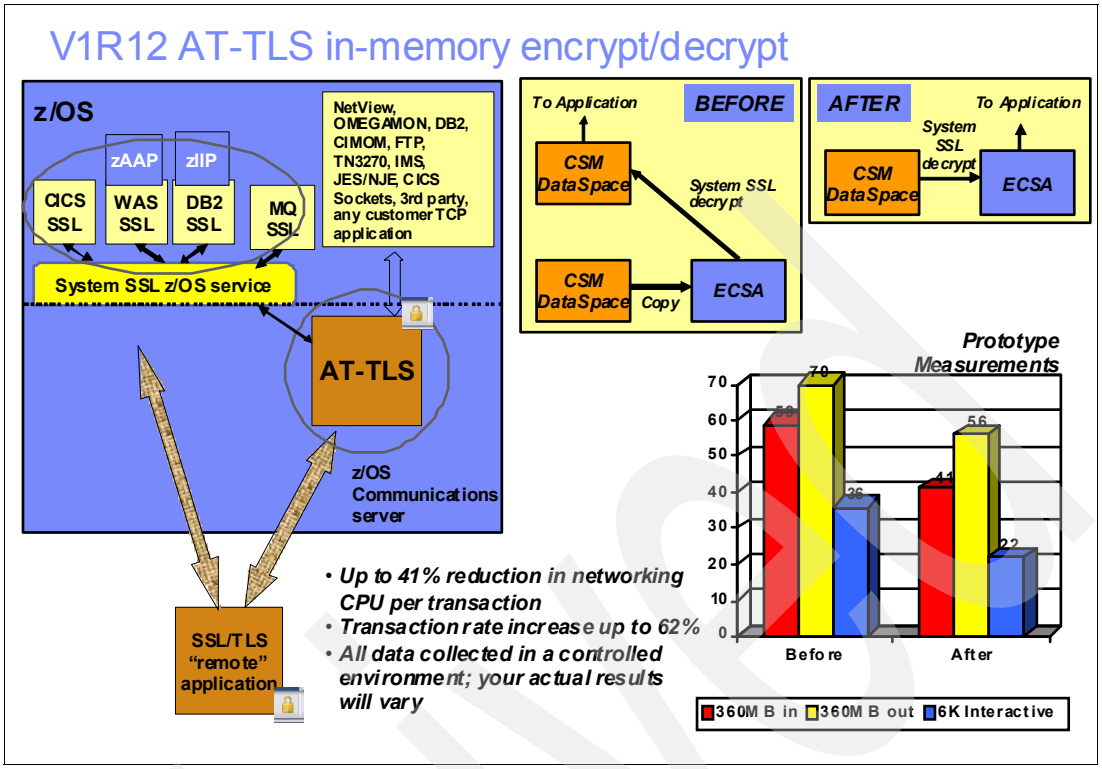


Figure 2-21 z/OS V1R12 AT-TLS in-memory encrypt / decrypt improvement

2.7 WLM

Since DB2 9 for z/OS became generally available, several WLM features and functions were introduced or shown to be useful in an operative DB2 environment. In this section, we introduce these WLM functions and features and provide information about interfaces that help in WLM administration and in DB2-related workload monitoring in WLM.

2.7.1 DSN_WLM_APPLENV stored procedure

DB2 for z/OS external stored procedures require WLM application environments (APPLENV) to be defined and activated in z/OS Workload Manager (WLM). If you want to define a WLM APPLENV, you normally have to use the WLM provided ISPF panel driven application in which you have to enter the WLM APPLENV definition manually. Manually entering a WLM APPLENV is difficult to integrate into existing system administration processes and procedures.

To address this issue and to provide an interface for batch scripting WLM APPLENV definitions that you need for executing your external procedures and functions, DB2 10 introduces the DSN_WLM_APPLENV stored procedure. You can call the DSN_WLM_APPLENV stored procedure from a DB2 client, such as the DB2 command line processor, to define, install, and activate a new WLM application environment in WLM.

The DSN_WLM_APPLENV call statement shown in Example 2-1 adds and activates WLM APPLENV DSNWLM_SAMPLE. At run time, WLM starts the APPLENV DSNWLM_SAMPLE using the DSNWLMS JCL procedure.

Example 2-1 DSN_WLM_APPLENV stored procedure call

```
CALL SYSPROC.DSN_WLM_APPLENV('ADD_ACTIVATE',
                             'ACTIVE',
                             'WLMNAME(DSNWLM_SAMPLE)
                              DESCRIPTION(DB2 SAMPLE WLM ENVIRONMENT)
                              PROCNAME(DSNWLMS)
                              STARTPARM(DB2SSN=&IWMSSNM,APPLENV=''DSNWLM_SAMPLE'')
                              WLMOPT(WLM_MANAGED)', ?, ?)
```

Upon successful completion the stored procedure returns the result shown in Figure 2-22.

```
RETURN_CODE: 0
MESSAGE: DSNTO23I DSNTWLMS ADD WLM APPLICATION ENVIRONMENT DSNWLM_SAMPLE
SUCCESSFUL

 APPLICATION ENVIRONMENT NAME : DSNWLM_SAMPLE
 DESCRIPTION                  : DB2 SAMPLE WLM ENVIRONMENT
 SUBSYSTEM TYPE               : DB2
 PROCEDURE NAME               : DSNWLMS
 START PARAMETERS             : DB2SSN=&IWMSSNM,APPLENV='DSNWLM_SAMPLE'

STARTING OF SERVER ADDRESS SPACES FOR A SUBSYSTEM INSTANCE:
  (x) MANAGED BY WLM
  ( ) LIMITED TO A SINGLE ADDRESS SPACE PER SYSTEM
  ( ) LIMITED TO A SINGLE ADDRESS SPACE PER SYSPLEX

  DSNTO23I DSNTWLMS ACTIVATE WLM POLICY WLMPOLY1 SUCCESSFUL
```

Figure 2-22 DSN_WLM_APPLENV output

For more information about the DSN_WLM_APPLENV stored procedure, refer to *DB2 10 for z/OS Application Programming and SQL Guide,* SC19-2969.

2.7.2 Classification of DRDA workloads using DB2 client information

JDBC, ODBC (DB2 CLI), and RRSAF DB2 clients can pass the following DB2 client information to DB2 for z/OS:

- Client user ID
- Client workstation name
- Client application or transaction name
- Client accounting string

When set by the client application, DB2 for z/OS stores the client information in the DB2 accounting trace records, which enables you to analyze application performance or to profile applications based on this client information.

This function addresses a the issue of applications connecting to DB2 using the same plan name, package collection ID, or authorization ID. For example, a JDBC application connecting to DB2 for z/OS through DRDA always uses the DISTSERV plan in DB2 and often uses the

NULLID package collection ID and the standard JDBC package names. The DB2 accounting trace records in such cases are not useful because they do not allow you to trace back the JAVA application. With DB2 client accounting information, for example, each JAVA application can set a unique client program name, allowing you to create accounting reports based on these client program names.

DB2 provides a variety of interfaces for setting client information. For example. you can set DB2 client information data source properties in WebSphere® Application Server, in the JDBC or DB2CLI properties file, you can invoke JDBC provided JAVA classes, DB2CLI, or RRSAF APIs for setting client information.

DB2 9 introduced the WLM_SET_CLIENT_INFO stored procedure through APAR PK74330. When this procedure is called, it uses the procedure input parameters to invoke the RRSAF SET_CLIENT_ID API to set the client information in DB2. For example, COGNOS can be configured to invoke the WLM_SET_CLIENT_INFO procedure to set individual client information for particular business intelligence (BI) workloads.

The WLM_SET_CLIENT_INFO stored procedure can also be used by local DB2 applications, which can be useful because it keeps the complexity of handling the RRSAF SET_CLIENT API away from your application logic.

In z/OS WLM, you can use the DB2 client information for service classification and RMF™ report class assignment, which takes resource accounting to a different level. It allows you to classify DRDA workloads based on client information set by the client application and enables you to use RMF for client information based reporting, application profiling, monitoring, and capacity planning.

The example shown in Figure 2-23 assigns the WLM service class DDFONL and the RMF report class ZSYNERGY when an SQL workload is run in the DB2 subsystem DB0B, and sets the DB2 client program name to ZSYNERGY.

```
Subsystem-Type  Xref  Notes  Options  Help
------------------------------------------------------------------------------
               Modify Rules for the Subsystem Type      Row 1 to 8 of 35
Command ===> _____________________________________________ Scroll ===> PAGE

Subsystem Type . : DDF         Fold qualifier names?   Y  (Y or N)
Description  . . . DDF Work Requests

Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                          More ===>
          --------Qualifier--------               -------Class--------
Action    Type       Name     Start                Service     Report
                                              DEFAULTS: DDFBAT      ________
 ____  1  SI         DB0B     ___                  DDFDEF      ________
 ____  2    PC         ZSYNERGY ___                  DDFONL      ZSYNERGY
```

Figure 2-23 WLM classification DRDA work based on program name

The example then uses the z/OS UNIX System Services command line processor to run the SQL shown in Example 2-2 to invoke the SET_CLIENT_INFO stored procedures and to run a simple SQL query.

Example 2-2 SET_CLIENT_INFO stored procedure invocation

```
update command options using c off;
connect to DBOB;
call SYSPROC.WLM_SET_CLIENT_INFO
('DataJoe','JoeWrkst','ZSYNERGY','ZSYNERGY ACCT STRING');
commit; -- commit activates the new client info
call sysproc.dsnwzp(?);
select count(*) from FCTEST;
TERMINATE;
```

While the query was running, we used the SDSF ENCLAVE display to verify the WLM settings assigned by WLM. Figure 2-24 shows the SQL query listed in Example 2-2. This query ran in WLM service class DDFONL with an RMF report class of ZSYNERGY.

```
 Display  Filter  View  Print  Options  Help
-------------------------------------------------------------
SDSF ENCLAVE DISPLAY  SC63     ALL                  LINE 1-9
COMMAND INPUT ===>                                      SC
NP   NAME             SSType Status   SrvClass Per PGN RptClass
     400000006D       DDF    ACTIVE   DDFONL     1     ZSYNERGY
```

Figure 2-24 SDSF enclave display of DRDA request

From the SDSF ENCLAVE display, we then displayed detailed enclave information. As shown in Figure 2-25, the SDSF process name field shows client program name that we set using the WLM_SET_CLIENT_INFO stored procedure invocation in Example 2-2.

```
 Enclave 400000006D on System SC63

 Subsystem type    DDF          Plan name         DISTSERV
 Subsystem name    DBOB         Package name      SYSSTAT
 Priority                       Connection type   SERVER
 Userid            DB2R5        Collection name   NULLID
 Transaction name               Correlation       db2jcc_appli
 Transaction class              Procedure name
 Netid                          Function name     DB2_DRDA
 Logical unit name              Performance group
 Subsys collection              Scheduling env
 Process name      ZSYNERGY
 Reset             NO
```

Figure 2-25 SDSF enclave information for client program ZSYNERGY

Archived

We then used the DB2 display thread command to display the entire DB2 client information used at query execution time. The command output shown in Figure 2-26 shows all client information that we previously set using the WLM_SET_CLIENT_INFO stored procedure.

```
-dis thd(*)
 DSNV401I  -DBOB DISPLAY THREAD REPORT FOLLOWS -
 DSNV402I  -DBOB ACTIVE THREADS -
 NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
 SERVER   SW *    24 db2jcc_appli DB2R5    DISTSERV 00A0   151
  V437-WORKSTATION=JoeWrkst, USERID=DataJoe,
       APPLICATION NAME=ZSYNERGY
  V429 CALLING PROCEDURE=SYSPROC.DSNWZP,
       PROC=        , ASID=0000, WLM_ENV=DSNWLM_NUMTCB1
```

Figure 2-26 DB2 display thread command showing DB2 client information

2.7.3 WLM blocked workload support

If the CPU utilization of your system is at 100%, workloads with low importance (low dispatch priority) might not get dispatched anymore. Such low importance workloads are CPU starved and are blocked from being dispatched by WLM due to its low dispatching priority. They are also known as *blocked workloads*. Blocked workloads can impact application availability if the low priority work (for example, a batch job) holds locks on DB2 resources that are required by high priority workloads (for example, CICS, IMS or WebSphere Application Server transactions).

To address this issue, z/OS V1R9 delivered a blocked workload support function, which was retrofitted into z/OS V1R7 and V1R8 through APAR OA17735.

The WLM blocked workload support temporarily promotes a blocked workload to a higher dispatch priority provided the corresponding z/OS work unit (TCB or SRB) is ready-to-run but does not get CPU service because of its low dispatch priority. WLM blocked workload support does not consider swapped out address spaces for promotion.

For details about WLM blocked workload support, refer to WSC FLASH *z/OS Availability: Blocked Workload Support*, which is available at:

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10609

IEAOPTxx function

The blocked workload support provides installation control over the amount of CPU blocked workloads can receive. The following IEAOPTxx parameters are introduced with this support:

- BLWLINTHD

  Parameter BLWLINTHD specifies the threshold time interval in seconds for which a blocked address space or enclave must wait before being considered by WLM for promotion. You can specify a value ranging from 5 to 65535 seconds. The BLWINTHD default value is 20 seconds.

- BLWLTRPCT

  Parameter BLWLTRPCT specifies in units of 0.1% how much of the CPU capacity is to be used to promote blocked workloads. This parameter does not influence the amount of CPU service that a single blocked address space or enclave is given. Instead, this parameter influences how many different address spaces or enclaves can be promoted at the same point in time. If the value specified with this parameter is not large enough,

blocked workloads might need to wait longer than the time interval defined by BLWLINTHD. You can specify a value ranging from 0 up to 200 (up to 200 x 0.1% = 20%). If you specify 0 you disable blocked workload support. The BLWTRPCT default value is 5 (5 x 0.1% = 0.5%).

For details about the IEAOPTxx parameters for WLM blocked workload support, refer to *MVS Initialization and Tuning Reference*:

http://publibz.boulder.ibm.com/epubs/pdf/iea2e2a1.pdf

Promotion for chronic contentions

z/OS V1R10 introduces changes to the WLM IWMCNTN service that allow resource managers, such as DB2, to tell WLM that a long lasting contention situation exists. WLM in turn manages the dispatching priority of the resource holder according to the goals of the most important waiter. DB2 10 uses the IWMCNTN WLM service to resolve chronic contention situations that potentially lead to deadlock or timeout situations due to DB2 lock holders that are blocked for long periods caused by circumstances such as lack of CPU.

IEAOPTxx parameter MaxPromoteTime

z/OS V1R10 introduces the IEAOPTxx z/OS parmlib parameter MaxPromoteTime to limit the time a resource holder (can be an address space or an enclave) that is suffering from a chronic contention is allowed to run promoted while it is causing a resource contention. You specify the time in units of 10 seconds. When MaxPromoteTime is exceeded, the promotion is cancelled. During promotion the resource holder runs with the highest priority of all resource waiters to guarantee the needed importance. Furthermore, the resource holder's address space (including the address space that is associated with an enclave) is be swapped out. The default value for the MaxPromote parameter is 6, which results in a maximum time a resource holder can run promoted of 60 seconds (6 x 10 seconds).

SDSF support for chronic resource contention promoted workloads

You can configure SDSF to provide a PROMOTED column in the DISPLAY ACTIVE (DA) and ENCLAVE (ENC) panel to show whether an address space or an enclave is promoted due to a chronic resource contention.

SMF type 30 record

The processor accounting section of the SMF type 30 record, field SMF30CRP, contains the CPU time that a resource holder consumed while being promoted due to a chronic resource contention.

For details about WLM and SRM resource contention and the WLM IWMCNTN service, refer to *z/OS Version 1 Release 10 Implementation*, SG24-7605.

RMF support for blocked workloads

In support of blocked workload, RMF has extended the SMF record types 70-1 (CPU activity) and 72-3 (workload activity) to contain information about blocked workloads and accordingly provided extensions to the CPU Activity and Workload Activity report.

Archived

RMF CPU activity report

The RMF postprocessor CPU activity report provides a new section with information about blocked workloads. A sample of the new blocked workload section is shown in Figure 2-27.

```
BLOCKED WORKLOAD ANALYSIS
 OPT PARAMETERS: BLWLTRPCT (%) 20.0 PROMOTE RATE: DEFINED  173 WAITERS FOR PROMOTE: AVG  0.153
                 BLWLINTHD       5              USED (%)  13                       PEAK     2
```

Figure 2-27 Blocked workload RMF CPU activity report

- PROMOTE RATE
  - DEFINED: Number of blocked work units which can be promoted in their dispatching priority per second. This value is derived from IEAOPTxx parameter BLWLTRPCT.
  - USED (%): The utilization of the defined promote rate during the reporting interval. This value is calculated per RMF cycle and averaged for the whole RMF interval. It demonstrates how many trickles were actually given away (in percent of the allowed maximum) for the RMF interval.
- WAITERS FOR PROMOTE
  - Average: Number of TCBs/SRBs and enclaves found blocked during the interval and not promoted according to IEAOPTxx parameter BLWLINTHD.
  - PEAK: The maximum number of TCBs/SRBs and enclaves found blocked and not promoted during the interval according to the IEAOPTxx parameter BLWLINTHD. The AVG value might be quite low although there were considerable peaks of blocked workload. Thus, the peak value is listed as well.

As long as WAITERS FOR PROMOTE is greater than 0, the system has work being blocked longer than the BLWLINTHD setting. In such a case, it might be advisable to increase BLWLTRPCT. If there are still problems with blocked work holding resources for too long even though there are no waiters in the RMF data, then decreasing the BLWLINTHD setting might be advisable.

RMF workload activity report

The RMF postprocessor workload activity report provides the CPU time, that transactions of a certain service or report class were running at a promoted dispatching priority. Figure 2-28 provides an example of a workload activity report promoted section.

```
 SERVICE CLASS=STCHIGH    RESOUCE GROUP=*NONE
 CRITICAL     =NONE
 DESCRIPTION  =High priority fo STC workloads
 ---------------------------------------------- SERVICE CLASS(ES)
       --PROMOTED--
       BLK    1.489
       ENQ    0.046
       CRM    5.593
       LCK    0.000
```

Figure 2-28 Blocked workload RMF workload activity report

Archived

The workload activity promoted section shows the CPU time in seconds that transactions in this group were running at a promoted dispatching priority, separated by the reason for the promotion. RMF currently reports the following promotion reasons:

- BLK: CPU time in seconds consumed while the dispatching priority of work with low importance was temporarily raised to help blocked workloads
- ENQ: CPU time in seconds consumed while the dispatching priority was temporarily raised by enqueue management because the work held a resource that other work needed
- CRM: CPU time in seconds consumed while the dispatching priority was temporarily raised by chronic resource contention management because the work held a resource that other work needed
- LCK: In HiperDispatch mode, the CPU time in seconds consumed while the dispatching priority was temporarily raised to shorten the lock hold time of a local suspend lock held by the work unit

2.7.4 Extend number of WLM reporting classes to 2,048

With systems and environments becoming larger, more reporting classes are needed in WLM. In z/OS V1R11, the number of report classes is increased from 999 to 2,048. This increase is expected to allow more fine-grained reporting of your DB2 workloads.

2.7.5 Support for enhanced WLM routing algorithms

z/OS V1R11 Communications Server enhances server-specific workload WLM recommendations that are used by the sysplex distributor to balance workload when DISTMETHOD SERVERWLM is configured on the VIPADISTRIBUTE statement. These configuration parameters enable WLM to do the following tasks:

- Direct more workload targeted for zIIP or zAAP specialty processors to systems that have these more affordable processors available, thereby reducing the overall cost of running those workloads. For this function, a minimum of an IBM zIIP or IBM zAAP processor is required.
- Consider the different importance levels of displaceable capacity when determining server-specific recommendations.

WLM can use the configuration options for server-specific recommendations only when all systems in the sysplex are at z/OS V1R11 or later.

For more information about the support for enhanced WLM routing algorithms, refer to *z/OS V1R11.0 Communications Server New Function Summary z/OS V1R10.0-V1R11.0,* GC31-8771.

2.8 Using RMF for zIIP reporting and monitoring

DB2 for z/OS no longer provides information about zIIP eligible time that was processed on a central processor (CP) due to unavailability of sufficient zIIP resources. As a consequence the information provided by DB2 for z/OS can no longer be used to determine whether further zIIP capacity might be required for TCO optimization. To monitor zIIP eligible time that was dispatched to be processed on a CP you therefore need to use the RMF report options. In this part of this book we outline how to set up WLM and to use RMF to monitor zIIP usage for DRDA and z/OS batch workloads using RMF report class definitions.

Using RMF for zIIP reporting and monitoring: If you want to use RMF for zIIP reporting and monitoring, you need to collect the SMF RMF records as described in *Effective zSeries Performance Monitoring Using Resource Measurement Facility*, SG24-6645.

2.8.1 DRDA workloads

In 2.7.2, “Classification of DRDA workloads using DB2 client information” on page 32, we define a service classification for DB2 subsystem DB0B for SQL workloads that have their DB2 client program name set to a value of ZSYNERGY and subsequently ran an SQL workload under the ZSYNERGY client program name. Upon SQL completion we extracted the SMF RMF records and created an RMF workload activity report for RMF report class ZSYNERGY. The RMF report for that report class is shown in Figure 2-29.

```
REPORT BY: POLICY=WLMPOL                  REPORT CLASS=ZSYNERGY


 -TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---
 AVG       0.00  ACTUAL          2.24.813  SSCHRT   1.7  IOC         0   CPU    0.442  CP       0.01
 MPL       0.00  EXECUTION       2.24.812  RESP     1.3  CPU     12551   SRB    0.000  AAPCP    0.00
 ENDED        2  QUEUED                 1  CONN     1.2  MSO         0   RCT    0.000  IIPCP    0.00
 END/S    0.00  R/S AFFIN              0  DISC     0.0  SRB         0   IIT    0.000
 #SWAPS      0  INELIGIBLE             0  Q+PEND   0.1  TOT     12551   HST    0.000  AAP      0.00
 EXCTD       0  CONVERSION             0  IOSQ     0.0  /SEC        3   AAP    0.000  IIP      0.00
 AVG ENC  0.00  STD DEV        1.21.329                                 IIP    0.178
 REM ENC  0.00                                    ABSRPTN  1303
 MS ENC   0.00                                             TRX SERV 1303
```

Figure 2-29 RMF workload activity report for RMF report class ZSYNERGY

2.8.2 Batch workloads

You can use RMF reports for batch workload monitoring and zIIP capacity planning. As a prerequisite for RMF reporting, you need to perform the following tasks:

- Define a WLM service classification rule for the batch job or the group of jobs that you want to monitor. For ease of use, make use of RMF report classes in your WLM classification.
- Configure SMF and RMF to gather the RMF SMF records.
- Upon successful job completion, create an RMF II workload activity report.

WLM classification

In the WLM service classification shown in Figure 2-30, we assign the WLM service class BATCHMED to any job that runs in class A with a job name starting with RS. For ease of RMF reporting, we assign a report class of RUNSTATS.

```
 Subsystem-Type  Xref  Notes  Options  Help
------------------------------------------------------------------------------
                Modify Rules for the Subsystem Type       Row 1 to 8 of
Command ===> _____________________________________________ Scroll ===> PAGE

Subsystem Type . : JES         Fold qualifier names?   Y  (Y or N)
Description  . . . Batch Jobs

Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                        More ===>
          --------Qualifier--------              -------Class--------
Action    Type       Name     Start                Service     Report
                                              DEFAULTS: BATCHLOW    BATCHDEF
 ____  1  TC         A        ___                  BATCHLOW    LOWJES2
 ____  2    TN         RS*      ___                BATCHMED    RUNSTATS
```

Figure 2-30 WLM classification for batch job

When we ran job RSFCTEST in class A we used the SDSF display active panel to confirm the service class and report class assignment (see Figure 2-31 for details).

```
 Display  Filter  View  Print  Options  Help

------------------------------------------------------------------------------
 SDSF DA SC63     SC63     PAG  0  CPU/L/Z   4/  3/  0  LINE 1-1 (1)
 COMMAND INPUT ===>                                      SCROLL ===> CSR
 NP   JOBNAME  U% CPUCrit StorCrit RptClass MemLimit   Tran-Act   Tran-Res Spin
      RSFCTEST 12 NO      NO       RUNSTATS  16383PB    0:00:01    0:00:01 NO
```

Figure 2-31 SDSF display active batch WLM classification

RMF workload activity report

Upon successful job completion we created an RMF workload activity report for the RUNSTATS report class using the JCL shown in Example 2-3.

Example 2-3 JCL RMF workload activity report

```
//RMFPP    EXEC PGM=ERBRMFPP
//MFPINPUT DD   DISP=SHR,DSN=DB2R5.RS.RMF
//MFPMSGDS DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//SYSIN    DD   *
  SYSRPTS(WLMGL(RCPER(RUNSTATS)))
  RTOD(0220,0225)
  DINTV(0100)
  SUMMARY(INT,TOT)
  NODELTA
  NOEXITS
```

```
MAXPLEN(225)
SYSOUT(A)
```

Figure 2-32 shows the RMF workload activity report for the RUNSTATS report class.

```
REPORT BY: POLICY=WLMPOL                          REPORT CLASS=RUNSTATS                    PERIOD=1

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---
AVG       0.01  ACTUAL             5.016  SSCHRT   0.2  IOC        63   CPU    0.359  CP       0.01
MPL       0.01  EXECUTION          4.892  RESP     0.4  CPU     10182   SRB    0.000  AAPCP    0.00
ENDED        1  QUEUED               123  CONN     0.3  MSO      1173   RCT    0.000  IIPCP    0.00
END/S     0.00  R/S AFFIN              0  DISC     0.0  SRB         3   IIT    0.002
#SWAPS       1  INELIGIBLE             0  Q+PEND   0.1  TOT     11421   HST    0.000  AAP      0.00
EXCTD        0  CONVERSION             0  IOSQ     0.0  /SEC       22   AAP    0.000  IIP      0.06
AVG ENC   0.00  STD DEV                0                                IIP    0.329
REM ENC   0.00                                          ABSRPTN  2347
MS ENC    0.00                                          TRX SERV 2347

GOAL: EXECUTION VELOCITY 20.0%     VELOCITY MIGRATION:   I/O MGMT   100%     INIT MGMT  100%

RESPONSE TIME EX   PERF  AVG   --EXEC USING%-------
 SYSTEM                VEL% INDX ADRSP  CPU AAP IIP I/O  TOT

 SC63        --N/A--     100  0.2   0.0  0.0 0.0 2.7  16  0.0

 EXEC DELAYS % -----------  -USING%-  --- DELAY % ---    %
                            CRY CNT  UNK IDL CRY CNT  QUI
                            0.0 0.0   81 0.0 0.0 0.0  0.0
```

Figure 2-32 RMF workload activity report for the RUNSTATS report class

The RMF report shown in Figure 2-32 provides information about zIIP utilization in the SERVICE TIME and the APPL report blocks. In our example we ran a RUNSTATS utility to collect standard statistics for a table space (all tables and indexes of that table space). In DB2 10, you can expect RUNSTATS to redirect some of its processing to run on zIIP processors.

The RMF SERVICE TIME report block shows a total CPU time of 0.359 seconds (includes CP and zIIP) and a total IIP (zIIP) time of 0.329 seconds which indicates a RUNSTATS zIIP redirect ratio of about 91% (0.329*100/0.359). If you want to verify whether there was any zIIP eligible time processed on a CP, review the IIPCP (IIP processed on CP in percent) information that is provided in the RMF APPL report block.

In our RUNSTATS utility example, the value for IIPCP is zero, indicating that there was no zIIP eligible work processed on a CP. Therefore, we come to the conclusion that there were sufficient zIIP resources available at the time of RUNSTATS utility execution.

Archive

To reconfirm that the RUNSTATS utility zIIP eligible time matches the zIIP eligible time shown in the RMF report, we created a DB2 accounting report for the interval in question. The special engine time (SE CPU TIME) of the accounting report shown in Figure 2-33 matches the zIIP eligible time of the RMF workload activity report shown in Figure 2-32.

```
MAINPACK   : DSNUTIL                CORRNMBR: 'BLANK'         LUW INS: C6A4809D233B
PRIMAUTH   : DB2R5                  CONNTYPE: UTILITY         LUW SEQ:           57
ORIGAUTH   : DB2R5                  CONNECT : UTILITY

TIMES/EVENTS  APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   ELAPSED TIME
------------  ----------  ----------  ----------    --------------------  ------------
ELAPSED TIME    4.840189    4.778304         N/P    LOCK/LATCH(DB2+IRLM)       0.000000
 NONNESTED      4.840189    4.778304         N/A     IRLM LOCK+LATCH           0.000000
 STORED PROC    0.000000    0.000000         N/A     DB2 LATCH                 0.000000
 UDF            0.000000    0.000000         N/A    SYNCHRON. I/O              0.057179
 TRIGGER        0.000000    0.000000         N/A     DATABASE I/O              0.007112
                                                     LOG WRITE I/O             0.050067
CP CPU TIME     0.021537    0.008329         N/P    OTHER READ I/O             4.053438
 AGENT          0.021537    0.008329         N/A    OTHER WRTE I/O             0.000000
  NONNESTED     0.021537    0.008329         N/P    SER.TASK SWTCH             0.021068
  STORED PRC    0.000000    0.000000         N/A     UPDATE COMMIT             0.011318
  UDF           0.000000    0.000000         N/A     OPEN/CLOSE                0.000000
  TRIGGER       0.000000    0.000000         N/A     SYSLGRNG REC              0.009750
 PAR.TASKS      0.000000    0.000000         N/A     EXT/DEL/DEF               0.000000
                                                     OTHER SERVICE             0.000000
 SECP CPU       0.000000         N/A         N/A    ARC.LOG(QUIES)             0.000000
                                                    LOG READ                   0.000000
SE CPU TIME     0.329314    0.329314         N/A    DRAIN LOCK                 0.000000
```

Figure 2-33 DB2 RUNSTATS utility accounting report

The RMF zIIP support is explained in detail in *DB2 9 for z/OS Technical Overview*, SG24-7330.

2.9 Warehousing on System z

For performance and security reasons many z/OS customers (and especially those who have most of their business critical information stored in DB2 for z/OS) want to apply their BI processes as close as possible to the place their data is stored. Applying BI processes as close as possible to where your data is stored provides tight integration with existing resources on the z/OS platform and can assist you in providing high performance throughput BI processes that comply with the requirement of a real-time operational data store.

2.9.1 Cognos on System z

Cognos® for Linux on System z gives you the opportunity to host a Cognos server environment on the System z platform. Cognos for System z can be redirected to run on another kind of speciality engine, the Integrated Facility for Linux (IFL). Like zIIPs and zAAPs an IFL is just another type of speciality engine that you can take advantage of to run Linux on System z applications just next to the z/OS LPAR that hosts your business critical information stored in DB2 for z/OS. The close proximity of the Cognos for Linux on System z server to DB2 for z/OS (both reside in the same physical box) provides fast access to your data and allows you to apply BI processes as efficiently and close to your DB2 for z/OS data as possible.

zIIP eligibility

Cognos connects to DB2 z/OS as a DRDA application requestor. Therefore, SQL DML requests can benefit from DRDA zIIP redirect. If the SQL submitted by Cognos qualifies for parallel query processing, the additional CPU required for parallel processing is eligible to be redirected onto a zIIP processor and provides additional TCO savings.

Synergy with System z

Using System z for your Cognos for Linux on System z environment, you benefit from the System z hardware infrastructure that already is provided to support high or even continuous availability for your existing z/OS applications. Since Cognos on System z version 8.4, you can store all Cognos data in DB2 for z/OS. Previous to version 8.4, you could not store the Cognos content store back-end data in DB2 for z/OS. Using DB2 for z/OS for Cognos, you can take advantage of existing DB2 backup and recovery procedures that are already in place to support the high availability requirements of the z/OS platform.

2.10 Data encryption

z/OS has a large breadth of cryptographic capabilities. The cryptographic capabilities are highly available and scalable and can take advantage of System z technologies, such as Parallel Sysplex and Geographically Dispersed Parallel Sysplex™ (GDPS). Key management is simpler on z/OS because a central keystore is easier to maintain than having many distributed keystores using multiple Hardware Security Modules. In addition, encryption keys are highly secure. For secure key processing, the key never appears within or leaves the System z server in the clear.

z/OS has a great depth of encryption technologies, with support for or support planned for the following encryption standards:

- Advanced Encryption Standard (AES)
- Data Encryption Standard (DES) and Triple DES
- Secure Hashing Algorithms (SHA)
- Public Key Infrastructure (PKI)
- Elliptical Curve Cryptography (ECC)
- Galois/Counter Mode encryption for AES (GCM)
- Elliptic Curve Diffie-Hellman key derivation (ECDH)
- Elliptic Curve Digital Signature Algorithm (ECDSA)
- Keyed-Hash Message Authentication Code (HMAC)
- RSA algorithms

2.10.1 IBM TotalStorage for encryption on disk and tape

DS8000 and DS8700 features, along with z/OS and IBM Tivoli® Key Lifecycle Manager functions, allow DS8000 and DS8700 storage controllers to encrypt sensitive data in place. Drive-level encryption is designed to have no performance impact, to be transparent to the application and to the server, and to help minimize the costs associated with encrypting data at rest with a solution that is simple to deploy and simple to maintain.

Two tape encryption options are also available:

- IBM System Storage tape drives (TS1120 and TS1130) with integrated encryption and compression are intended to satisfy the needs for high volume data archival and back-up/restore processes. Handling encryption and compression in the tape drive offloads processing from System z servers to the tape subsystems, freeing up cycles for your mission-critical work-loads.

- The Encryption Facility for z/OS can help protect valuable assets from exposure by encrypting data on tape prior to exchanging it with trusted business partners. The Encryption Facility for z/OS V1.2 with OpenPGP support provides a host-based security solution designed to help businesses protect data from loss and inadvertent or deliberate compromise.

2.11 IBM WebSphere DataPower

IBM WebSphere DataPower® is a hardware solution that is well known for its security features and its high throughput in data transformations (for example, XML processing). WebSphere DataPower can access existing data through WebSphere MQ and IMS Connect. You can configure it to function as a multiprotocol gateway, because it supports protocols such as SOAP, REST, HTTP, XML, FTP, MAILTO, SMTP, SSH, and other protocols.

With the ODBC option enabled, WebSphere DataPower can access DB2 for z/OS through DRDA. Through DRDA access, WebSphere DataPower can host IBM Data Web Services (implemented as XSL stylesheets that query DB2 and return the DB2 response as an XML document to the Web Services consumer).

Figure 2-34 illustrates the DataPower DB2 access capabilities.

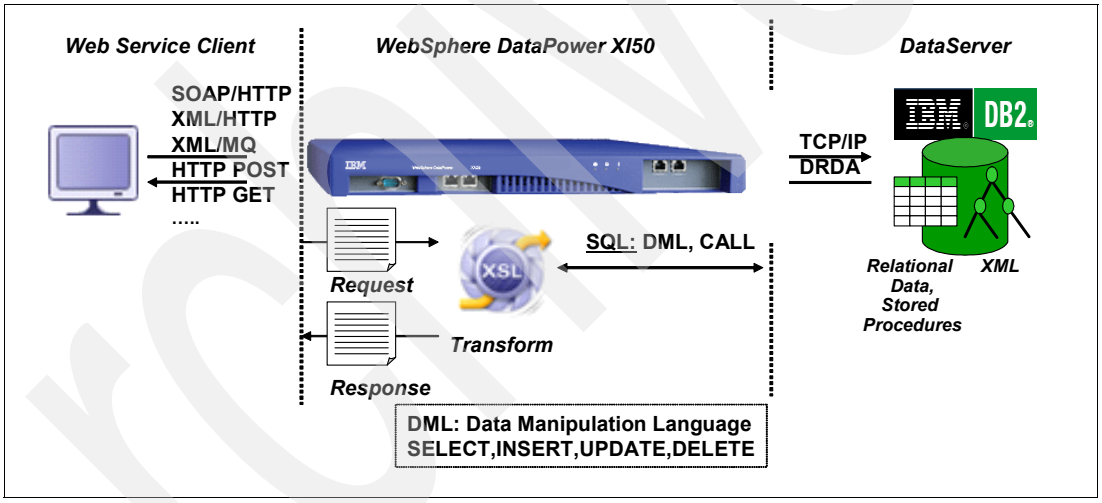


Figure 2-34 WebSphere DataPower DRDA capabilities

If you use WebSphere DataPower to access existing data, you need to set up your DataPower appliances to cater for workload balancing and failover capability to avoid an SPOF. Starting with WebSphere DataPower firmware level 3.8.1, you can configure a Sysplex Distributor Target Control Service on your WebSphere DataPower appliance. The Sysplex Distributor Target Control Service on the DataPower appliance establishes control connections with the z/OS Sysplex Distributor that allow the z/OS Sysplex Distributor to intelligently distribute traffic across multiple DataPower appliances. By letting z/OS Sysplex Distributor manage your WebSphere DataPower appliances you implicitly reuse the availability features that are already in place for your Parallel Sysplex Distributor infrastructure.

Archive

2.12 Additional zIIP and zAAP eligibility

DB2 for z/OS began using zIIP speciality processors in Version 8 and continued to improve total cost of ownership (TCO) by further using zIIP engines in DB2 9. DB2 10 continues this trend and provides additional zIIP workload eligibility as we describe in this section.

Figure 2-35 summarizes the availability of the specialty engines and their applicability to DB2 workloads.

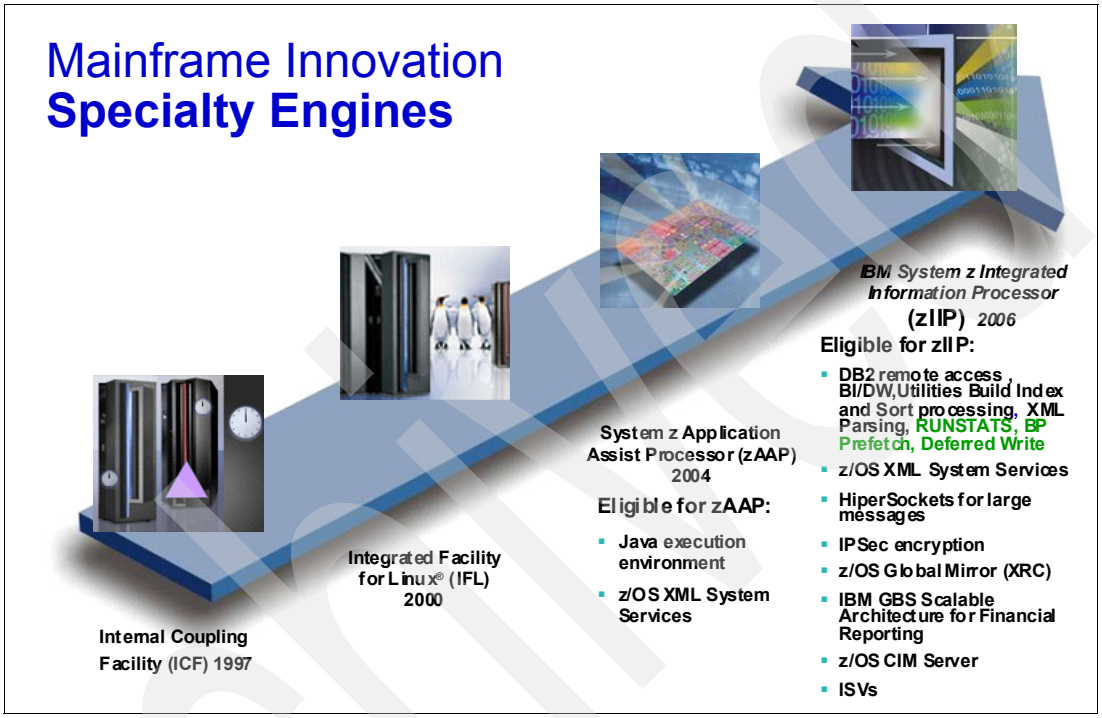


Figure 2-35 Specialty engines applicability

For further information about zIIP eligibility as of DB2 9 for z/OS, refer to *DB2 9 for z/OS Technical Overview*, SG24-7330.

Archive

Figure 2-36 provides a quick comparison of characteristics of zAAP and zIIP.

zAAP	zIIP
Introduced in **2004**.	Introduced in **2006**.
System z Application Assist Processor (originally the zSeries Application Assist Processor). Available on IBM zEnterprise 196 (z196), IBM System z10™, and IBM System z9® servers, and IBM eServer™ zSeries® 990 and 890 (z990, z890).	**System z Integrated Information Processor** Available on IBM zEnterprise, System z10, and System z9 servers.
Intended to help implement new **application** technologies on System z, such as Java and XML.	Intended to help integrate **data and transaction** processing across the enterprise
Exploiters include: ▪ Java via the IBM SDK (IBM Java Virtual Machine (JVM)), such as portions of: – **WebSphere Application Server** – **IMS™** – **DB2** – **CICS®** – **Java batch** – **CIM Client applications** ▪ z/OS XML System Services, such as portions of: – **DB2 9 (New Function Mode), and later** – **Enterprise COBOL V4.1, and later** – **Enterprise PL/I V3.8, and later** – **IBM XML Toolkit for z/OS, V1.9 and later** – **CICS TS V4.1**	**Exploiters include, portions of:** ▪ DB2 V8, DB2 9, DB2 10 for z/OS – **Data serving** – **Data Warehousing** – **Select utilities** ▪ z/OS Communications Server – **Network encryption** – **HiperSockets for large messages** ▪ z/OS XML System Services – **DB2 9 New Function Mode** ▪ z/OS Global Mirror (XRC), System Data Mover (SDM) ▪ IBM GBS Scalable Architecture for Financial Reporting ▪ z/OS CIM server New zAAP on zIIP capability

Figure 2-36 Comparison of zAAP and zIIP

2.12.1 DB2 10 parallelism enhancements

When you use DB2 for z/OS to run parallel queries, portions of such parallel SQL requests are zIIP eligible and can be directed to run on a zIIP speciality processor.

In DB2 10, more queries qualify for query parallelism which in turn introduces additional zIIP eligibility. For further details on DB2 10 additional query parallelism refer to 13.21, “Parallelism enhancements” on page 593.

2.12.2 DB2 10 RUNSTATS utility

In DB2 10, portions of the RUNSTATS utility are eligible to be redirected to run on a zIIP processor. The degree of zIIP eligibility depends upon the statistics that you gather. If you run RUNSTATS with no additional parameters, the zIIP eligibility can be up to 99.9%. However, if you require more complex statistics (for example, frequency statistics), the degree of zIIP eligibility is less.

Depending on the characteristics and complexity of your SQL workload, you might need to collect additional statistics to support the application performance that you have to deliver to comply with your service level agreements (SLA). Therefore, you might find a varying degree of zIIP eligibility when you execute your RUNSTATS utility workload.

2.12.3 DB2 10 buffer pool prefetch and deferred write activities

Buffer pool prefetch, which includes dynamic prefetch, list prefetch, and sequential prefetch activities, is 100% zIIP eligible in DB2 10. DB2 10 zIIP eligible buffer pool prefetch activities are asynchronously initiated by the database manager address space (DBM1) and are

executed in a dependent enclave that is owned by the system services address space (MSTR). Because asynchronous services buffer pool prefetch activities are not accounted to the DB2 client, they show up in the DB2 statistics report instead.

Deferred write is also eligible for zIIP.

Attention: With APAR PM30468 (PTF UK64423), prefetch and deferred write CPU, when running on a zIIP processor, is reported by WLM under the DBM1 address space, not under the MSTR. The RMF example is still useful to verify the actual level of zIIP engine utilization.

Statistics report

DB2 10 buffer pool prefetch activities are shown in the CPU times block of the DB2 statistics report, specifically in the preemptilbe IIP SRB field of the database services address space. In the report shown in Figure 2-37, the CPU TIMES block of the DB2 statistics report shows a preemptible zIIP SRB time spent on buffer pool prefetch activities for the DBM1 address space of 0.513212 seconds.

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME	PREEMPT IIP SRB
SYSTEM SERVICES ADDRESS SPACE	0.004481	0.000487	0.001857	0.006825	N/A
DATABASE SERVICES ADDRESS SPACE	0.011464	0.006950	0.016452	0.034866	0.513212
IRLM	0.000001	0.000000	0.064572	0.064573	N/A
DDF ADDRESS SPACE	0.000168	0.000034	0.000049	0.000251	0.000000
TOTAL	0.016115	0.007470	0.082930	0.106515	0.513212

Figure 2-37 OMEGAMON® PE statistics report DBM1 zIIP usage

MSTR address space RMF workload activity report

Because DB2 10 buffer pool prefetch activities are executed in a dependent enclave that is owned by MSTR address space, the zIIP processor time consumed for these buffer pool prefetches are accounted to the MSTR address space and, as such, are visible only in the RMF workload activity report for the WLM service class or report class of the MSTR address space.

Archive

To create an individual RMF workload activity report just for the MSTR address space, change the WLM service classification rule to assign a unique report class to the MSTR address space. In our scenario, we used the WLM service classification shown in Figure 2-38 to assign a unique report class named DB0BMSTR.

```
 Subsystem-Type  Xref  Notes  Options  Help
------------------------------------------------------------------------------
                 Modify Rules for the Subsystem Type      Row 2 to 9 of 19
Command ===> _____________________________________________ Scroll ===> PAGE

Subsystem Type . : STC         Fold qualifier names?   Y  (Y or N)
Description  . . . Started Tasks

Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                        More ===>
          --------Qualifier--------             -------Class--------
Action    Type       Name     Start                Service     Report
                                              DEFAULTS: STC         RSYSDFLT
 ____  1  TN         DB0BMSTR ___                  STCHI       DB0BMSTR
```

Figure 2-38 WLM report class assignment for the MSTR address space

The SDSF DISPLAY ACTIVE (DA) panel RptClass field shown in Figure 2-39 confirms that the correct WLM report class was assigned for the DB0BMSTR address space.

```
SDSF DA SC63     SC63     PAG  0  CPU/L/Z   3/  3/  0  LINE 1-1 (1)
COMMAND INPUT ===>                                        SCROLL ===> CS
NP   JOBNAME  U% CPUCrit StorCrit RptClass MemLimit   Tran-Act   Tran-Res Sp
     DB0BMSTR 01 NO      NO       DB0BMSTR  16383PB  147:07:47  147:07:47 NO
```

Figure 2-39 SDSF display active panel to verify MSTR report class assignment

After the RMF interval we were interested in was written to SMF we created the RMF report shown in Figure 2-40 to review the zIIP processor time used by DB2 10 for executing buffer pool prefetch activities. The RMF report shows an IIP service time of 0.513 seconds, which matches the preemptible IIP SRB value shown for the DBM1 address space in Figure 2-37 on page 47. An IIPCP (IIP executed on a CP) value of 0% indicates that sufficient zIIP processor resources were available at the time the RMF interval was taken.

```
REPORT BY: POLICY=WLMPOL                  REPORT CLASS=DB0BMSTR                PERIOD=1
                                  HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS STCHI

 -TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---
 AVG       1.00  ACTUAL                 0  SSCHRT   0.2  IOC         1   CPU    0.518  CP      0.01
 MPL       1.00  EXECUTION              0  RESP     0.1  CPU     14703   SRB    0.002  AAPCP   0.00
 ENDED        0  QUEUED                 0  CONN     0.1  MSO      3784   RCT    0.000  IIPCP   0.00
 END/S     0.00  R/S AFFIN              0  DISC     0.0  SRB        53   IIT    0.000
 #SWAPS       0  INELIGIBLE             0  Q+PEND   0.1  TOT     18541   HST    0.000  AAP     0.00
 EXCTD        0  CONVERSION             0  IOSQ     0.0  /SEC      309   AAP    0.000  IIP     0.86
 AVG ENC   0.00  STD DEV                 0                              IIP    0.513
 REM ENC   0.00                                        ABSRPTN   309
 MS ENC    0.00                                         TRX SERV  309
```

Figure 2-40 RMF workload activity report zIIP usage for buffer pool prefetches

2.12.4 z/OS sort utility (DFSORT)

DFSORT under z/OS 1.12 provides greatly reduced CPU and elapsed time and more consistent performance. PM18196 retrofits this functions to z/OS 1.10.

In z/OS V1R11, DFSORT is modified to allow additional zIIP redirect for DB2 utilities in case of in memory object sort operations of fixed length records. This TCO enhancement is included in the z/OS V1R11 base and is delivered to z/OS V1R10 through APAR PK85856. This feature is included in the DB2 10 base and requires DB2 APAR PK85889 to be installed to function in DB2 for z/OS version 8 and 9. When the additional zIIP redirect takes place, DFSORT issues message ICE256I:

```
ICE256I DFSORT CODE IS ELIGIBLE TO USE ZIIP FOR THIS DB2 UTILITY RUN
```

This TCO enhancement applies to DB2 utility operations that invoke DFSORT for fixed-length record sort processing (for example, index key sort). Be aware that this enhancement does not apply to REORG data sorts because these sorts involve variable-length record sort processing. The sort record type is indicated by the DFSORT runtime message ICE201I:

```
ICE201I G RECORD TYPE IS F - DATA STARTS IN POSITION 1
```

Message ICE201I indicates a fixed-length record type sort was performed which does not cause any additional DFSORT zIIP eligibility.

2.12.5 DRDA workloads

In DB2 10, the zIIP eligibility for DRDA workloads is increased from 55% to 60%. This TCO enhancement is delivered in DB2 for z/OS version 8 and 9 through APAR PM12256.

2.12.6 zAAP on zIIP

z/OS V1R11 introduced the capability to enable zAAP-eligible workloads to run on zIIP speciality engines. This capability was enabled in z/OS V1R9 and V1R10 through APAR OA27495. With zAAP on zIIP support, you can use zIIP processors for both Java and DB2 zIIP eligible workloads provided you have no zAAPs defined. This capability enables you to run zAAP eligible workloads such as Java and XML parsing and XML schema validation on zIIP processors. For example, with the zAAP on zIIP capability workloads, such as z/OS JAVA batch programs and JAVA stored procedures or XML parsing requests, can be redirected to run on a zIIP processor.

Additional information: For more information about the zAAP on zIIP capability introduced by z/OS V1R11, refer to *z/OS Version 1 Release 11 Implementation*, SG24-7729.

2.12.7 z/OS XML system services

DB2 10 pureXML uses the z/OS XML system services for XML schema validation and XML parsing and, therefore, is 100% eligible to be executed on a zIIP or zAAP processor. Figure 2-41 shows the z/OS XML system services processing flow. If the zAAP on zIIP feature is activated zAAP eligible z/OS XML system services workloads are eligible to be processed on a zIIP processor.

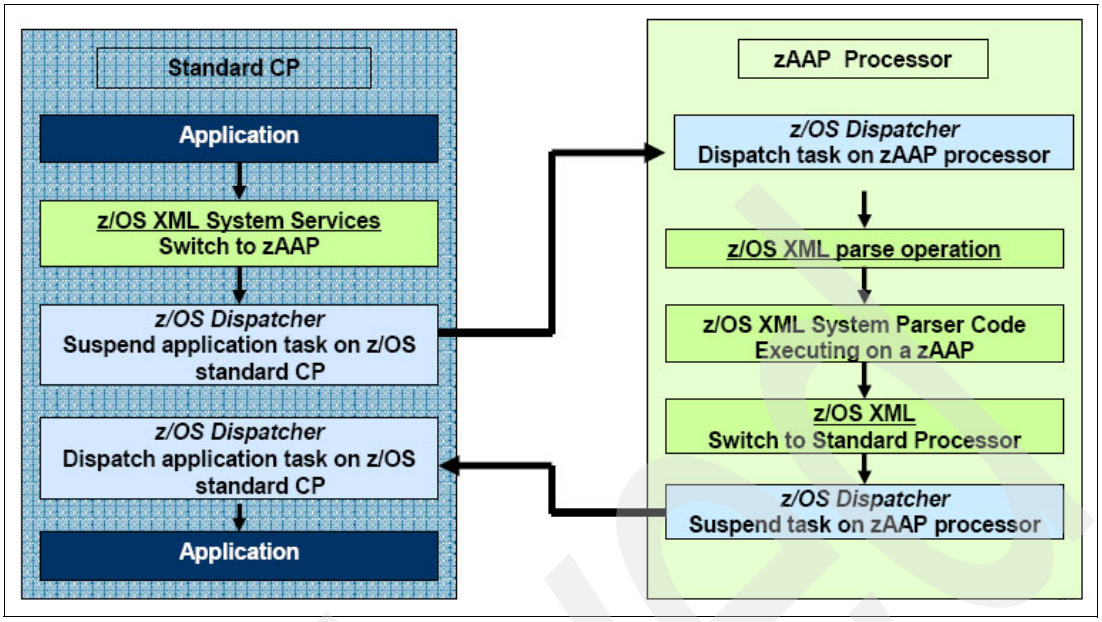


Figure 2-41 z/OS XML system services zIIP and zAAP processing flow

DB2 for z/OS pureXML XML parsing

DB2 pureXML invokes the z/OS XML system services for XML parsing. As a result, the XML parsing request becomes 100% zIIP or zAAP eligible, depending on whether the schema validation request is driven by DRDA through a database access thread (DBAT) or through an allied DB2 thread.

Built in function DSN_XMLVALIDATE

In DB2 10, the SYSIBM.DSN_XMLVALIDATE function is provided inside the DB2 engine as a built-in function (BIF) and uses z/OS XML System Services for XML validation. Thus, DSN_XMLVALIDATE invocations are 100% zIIP or zAAP eligible in DB2 10.

DB2 9 provides XML schema validation through the SYSFUN.DSN_XMLVALIDATE external UDF. The DB2 9 DSN_XMLVALIDATE UDF was executed in task control block (TCB) mode and did not use the z/OS XML system service (XMLSS) for XML validation. Therefore, DSN_XMLVALIDATE invocations were neither zIIP nor zAAP eligible. However, APARs PK90032 and PK90040 have enabled SYSIBM.DSN_XMLVALIDATE in DB2 9.

3

Scalability

Business resiliency is a key component of the value proposition of DB2 for z/OS and the System z platform, supporting your efforts to keep your business running even when things go wrong or you need to make changes. DB2 10 innovations drive new value in resiliency through scalability improvements and fewer outages—planned or unplanned. Scalability delivers the ability to handle up to 20,000 concurrent active users on a single DB2 subsystem, many more than in previous versions.

In this chapter, we discuss the scalability improvements implemented in DB2 10 for z/OS to eliminate or reduce inhibitors to the full exploitation of faster and more powerful hardware. In particular, there are several improvements in the areas of locking and latching, logging, virtual storage constraint relief, and support for large disk volumes. We discuss availability in Chapter 4, “Availability” on page 67.

In this chapter, we discuss the following topics:

- Virtual storage relief
- Reduction in latch contention
- Reduction in catalog contention
- Increased number of packages in SPT01
- The WORKFILE database enhancements
- Elimination of UTSERIAL for DB2 utilities
- Support for Extended Address Volumes
- Shutdown and restart times, and DSMAX
- Compression of SMF records

© Copyright IBM Corp. 2010. All rights reserved.

3.1 Virtual storage relief

Prior to DB2 10, the primary constraint to vertical scalability was virtual storage below the 2 GB bar. DB2 V8 introduced 64-bit support for DB2 control blocks and data areas. DB2 9 for z/OS provided an additional10-15% relief for agent thread usage. The DB2 10 for z/OS target is that up to 90% of the DBM1 address space and the distributed data facility (DDF) address space will be running above the bar. The result is the ability to run much more concurrent work in a single subsystem and to run from five to ten times more concurrent threads. For instance, if your configuration and application mix can support 500 concurrent threads on a single subsystem with DB2 9, you can support as many as 5000 concurrent threads on a single subsystem with DB2 10.

Each LPAR has a cost in CPU, memory, and disk. In addition, users spend time monitoring and tuning virtual storage to have a memory cushion. Then, when memory limits are reached, extensive tuning and changes to the subsystem and databases are needed. When users need more than one DB2 subsystem on an LPAR, this implies extra cost in real storage and CPU time.

With the change in virtual storage in DB2 10, more work can run in one DB2 subsystem, allowing a consolidation of LPARS as well as DB2 members, and storage monitoring is also reduced.

The net result for this virtual storage constraint release change is reduced cost, improved productivity, easier management, and the ability to scale DB2 much more easily.

DB2 10 also includes increased limits for the CTHREAD, MAXDBAT, IDFORE, IDBACK, MAXOFILR threads. Specifically, the improvement allows a 10 times increase in the number of these threads (meaning 10 times the current supported value at your installation, not necessarily 10 times 2000). So, for example, if in your installation you can support 300-400 concurrently active threads based on your workload, you might now be able to support 3000-4000 concurrently active threads.

3.1.1 Support for full 64-bit run time

To get the benefit of thread storage reduction, you need to REBIND your applications. After you REBIND, the work areas are moved above the bar. You can bring the package above the bar for virtual storage relief.

In DB2 10 many control blocks saved in the runtime structure of a bound SQL statement are converted to use 64-bit addressability. This conversion enables these packages to be brought above the 2 GB bar for virtual storage relief.

Static queries that are bound with query parallelism in releases prior to DB2 10 go through incremental rebind to pick up DB2 10 bound structures before execution. You should rebind these types of queries in DB2 10 conversion mode. After migrating to DB2 10 conversion mode, you can monitor incremental rebinds on the DB2 accounting and statistic reports and issue explicit rebind for packages reported on IFCID 360 to avoid the cost of incremental rebinds at execution time.

To help you identify which packages are affected by additional authorization checking, the DB2 10 DSNTIJPM premigration job includes the query shown in Example 3-1. This same change was made to the DSNTIJPA job, which is the DB2 10 premigration job that shipped in DB2 V8 and DB2 9.

Example 3-1 Queries with parallelism enabled

```
SELECT COLLID || '.' || NAME, HEX(CONTOKEN)
FROM SYSIBM.SYSPACKAGE
WHERE DEGREE='ANY';
```

The package MYCOLLID.MYPACKAGE that we list here is currently bound with parallelism, and the queries bound with parallel mode are incrementally rebound each time they are executed on DB2 10.

```
Collection ID and package name                                CONTOKEN
--------------------------------------------- ... --- ----------------
MYCOLLID.MYPACKAGE <-- 257 bytes -->                    7C19120F67AB7879
```

3.1.2 64-bit support for the z/OS ODBC driver

As many of the 31-bit ODBC applications have outgrown their memory needs, products such as WebSphere Message Broker and PeopleSoft have begun their transition to the 64-bit operating environment. The move to 64 bits provides relief to virtual storage constraint by allowing for the allocation of data stores above the 2 GB bar in the application address space. Many ODBC applications that work with large volumes of data have positioned themselves for 64-bit addressing to take advantage of larger addressable storage.

However, the initial implementation of the DB2 for z/OS ODBC driver runs only in 31-bit environments and does not work with 64-bit applications. The lack of a 64-bit implementation of the DB2 for z/OS ODBC driver prevented many ODBC applications from making the transition to 64-bit.

The 64-bit ODBC driver is a driver for DB2 9, made available through APARs PK83072 and PK99362 (respectively PTF UK50918 and UK52302). It is shipped in addition to the 31-bit ODBC drivers (non-XPLINK and XPLINK) that are supported by DB2. The driver consists of two 64-bit DLLs, DSNAO64C and DSNAO64T, and a definition side-deck that contains functions exported by DSNAO64C. The DLLs reside in the db2prefix.SDSNLOD2 data set and the side-deck resides in the db2prefix.SDSNMACS data set as member DSNAO64C.

To prepare your ODBC application to run on the 64-bit driver, you must first compile your application with the LP64 compiler option and link your application with the new side-deck using the DFSMS binder. You also need to include the new DLLs in the JOBLIB DD statement for the job or in the STEPLIB DD statement for the job step that invokes the application.

DB2 10 provides a 64-bit AMODE(64) version of the ODBC driver to the z/OS platform. The ODBC driver is completely recompiled and rebuilt with the LP64 compiler option. For APIs that currently accept pointer arguments, the pointer values are expanded to 64 bits, allowing for addressing up to 16 exabytes. The 64-bit architecture also provides ODBC the opportunity to move some of its control block structures above the 2 GB bar to alleviate virtual storage constraints.

Sending and receiving data: Although 64-bit mode provides larger addressable storage, the amount of data that can be sent to or retrieved from DB2 by an ODBC application is limited by the amount of storage that is available below the 2 GB bar. Therefore, for example, an application cannot declare a 2 GB LOB above the bar and insert the entire LOB value into a DB2 LOB column.

The following functions are not supported in the 64-bit driver:

- SQLSetConnectOption (SQLHDBC hdbc, SQLUSMALLINT fOption, SQLUINTEGER vParam)
- SQLSetStmtOption (SQLHSTMT hstmt, SQLUSMALLINT fOption, SQLUINTEGER vParam)

These functions have an integer parameter, vParam, that can be cast to a pointer, which is not possible on a 64-bit system. You need to modify applications to use SQLSetConnectAttr() and SQLSetStmtAttr() before you can use the vPARAM in a 64-bit environment. The 64-bit driver returns SQLSTATE HYC00/S1C00 (Driver not capable) for these two functions.

3.2 Reduction in latch contention

A latch is a DB2 mechanism for controlling concurrent events or the use of system resources. Latches are conceptually similar to locks in that they control serialization. They can improve concurrency because they are usually held for a shorter duration than locks and they cannot “deadlatch”. However, latches can wait, and this wait time is reported in accounting trace class 3 data.

DB2 reports lock and latch suspension times in IFCID 0056 and IFCID 0057 pairs. This can be reported on by IBM Tivoli OMEGAMON XE for DB2 Performance Expert Version (OMEGAMON PE) V5R1 in its accounting trace.

Most of DB2 latches which could impact scalability have an improvement with DB2 10 CM:

- LC12: Global Transaction ID serialization
- LC14: Buffer Manager serialization
- LC19: Log write in both data sharing and non data sharing (CM and also NFM)
- LC24: EDM thread storage serialization (Latch 24)
- LC24: Buffer Manager serialization (Latch 56)
- LC27: WLM serialization latch for stored procedures and UDF
- LC32: Storage Manager serialization
- IRLM: IRLM hash contention
- CML: z/OS Cross Memory Local suspend lock
- UTSERIAL: Utility serialization lock for SYSLGRNG (removed in NFM)

Latch class 19 contention can limit workload scalability in environments where there is a large number of log records created in the same time interval.

Since DB2 Version 1, a single latch was used for the entire DB2 subsystem to serialize updates to the log buffers when a request is received to create a log record. Basically, the latch is obtained, an RBA range is allocated, the log record is moved into the log buffer, and then the latch is released. This method simplifies processing but also creates a bottleneck when CPUs get faster and the number of CPUs on a system increases. Recent tests in IBM labs were able to hit several hundred thousand log latch contentions per second on a 5-way z10 system with DB2 9.

With DB2 10, the log latch is held for the minimum time necessary to allocate space for the log record. The movement of the log record and updating of the control structures is done after the latch is released, allowing multiple log records to be moved in parallel.

Note: With the possible Increase in logging rate, you need to ensure that products that read log records can handle increased logging rates. For products that use the IFCID 306 interface (such as DB2 DataPropagator for z/OS), you might need to use larger buffers to keep up with increased logging or change from using merged member reads to reading from each data sharing member, or both.

You specify the log output buffers in the OUTPUT BUFFER field of the installation panel DSNTIPL. The minimum value is increased from 40 KB to 400 KB, and the default is changed from 400 KB to 4 MB. These buffers are permanently page fixed to reduce CPU time. Specifying more than 4 MB will use more real storage and generally will not help DB2 write the log buffers faster. However, more buffers might help with high spikes of workload dependent write activity and the performance of undo processing.

Refer to 3.8, “Shutdown and restart times, and DSMAX” on page 64 for information regarding the page fixing of OUTBUFF.

3.3 Reduction in catalog contention

DB2 10 reduces catalog contention by eliminating catalog links and converting the catalog tables to use row level locking instead of page-level locking. This change, made available in new-function mode, reduces catalog contention dramatically during such events as concurrent DDL processing and BIND operations.

The following SQL statements that update the catalog table spaces contend with each other when those statements are on the same table space:

- CREATE TABLESPACE, TABLE, and INDEX
- ALTER TABLESPACE, TABLE, INDEX
- DROP TABLESPACE, TABLE, and INDEX
- CREATE VIEW, SYNONYM, and ALIAS
- DROP VIEW and SYNONYM, and ALIAS
- COMMENT ON and LABEL ON
- GRANT and REVOKE of table privileges
- RENAME TABLE
- RENAME INDEX
- ALTER VIEW

Although BIND and REBIND might take longer to process, they cause only minimal contention.

3.4 Increased number of packages in SPT01

In today’s very large DB2 installations, you can have tens of thousands or even hundreds of thousands of packages stored in SPT01, especially if you are running large application software systems or have development systems with many versions of application packages or old packages that might no longer be needed. Enhancements such as plan stability and plan versioning have put further pressure on the size of SPT01.

Users have come close to reaching the 64 GB size limit for SPT01. To prevent you from reaching this limit, DB2 9 provides the capability to turn on compression for SPT01 (APAR PK80375[1]), which allows for increased headroom in SPT01.

DB2 10 restructures SPT01 to allow storing packages in LOBs. SPT01 is split into several pieces with the larger sections of each package stored in two LOBs. This greatly expands the number of packages of SPT01. However, it makes compression ineffective on packages because LOBs cannot be compressed. DB2 10 also moves SPT01 to the 32 KB buffer pool to further position towards additional improvements.

Do not be too concerned with resizing your 32 KB buffer pool to support the move of SPT01 there, the buffer pool is only temporarily used to move package information into the EDMPOOL as packages are used, there should not be a big demand for these buffers. However, if you plan on doing massive amounts of BINDs or REBINDs, you need to consider increasing the size of the 32 KB buffer pool assigned to SPT01. This increase can be a temporary change, and after you have completed the BINDs or REBINDs, you can decrease the size of this 32 KB buffer pool.

3.5 The WORKFILE database enhancements

In this section, we discuss the following work file management changes:

- Support for spanned work file records
- In-memory work file enhancements for performance
- The CREATE TABLESPACE statement
- Installation changes

3.5.1 Support for spanned work file records

DB2 10 allows work file records to be spanned, which provides the functionality to allow the work file record length to be up to 65529 bytes by allowing the record to span multiple pages. This support alleviates the issue of applications receiving SQLCODE -670 (SQLSTATE 54010) if the row length in the result of a join or the row length of a large sort record exceeds the 32 KB maximum page size of a work file table space.

The spanned work file records support allows the records of the work files created for JOINs and large sorts to span multiple pages to accommodate larger record length and larger sort key length for sort records.

When the row length in the result of a JOIN or the row length of a large sort record exceeds approximately one page of work file space, the work file manager allows the work file record to span multiple pages, provided that the work file record length is within the new limit of 65529 bytes.

The *maximum* limit for sort key length for sort is increased from 16000 to 32000 bytes. This limit alleviates the issue of applications receiving SQLCODE -136 (SQLSTATE 54005) if the length of the sort key derived from GROUP BY, ORDER BY, DISTINCT specifications in the SQL statement exceeds the limit of 16000 bytes.

The (SQLCODE -670,SQLSTATE 54010) or (SQLCODE -136,SQLSTATE 54005) is issued when the row length in the result of a JOIN or the row length of a large sort record exceeds the limit of 65529 bytes or when the sort key length for a sort exceeds the limit of 32000 bytes, respectively.

Spanned records support: This support is available in DB2 10 new-function mode.

[1] This DB2 9 APAR adds an opaque DB2 subsystem parameter called COMPRESS_SPT01 to DSN6SPRM that can be used to indicate whether the SPT01 directory space should be compressed. Not applicable to DB2 10.

3.5.2 In-memory work file enhancements for performance

The in-memory work file enhancements are intended to provide performance improvements to help workloads with queries that require the use of small work files to consume less CPU time. These enhancements facilitate wider usage of the in-memory work files in the work file database by allowing simple predicate evaluation for work files. This support is intended to reduce the CPU time consumed by workloads that execute queries which require the use of small work files.

In-memory work file support: This support is available in DB2 10 conversion mode.

In DB2 9, the Real Time Statistics table SYSIBM.SYSTABLESTATS maintains the disk storage allocated information for each table space in the work file database by means of one row per table space. In DB2 10 for partition-by-growth table spaces in the work file database, there is one row for each partition of the table space.

If you have programs to monitor total disk storage that is used for the work file database, you might need to change the programs to adapt to the partition-by-growth statistics. The DB2 10 data collected from the DB2 accounting trace class (1,2) and examination of the fields Accounting class 2 elapsed time and Accounting class 2 CPU time can be used for comparison with corresponding baseline data from prior releases.

3.5.3 The CREATE TABLESPACE statement

Table spaces in the work file database are allowed to be partition-by-growth table spaces.

This option can alleviates the issue of applications that use DGTTs receiving SQLCODE -904 for lack of available space in the work file database that is limited to a single table space. size of 64 GB.

A second value is to provide a more effective means of preventing “runaway” DGTTs. You can now specify a large SECQTY (which is good for performance) and limit the value of either MAXPARTITIONS or DSSIZE.

Partition-by-growth table spaces in WORKFILE database: This support is available in DB2 10 new-function mode.

When a table space in the work file database grows in the number of partitions, the new partitions remain. The data sets for the new partitions are not deleted when DB2 is restarted.

The work file enhancements require changes in the CREATE TABLESPACE statement. They do not affect DB2 commands and utilities.

Note the following changes for the CREATE TABLESPACE statement:

- The following DB2 9 restriction is removed:

  MAXPARTITIONS should not be specified for a table space that is in a work file database.
- If MAXPARTITIONS is specified for a table space in a work file database, the table space is a partition-by-growth table space.

  If the DSSIZE or NUMPARTS clauses are not specified, the default values for DSSIZE and NUMPARTS are used.

- If MAXPARTITIONS is not specified for a table space in a work file database, the table space is a segmented table space; therefore, DSSIZE and NUMPARTS cannot be specified.

3.5.4 Installation changes

Figure 3-1 shows the updated installation panel DSNTIP9.

```
DSNTIP9     INSTALL DB2 - WORK FILE DATABASE
===>
Enter work file configuration options below:

 1 TEMP 4K SPACE          ===> 20      Amount of 4K-page work space (MB)
 2 TEMP 4K TBL SPACES     ===> 1       Number of table spaces for 4K-page data
 3 TEMP 4K SEG SIZE       ===> 16      Segment size of 4K-page table spaces
 4 TEMP 32K SPACE         ===> 20      Amount of 32K-page work space (MB)
 5 TEMP 32K TBL SPACES    ===> 1       Number of table spaces for 32K-page data
 6 TEMP 32K SEG SIZE      ===> 16      Segment size of 32K-page table spaces
 7 MAX TEMP STG/AGENT     ===> 0       Maximum MB of temp storage space
                                       that can be used by a single agent
 8 SEPARATE WORK FILES    ===> NO      Unconditionally separate DGTT work and
                                       workfile work in different work file TSs
                                       based on their allocation attributes
 9 MAX TEMP RIDS          ===> NOLIMIT Max RID blocks of temp storage space
                                       that can be used by a single RID list
```

Figure 3-1 Updated installation panel DSNTIP9

From this panel you can select the following options:

- TEMP 4K SPACE

  Specifies the total amount of space for all 4 KB table spaces in the work file database. This value is specified in MB. The following values are acceptable values:

  – For installation 1 to 8,388,608,000 (1 to 32,702,464 in DB2 9)
  – For migration 0 to 32,702,464 (same as in DB2 9)

  The default value for installation is 20 and for migration is 0 (same as in DB2 9).

- TEMP 4K TBL SPACES

  Specifies the number of 4 KB work file table spaces that are to be created. The following values are acceptable values:

  – For installation 1 to 500
  – For migration 0 to 500 (same as in DB2 9)

  The default value for installation is 1 and for migration is 0 (same as in DB2 9).

- TEMP 4K SEGSIZE

  Specifies the segment size for 4 KB work file table spaces, as a multiple of 4, from 4 to 64. In migration mode from Version 8 only, the value of TEMP 4K SEGSIZE is always set to 16 and cannot be changed. The value of TEMP 4K SEGSIZE does not apply to existing 4 KB work file table spaces.

- TEMP 32K SPACE

  Specifies the total amount of space for all 32 KB table spaces in the work file database. This value is specified in MB. The following values are acceptable values:

  – For installation 1 to 67,108,864,000 (1 to 32,702,464 in DB2 9)
    For migration 0 to 32,702,464 (same as in DB2 9)

  The default value for installation is 20 and for migration is 0 (same as in DB2 9).

- TEMP 32K TBL SPACES

  Specifies the number of 32 KB work file table spaces that are to be created. The following values are acceptable values:

  – For installation 1 to 500
  – For migration 0 to 500 (same as in DB2 9)

  The default value for installation is 1 and for migration is 0 (same as in DB2 9).

- TEMP 32K SEGSIZE

  Specifies the segment size for 32 KB work file table spaces, as a multiple of 4, from 4 to 64.

  In migration mode from Version 8 only, the value of TEMP 32K SEGSIZE is always set to 16 and cannot be changed. The value of TEMP 32K SEGSIZE does not apply to existing 32 KB work file table spaces.

- SEPARATE WORKFILES

  This field corresponding to a new DSNZPARM called WFDBSEP (in DSN6SPRM) specifies whether DB2 should provide an unconditional separation of table spaces in a work file database based on their allocation attributes. Acceptable values are NO (default) and YES.

  If you specify YES, DB2 always directs DGTT work only to DB2-managed (STOGROUP) work file table spaces defined with a non-zero SECQTY and work file work only to other work file table spaces (DB2-managed table spaces defined with a zero SECQTY or user-managed table spaces). If no table space with the preferred allocation type is available, DB2 issues an error (message DSNT501I or SQLCODE -904).

  If you use the default NO, DB2 attempts to direct DGTT work to DB2-managed (STOGROUP) work file table spaces defined with a non-zero SECQTY and work file work to any other work file table space (DB2-managed table spaces defined with a zero SECQTY or user-managed table spaces). If no table space with the preferred allocation type is available, DB2 selects a table space with a non-preferred allocation type.

- MAX TEMP RIDS

  This field corresponding to new DSNZPARM called **MAXTEMPS_RID** (in DSN6SPRM) determines the maximum amount of temporary storage in the work file database that a single RID list can use at a time. The work file storage is used for the RID list when the RID pool storage cannot be used to contain all the RIDs. In other words, when RID pool storage overflow occurs for a RID list, DB2 attempts to store the RID list in work file storage instead of falling back to a relational scan. This parameter specifies the maximum number of RIDs (measured in RID blocks) that DB2 is allowed to store in the work file. If the maximum number of RIDs is exceeded, DB2 falls back to a relational scan.

  The DSNZPARM MAXTEMPS_RID does not affect RID list processing for pair-wise join.

  Acceptable values are NONE, NOLIMIT, or 1 to 329166 and the default is NOLIMIT.

Archived

Changes to space calculations

In migration mode, the installation CLIST space calculations for work file table spaces are unchanged from those used in DB2 9 because work file table spaces cannot be created as partition-by-growth in DB2 10 conversion mode.

In (new) installation mode, the installation CLIST performs the following calculations to determine either the PRIQTY or DSSIZE and MAXPARTITION settings:

- For 4 KB page size table spaces

  Given the following data:

  ```
  X = total MBs of space (from field 1 TEMP 4K SPACE)
  Y = number of TSs (from field 2 TEMP 4K TBL SPACES)
  Z = CEILING(X/Y/1024) = GB of space per work file table space
  ```

  Then:

  ```
  If Z < 1G then PRIQTY = CEILING(X/Y)
  Else if Z <= 16,384G then DSSIZE=4G, MAXPARTITIONS=Z/4
  Else DSSIZE=4G, MAXPARTITIONS=4096
  ```

- For 32 KB page size table spaces

  Given the following data:

  ```
  X = total MBs of space (from field 4 TEMP 32K SPACE)
  Y = number of TSs (from field 5 TEMP 32K TBL SPACES)
  Z = CEILING(X/Y/1024) = GB of space per work file table space
  ```

  Then:

  ```
  If Z < 1G then PRIQTY = CEILING(X/Y)
  Else if Z <= 16,384G then DSSIZE=4G, MAXPARTITIONS=Z/4
  Else if Z <= 131,072G then DSSIZE=32G, MAXPARTITIONS=Z/32
  Else DSSIZE=32G, MAXPARTITIONS=4096
  ```

Changes to the work file generator exec

The parameters on the DSNTIP9 panel are used to generate the last step in job DSNTIJTM. This step executes a tool (DSNTWFG DB2 REXX exec) for creating the specified number of 4 KB or 32 KB work file table spaces. Note that this is a REXX program and requires DB2 REXX Language support.

Example 3-2 shows a sample of this step and the description of its input parameters.

Example 3-2 DSNTWFG and description of its parameters

```
//*
//* DSNTWFG: ADD 4-KB AND 32-KB TABLE SPACES IN THE WORK FILE DB
//* PARMS: 1. LOCAL DB2 SSID
//*        2. AUTHORIZATION ID TO PERFORM THE WORK
//*        3. NAME OF THE DB2 WORK FILE DATABASE
//*        4. NUMBER OF 4-KB TYPE TABLE SPACES TO BE ADDED
//*        5. MAXIMUM NUMBER OF PARTITIONS (MAXPARTITIONS) IN EACH
//*           4-KB TYPE TABLE SPACE
//*        6. IF THE PRECEDING PARAMETER IS NON-ZERO, THIS PARAMETER SPECIFIES
//*           THE MAXIMUM SIZE IN GB (DSSIZE) FOR EACH PARTITION IN EACH
//*           4-KB TYPE TABLE SPACE
//*        7. SEGMENT TYPE FOR 4-KB TYPE TABLE SPACES
//*        8. NAME OF BUFFERPOOL FOR 4-KB TYPE TABLE SPACES
//*        9. NAME OF DB2 STORAGE GROUP FOR 4-KB TYPE TABLE SPACES
```

```
//*       10. NUMBER OF 32-KB TYPE TABLE SPACES TO BE ADDED
//*       11. MAXIMUM NUMBER OF PARTITIONS (MAXPARTITIONS) IN EACH
//*           32-KB TYPE TABLE SPACE
//*       12. IF THE PRECEDING PARAMETER IS NON-ZERO, THIS PARAMETER SPECIFIES
//*           THE MAXIMUM SIZE IN GB (DSSIZE) FOR EACH PARTITION IN EACH
//*           32-KB TYPE TABLE SPACE
//*       13. SEGMENT TYPE FOR 32-KB TYPE TABLE SPACES
//*       14. NAME OF BUFFERPOOL FOR 32-KB TYPE TABLE SPACES
//*       15. NAME OF DB2 STORAGE GROUP FOR 32-KB TYPE TABLE SPACES
//*
//DSNTIST EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSEXEC DD DISP=SHR,DSN=DSN!!0.SDSNCLST
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSNTWFG DSN SYSADM DSNDB07 +
1 16 0 16 BP0   SYSDEFLT +
1  1 0 16 BP32K SYSDEFLT
=
```

The following parameters are added to the DSNTWFG parameter list or modified:

- Parameter 5 is added to pass the maximum number of partitions in each 4 KB type table space. If non-zero, each such table space created is partitioned-by-growth. If zero, each such table space created is segmented.
- Parameter 6 is modified to pass either of the following values:
  - The data set size when 4 KB type work file table spaces are created as partition-by-growth.
  - The primary space quantity when 4 KB type work file table spaces are created as segmented (as in DB2 9).
- Parameter 11 is added to give the maximum number of partitions in each 32 KB type table space. If non-zero, each table space created is partition-by-growth. If zero, each table space that is created is segmented.
- Parameter 12 is modified to pass either of the following values:
  - The data set size when 32 KB type work file table spaces are created as partition-by-growth.
  - The primary space quantity when 32 KB type work file table spaces are created as segmented (as in DB2 9).

3.6 Elimination of UTSERIAL for DB2 utilities

Prior to DB2 10, access to the directory table space SYSUTILX was controlled by a unique lock called the *UTSERIAL* lock. Utilities acquired the UTSERIAL lock to read or write in SYSUTIL or SYSUTILX tables. Utilities acquired this lock during utility initialization to check for compatibility with other utilities already executing. Additionally, this lock was taken at the start of each utility phase to update the utility's status. Commands such as TERM UTIL also took this lock. During heavy utility processing, it was possible for utilities to wait on this lock and occasionally time out.

As systems have become faster and users have moved to more automated forms of running utilities, it is now more common for tens of utility jobs to be running concurrently against hundreds of objects. Thus, using the UTSERIAL lock might lead to contention and time out issues.

DB2 10 NFM eliminates the UTSERIAL lock and merges tables SYSUTILX and SYSUTIL into one table called SYSUTILX and takes a more granular page level lock against this table. This locking strategy eliminates the majority of the contention and time out issues. With DB2 10, you can scale up and run many more concurrent utilities with minimal contention on SYSUTILX.

3.7 Support for Extended Address Volumes

Extended Address Volume (EAV) support in z/OS 1.10 brought a major breakthrough in disk storage size. For over four decades, disk tracks were addressed as *CCCCHHHH*, where *CCCC* represented a 16-bit cylinder number and *HHHH* was a 16-bit head number. The maximum possible size per volume was limited to a little less than 64K cylinders or 65,520 cylinders and roughly 54 GB in the 3390 mod 54 disk.

EAV introduces a track addressing format *CCCCcccH*, where *CCCCccc* allows for a 28-bit cylinder number. Thus, the disk storage can theoretically grow up to 268,434,453 cylinders per volume or about 221 TB.

An *EAV volume* is defined as any volume that has more than 65,520 cylinders. EAV hardware support is provided by the TotalStorage® DS8000 DASD subsystem. The EAV volume format maintains the 3390 track size, track image, and tracks per cylinder. An EAV volume is logically partitioned into the following regions. A system parameter, the Breakpoint Value (default 10 cylinders), determines in which region the data set or extent is to reside.

- Base addressing space

  Also known as *track-managed space*, the base addressing space is the area on an EAV volume list located within the first 65,520 cylinders. Space in track-managed space is allocated in track or cylinder increments (same as today).

- Extended addressing space (EAS)

  Also known as *cylinder-managed space*, the EAS is the area on an EAV volume located above the first 65,520 cylinders. Space in cylinder-managed space is allocated in multi-cylinder units. Currently, on an EAV volume, a multi-cylinder unit is 21 cylinders

EAV support has been be staged over several z/OS releases. z/OS 1.10 provided the initial implementation of EAV and support for VSAM data sets in EAS. The maximum supported volume size is 256 K cylinders or 223 GB. z/OS 1.11 added support of extended format (EF) sequential data sets. Basic and large sequential data sets, PDS, PDSE, BDAM, and catalogs are supported in EAV in z/OS 1.12.

For more information, see the announcement letter at:

`http://www.ibm.com/common/ssi/rep_ca/8/897/ENUS210-008/ENUS210-008.PDF`

We also discuss this topic in 2.4.1, “Extended address volumes” on page 13 as a Synergy with System z topic.

With z/OS 1.11, EAV introduces a new data set level attribute, extended attribute (EATTR), to control whether the data set can reside in EAS. EATTR can be specified using the DD statement, the IDCAMS DEFINE CLUSTER command, and SMS data class.

Support for the EAV enhancements for all VSAM data sets provided by z/OS 1.10, z/OS 1.11, and z/OS 1.12 are implemented in DB2 V8 and DB2 9 through the following APARs:

- PK58291 adds support to allow DB2-supplied administrative enablement stored procedures to access EAV.
- PK58292 adds DB2 log manager support for EAV, allows DB2 bootstrap data sets (BSDS) and active logs to reside anywhere on an EAV.
- PK81151 provides additional EAV support for DB2 utilities.

DB2 data sets can take more space or grow faster on EAV versus non-EAV volumes. The reason is that the allocation unit in the EAS of EAV volumes is multiple of 21 cylinders and every allocation is rounded up to this multiple. Your data set space estimation must take this into account if EAV is be used. This effect is more pronounced for smaller data sets.

With z/OS 11 EAV support, you can use the new EATTR attribute provided by z/OS to control data set allocation in EAS. For example, the EATTR setting in an SMS data class will allow or disallow DB2 page set allocation in EAS. For a user defined data set, you can specify EATTR in a DEFINE CLUSTER command. Where applicable, EATTR can be specified along with other data set attributes in JCL DD statements. For DB2 managed data sets, DB2 does not add the EATTR attribute when issuing IDCAMS DEFINE CLUSTER commands. EATTR is applicable only through SMS data class.

z/OS EAV supports SMS and non-SMS VSAM data sets. So a DB2 V8 user can have EAV volumes for catalog and directory which are either SMS or non-SMS managed.

Table 3-1 shows the data sets that are eligible to be placed in EAS. In DB2 10, the catalog and directory are SMS managed and require an SMS data class with the EA attribute (and therefore EAV enabled). The DB2 10 catalog and directory can be placed in EAS. For more information, refer to 12.9, “SMS-managed DB2 catalog” on page 516.

Table 3-1 EAV enablement and EAS eligibility

DB2 objects	z/OS 1.10	z/OS 1.11	z/OS 1.12
Tables and Indexes	Yes	Yes	Yes
BSDS	Yes	Yes	Yes
Active logs	Yes	Yes	Yes
Archive logs	No	Yes, if EF sequential	Yes
Utilities sequential input and output data sets	No	Yes, if EF sequential	Yes,
Utilities partitioned data sets and PDSEs	No	No	Yes
Sort work data sets	No	No	Yes if DFSORT used by utilities
DB2 installation data sets (CLISTs, panels, samples, macros, etc.)	NO	No	Yes
SDSNLINK, SDSNLOAD	No	No	Yes

3.8 Shutdown and restart times, and DSMAX

z/OS 1.12 provides a substantial reduction in allocation and deallocation of data sets in addition to improvements in the time required for some phases of z/OS initialization processing.

Any time there is a large number of allocations or deallocations, the performance of DB2 10 is improved.

DB2 10 systems that use a large number of data sets (DSMAX of 100,000) will see improvements in shutdown and restart times.

During normal operation, if a table space with a large number of partitions, needs to have all partitions to be opened by DB2 at about the same time, the performance will benefit from this z/OS R12 function.

The same benefits are available to DB2 V8 and DB2 9, if you are using z/OS 1.12 or later, with APARs PM00068, PM17542, and PM18557. In this case you should take one of the following actions to improve the performance of opening and closing data sets:

- Update the ALLOCxx parmlib member to set the SYSTEM MEMDSENQMGMT value to ENABLE

  Updating the ALLOCxx parmlib is strongly recommended because the change remains effective across IPLs.

- Issue system command SETALLOC SYSTEM,MEMDSENQMGMT=ENABLE

  If the SETALLOC command is used to enable SYSTEM MEMDSENQMGMT, a DB2 restart is required to make the change effective.

3.9 Compression of SMF records

Because SMF record volume from DB2 can be very large, there can be significant savings in compressing DB2 SMF records. Compression can provide increased throughput as the records written are smaller and savings of auxiliary storage space to house these files. Laboratory measurements show that SMF compression generally saves 60% to 80% of the space for DB2 SMF records and requires less than 1% in CPU for overhead to do it.

The DSNZPARM SMFCOMP, new with DB2 10, activates compression of DB2 trace records destined for SMF using z/OS compression service CSRCESRV[2]. The parameter is specified in DSNTIPN panel field COMPRESS SMF RECS and the default is OFF.

If this parameter is enabled, any data after the SMF header for the SMF100, 101, and 102 records is compressed. In a data sharing environment, the SMFCOMP parameter has member scope.

OMEGAMON PE V5.1 handles the compressed SMF records automatically.

APAR PM27872 (PTF UK64597) adds routine DSNTSMFD to DB2 10. DSNTSMFD is a DB2-supplied sample application which accepts an SMF data set and produces an output SMF data set. All SMF record types are accepted, but only compressed type 100 (DB2 statistics), type 101 (DB2 accounting), and type 102 (DB2 performance) records generated by

[2] The service uses an algorithm called run-length encoding (RLE) which is a compression technique that compresses repeating characters, RLE is a simple widely used lossless compression technique with minimal impact.

DB2 are decompressed. All other records (including uncompressed DB2 records) are copied unchanged to the output SMF data set.

Another item which can reduce the amount of DB2 SMF tracing is the improved rollup for accounting, as noted in 13.23.6, “Package accounting information with rollup” on page 601.

Archived

Archived

4

Availability

DB2 10 for z/OS continues to bring changes that improve availability, keeping up with the explosive demands of transaction processing and business intelligence that require on-demand actions and changes without interruption of service. DB2 10 delivers increased application and subsystem availability with more functions for schema evolution, logging enhancements, and online utilities.

In this chapter, we discuss the following topics:

- Online schema enhancements
- Autonomic checkpoint
- Dynamically adding an active log data set
- Preemptible backout
- Support for rotating partitions
- Compress on insert
- Long-running reader warning message
- Online REORG enhancements
- Increased availability for CHECK utilities

In other chapters in this book, we discuss the following DB2 10 functions that also contribute to availability:

- DB2 10 effectively eliminates the risks that are associated with virtual storage constraint, which can cause interruption of service. See 3.1, “Virtual storage relief” on page 52.
- The DB2 10 catalog restructuring allows more concurrency among SQL executions that require update actions on the catalog. See 12.6, “Implications of DB2 catalog restructure” on page 500.
- The DB2 10 BIND option allows applications to retrieve data without incurring infrequent timeout. See 7.5, “Access to currently committed data” on page 238.

© Copyright IBM Corp. 2010. All rights reserved.

4.1 Online schema enhancements

Prior to DB2 V8, a change to a database definition almost always required you to unload the data, drop the object, re-create the object, reestablish authorization on the object, re-create the views, and then reload the data. DB2 V8 introduced ALTER statements to change database object attributes online. DB2 9 added more possible changes. DB2 10 includes a large number of additional ALTERs that allow you to convert the structure of the table space and change attributes.

Key point: All the DB2 10 ALTER options apply to universal table spaces (UTS).

You might want to convert the structure from simple table spaces, which are deprecated since DB2 9, or you might want to convert the segmented and partitioned table spaces to the UTS structure to exploit its characteristics.

4.1.1 UTS with DB2 9: Background information

Starting with DB2 9 for z/OS new-function mode (NFM), you can combine the benefits of segmented space management with partitioned table space organization using UTS. The combined advantages are as follows:

- A segmented space-map page has more information about free space than a partitioned space-map page.
- Mass delete performance is improved because mass delete in a segmented table space organization tends to be faster than in other types of table space organizations.
- All or most of the segments of a table are ready for immediate reuse after the table is dropped or mass deleted.
- Partitioning allows for support of large table spaces and parallelism of accesses.

There are two types of UTS:

- *Partition-by-growth* table spaces (also known as *PBG* table spaces) are like single-table DB2-managed segmented table spaces. DB2 manages partition-by-growth table spaces automatically. There is no partitioning value binding data to partitions. The table space begins as a single-partition table space and grows automatically, as needed, as more partitions are added to accommodate data growth. DB2 adds a new physical partition automatically, not related to value ranges but to size, when it needs more space to satisfy an insert. Compression dictionaries are copied automatically across the new partitions. A partition-by-growth table space can grow up to 128 TB, and its maximum size is determined by MAXPARTITIONS, DSSIZE, and page size. This structure is useful to deal with table spaces that increase the overall size on demand.
- *Range-partitioned* table spaces (also know as *partition-by-range* or *PBR* table spaces) are based on partitioning value ranges. UTS range-partitioned table spaces requires table controlled partitioning, index partitioning is not supported. They are similar to the traditional data partitioned table spaces with the addition of segmented characteristics. The maximum size of a range-partitioned table space is 128 TB. A range-partitioned table spaces is segmented. However, it contains a single table, which makes it similar to the regular partitioned table space. Range-partitioned table spaces are defined by specifying the SEGSIZE and NUMPARTS keywords on a CREATE TABLESPACE statement. After the table space is created, activities that are already allowed on partitioned or segmented table spaces are allowed.

Archived

All table spaces, including UTS, are created in reordered row format by default, unless the DSNZPARM SPRMRRF is set to DISABLE.

UTS are identified by values in columns of the DB2 catalog table SYSTABLESPACE.

The TYPE column includes one of the following values:

R	Range-partitioned UTS
P	Implicit table space created for pureXML columns
O	Table space is a LOB table space
G	Partitioned-by-growth table space

With DB2 9, the MAXPARTITIONS column shows the maximum number of partitions. The PARTITIONS column contains the number of physical partitions (data sets) that currently exist.

DB2 9 implicitly defines the XML table space as UTS, and the base table can be segmented, partitioned, or UTS. You need to define the XML base table as UTS if you want to take advantage of the DB2 10 XML versioning function (see 8.5, “Support for multiple versions of XML documents” on page 273).

4.1.2 UTS with DB2 10: The ALTER options

With DB2 9, you cannot convert a table space to UTS without a drop and re-create. In DB2 10, you can convert to UTS using ALTER and REORG. After the table space is converted to UTS, you can also change many other attributes, such as page size, or take advantage of new functions that require UTS. Thus, to be able to alter these other attributes, create all new table spaces as UTS, and as time and resources permit, convert all existing table spaces to UTS.

DB2 10 largely simplifies changes to table space structure and attributes.

Figure 4-1 shows the ALTER supported table space type conversions, which are as follows:

- Convert index partitioned table space to table partitioned
- Convert classic table partitioned table space to a range-partitioned table space adding SEGSIZE
- Convert simple table space with one table to a partition-by-growth table space
- Convert segmented table space with one table to a partition-by-growth table space
- Convert a partition-by-growth table space to hash table space
- Convert a range-partitioned tables space to hash table space

Archived

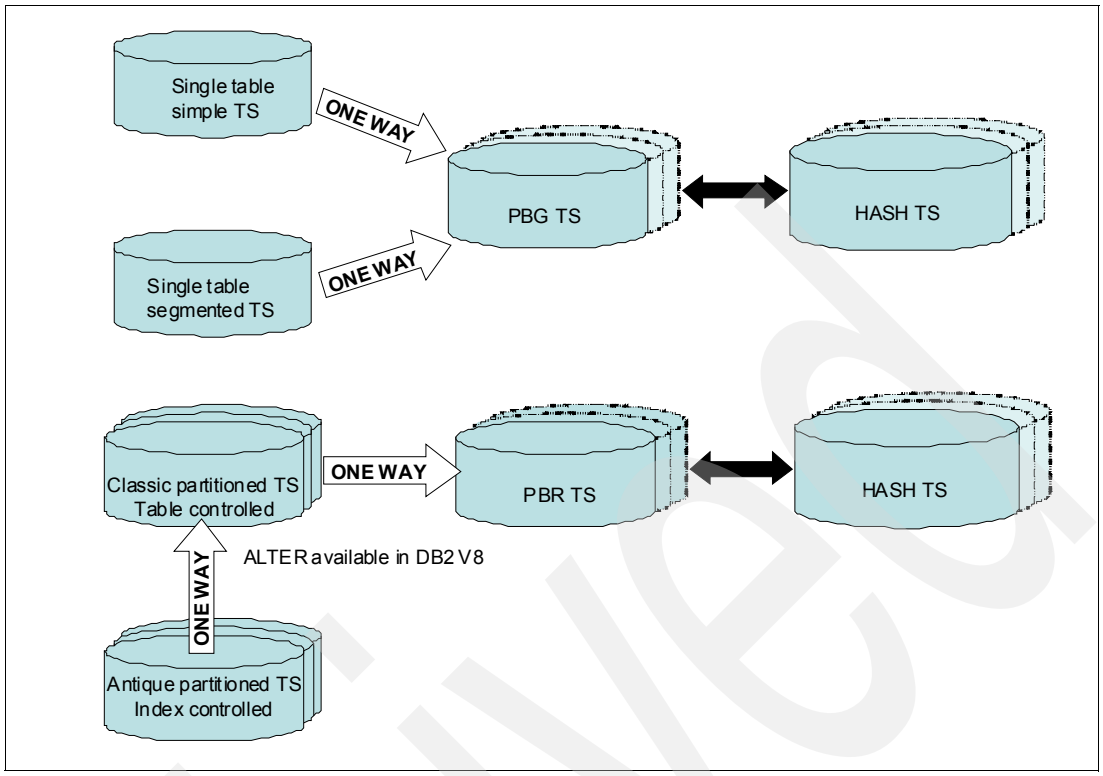


Figure 4-1 Possible table space type conversions

Additional notes: Since DB2 V8, you can convert from an index-partitioned table space to a table-partitioned table space and unbundle partitioning from clustering. For more information, see *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079.

Also, in some cases you can ALTER the table space back and remove your changes as the arrows indicate in Figure 4-1.

You cannot reverse the changes for converting from one type to a UTS type.

DB2 10 adds several options to ALTER table space attributes. Figure 4-2 can helps you understand the possible changes for the attributes:

- ALTER SEGSIZE of partition-by-growth table spaces, range-partitioned table spaces, and XML table spaces
- ALTER page size (buffer pool association) of partition-by-growth table spaces, range-partitioned table spaces, and LOB table spaces.
- ALTER page size of indexes on either range-partitioned or partition-by-growth table spaces.
- ALTER DSSIZE of partition-by-growth table spaces, range-partitioned table spaces, XML table spaces, and LOB table spaces.

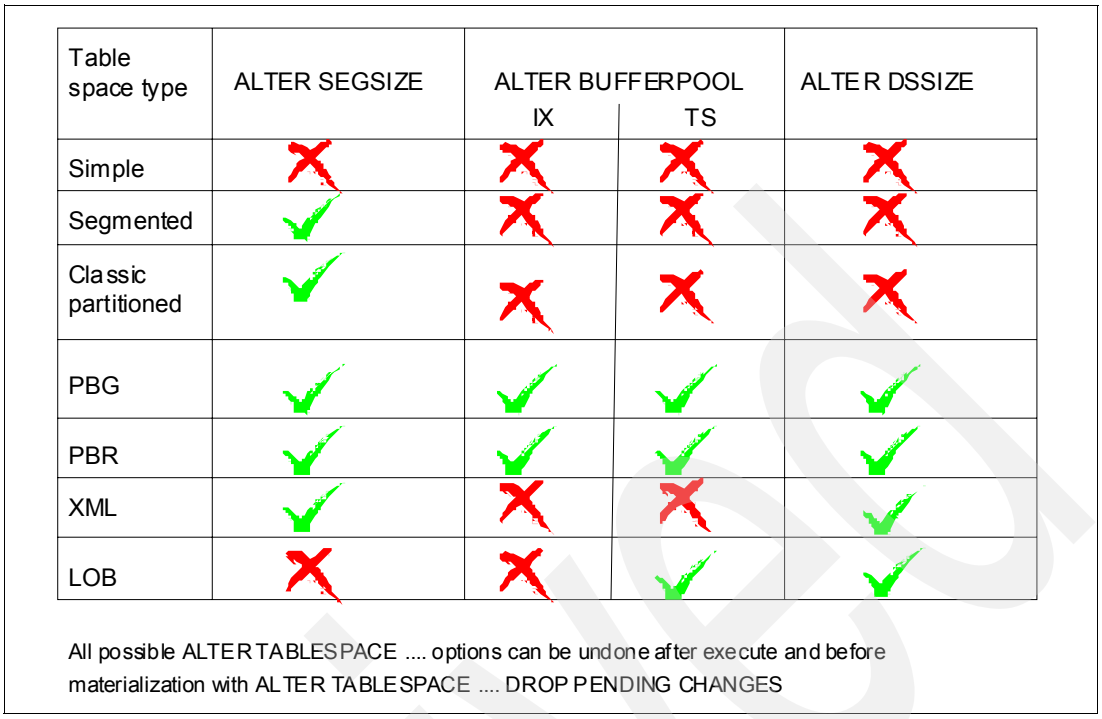


Figure 4-2 Possible online schema changes for table spaces and indexes

You can later reverse all possible alters shown in Figure 4-2 by increasing or decreasing the changed value respectively. An ALTER statement with a SEGSIZE clause is allowed for universal table spaces and partitioned table spaces that use table-controlled partitioning. You can add SEGSIZE to a table-controlled partitioned table space to convert it into a range-partitioned table space. Alternatively, you can add MAXPARTITIONS to convert a segmented or simple table space to a partition-by-growth table space and then change the attributes for BUFFERPOOL (page), DSSIZE and SEGSIZE.

We describe each type of online changes for DB2 10 in the following sections.

With DB2 9, depending on which change you made, the change can be implemented immediately by updating the attributes in the catalog and directory, and the page sets might be placed in pending states such as the restrictive *REORG-pending* (REORP), the advisory *REORG-pending* (AREO*) or *Rebuild-pending* (RBDP) with the associated consequences, such as indexes not being considered by the optimizer.

With DB2 10, in addition to new online schema changes, some of the changes that might restrict access to the changed database objects cause no immediate action and do not materialize immediately. DB2 10 introduces the concept of *pending changes* (which are materialized only when you reorganize the altered objects) and the new associated *advisory pending state* (AREOR), which we discuss in the next section. Other changes, which can be handled without the need to be pending, materialize immediately. One exception is the change from range-partitioned or partition-by-growth to hash organization, which provides a “mixture” between the pending definition change and immediate action.

With DB2 10, objects in REORP restrictive state or AREOR advisory pending state are reset by REORG with SHRLEVEL REFERENCE or CHANGE, instead of the restriction to just NONE in DB2 9.

4.1.3 Pending definition changes

Before we look in detail at the online schema changes available in DB2 10, we first explain the concept of pending definition changes, so that, when we differentiate between pending and immediate changes later, you can understand the concept that we are discussing.

If DB2 requires your schema change to be a pending definition change, semantic validation and authorization checking are done immediately when you execute the ALTER statement. The object that you alter is then placed into a new pending state called *advisory REORG-pending* (AREOR).

Objects in AREOR are available, but the state advises that you have to do a reorganization of the object or rebuild an index if you want the pending changes materialized. As with other advisory states, you do not see those objects if you issue the -DIS DB(...) SPACE(...) RESTRICT command. Because this is not a restrictive state, you must specify one of the following commands:

- -DIS DB(...) SPACE(...)
- -DIS DB(...) SPACE(...) ADVISORY

If a table space is placed in AREOR, it is accompanied by SQLCODE +610 for the ALTER. In addition, one or more rows are added to a the SYSIBM.SYSPENDINGDDL table. This table contains information about objects that have pending definition changes. Each pending option has a separate row, even if the options are specified on the same ALTER statement.

Table 4-1 shows the columns of the SYSIBM.SYSPENDINGDDL table.

Table 4-1 SYSIBM.SYSPENDINGDDL

Column name	Data type	Description
DBNAME	VARCHAR(24) NOT NULL	Name of the database for the pending option
TSNAME	VARCHAR(24) NOT NULL	Name of the table space for the pending option
DBID	SMALLINT NOT NULL	Identifier of the database
PSID	SMALLINT NOT NULL	Identifier of the table space page set description
OBJSCHEMA	VARCHAR(128) NOT NULL	The qualifier of the object that contains the pending option
OBJNAME	VARCHAR(128) NOT NULL	Name of the object that contains the pending option
OBJODIB	SMALLINT NOT NULL	Identifier of the object
OBJTYPE	CHAR(1) NOT NULL	1: Object is an index 2: Object is a table space
STATEMENT_TYPE	CHAR(1) NOT NULL	The type of the statement for the pending option A: An ALTER statement
OPTION_ENVID	INTEGER NOT NULL	Identifier of the environment for the pending option
OPTION _KEYWORD	VARCHAR(128) NOT NULL	The keyword of the pending option

Column name	Data type	Description
OPTION_VALUE	VARCHAR(4000) NOT NULL	The value of the pending option as string
OPTION_SEQNO	SMALLINT NOT NULL	The sequence of the pending option within the statement
CREATEDTS	TIMESTAMP NOT NULL	Timestamp when the pending option was created
IBMREQD	CHAR(1) NOT NULL	A value of Y indicates that the row came from the basic machine readable material (MRM) tape.
ROWID	ROWID	ID to support LOG columns for source text
STATEMENT TEXT	CLOB(2M) NOT NULL	The source text of the original statement for the pending option

The entries in this table exist only during the window between when the ALTER statements were executed and when they are finally materialized. Materialization and removal from the AREOR state occur when you run REORG TABLESPACE or REORG INDEX on the *entire table space* or *index space*.

Most of the introduced changes are processed as pending ALTERs. Immediate ALTERS occur only for the new online schema changes when the data sets of a table or index space are undefined when you issue the ALTER statement. In addition, the following ALTER statements are also considered as immediate ALTERs:

- Alter a table space to a buffer pool with the same page size
- Alter a partition-by-growth table space MAXPARTITIONS attribute
- Alter to hash
- Alter for an inline LOB

All other ALTERs require the pending procedure.

When a table space or index is in AREOR status, it is fully operational. However, numerous restrictions apply to DDL while pending changes are due for the table space or index. Refer to 4.1.6, “Impact on immediate options and DROP PENDING CHANGES” on page 89 for more information.

4.1.4 Online schema changes in detail

We listed the online schema changes in the introduction to this section. Now we discuss the different options in more detail.

Converting a simple or segmented table space to a partition-by-growth table space

With DB2 9, for simple and segmented table spaces, to change the table space structure to a partition-by-growth table space structure, you follow the same steps that we describe in “Converting a table controlled partitioned table space to a range-partitioned table space” on page 74.

With DB2 10, you use the MAXPARTITIONS extension to the ALTER TABLESPACE statement. Again, as with other ALTERs, if you specify MAXPARTITIONS and the table space

does not yet exist physically, the change takes effect immediately if no other changes are pending for this specific table space. If the table space data sets are already defined, this change is another pending definition change that is materialized by DB2 when you run the next REORG table space. Remember, you must use REORG SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

Note about using ALTER: You can use ALTER TABLESPACE ... MAXPARTITIONS only for *single* table simple or segmented table spaces. All UTS can hold only single table data.

As part of the materialization of pending definition changes, the following events occur:

- The value of column TYPE on catalog table SYSIBM.SYSTABLESPACE it set to *G*.
- If the original SEGSIZE of the table space is larger than or equal 32, then the new UTS inherits the original setting. In all other cases, the segsize is increased to 32.
- The DSSIZE is set to *4*.
- MEMBER CLUSTER setting is inherited from the original table space.
- If LOCKSIZE was set to TABLE, then it becomes LOCKSIZE TABLESPACE.

Converting a table controlled partitioned table space to a range-partitioned table space

With DB2 9, the procedure of changing from the classic partitioned table space structure to the new UTS structure impacts availability. You need to unload the data, drop the table space, re-create the table space, reload the data, regrant authorizations, and rebind related plans and packages, because DB2 9 does not allow you to alter existing table space definitions accordingly.

With DB2 10, the ALTER TABLESPACE statement allows you to change the table space definition from a classic partitioned table space to a UTS range-partitioned table space. The keyword to use is SEGSIZE. When you specify SEGSIZE, this keyword is mutually exclusive with the other keywords on your ALTER TABLESPACE statement.

Remember: As mentioned previously, all the changes that we describe in this section, including ALTER TABLESPACE SEGSIZE, are handled as an immediate change only if the table space data sets are not yet defined. If the table space is allocated physically, that is, the VSAM clusters exist, then this change is handled as a pending definition change, and your table space is placed in AREOR state.

When you materialize the pending definition change with REORG TABLESPACE, the following events occur:

- The value of column TYPE in SYSIBM.SYSTABLESPACE is changed to *R*.
- The MEMBER CLUSTER attribute is inherited from the original table space. DB2 9 does not support MEMBER CLUSTER for UTS. The fact that MEMBER CLUSTER is supported in DB2 10 for UTS makes this behavior possible. For more information about MEMBER CLUSTER and UTS, see 5.4, “Universal table space support for MEMBER CLUSTER” on page 114.
- If the existing FREEPAGE value is greater than or equal to the segment size that you specified on the ALTER TABLESPACE statement, then the number of FREEPAGE pages is adjusted downward to one page less than the segment size. For example, if the FREEPAGE was 16 and the specified SEGSIZE is 8, FREEPAGE is reduced to 7.
- The number of partitions is inherited from the original table space.

- If the value for DSSIZE was not specified upon creation of the original table space and therefore the current setting is 0, this is changed to the maximum possible DSSIZE. If DSSIZE was specified already, the range-partitioned table space inherits the setting from the original table space.

Table 4-2 summarizes the DSSIZE value considerations.

Table 4-2 Maximum DSSIZE depending on table space attributes

LARGE option used?	DSSIZE specified	Number of partitions	Value of DSSIZE in SYSTABLESPACE	Maximum size per partition
NO	NO	1 to 16	0	4 GB
NO	No	17 to 32	0	2 GB
NO	NO	33 to 64	0	1 GB
NO	NO	65 to 4096	0	4 GB
NO	NO	255 to 4096	> 0, the DSSIZE equals the page size of the table space	same as DSSIZE
YES	NO	1 to 4096	0	4 GB
NO	YES	1 to 4096	> 0, the DSSIZE must be valid, depending on the number of partitions and the page size of the table space	same as DSSIZE

Changing the page size of a UTS partition-by-growth table space

Prior to DB2 10, you can change the buffer pool of a table space, but you can do so only if both the original and the new page size is the same. With DB2 10, you can change the existing page size to either a smaller or larger new page size using the BUFFERPOOL keyword on an ALTER TABLESPACE statement.

Figure 4-3 shows an ALTER statement that changes the page size for the UTS partition-by-growth table space ITSODB.TS1 from 4 KB to 8 KB by assigning an 8 KB buffer pool to it. The result of this ALTER statement is SQLCODE +610, which is a warning message only, indicating that the table space is in AREOR state.

```
ALTER TABLESPACE ITSODB.TS1 BUFFERPOOL BP8K0;                          00006013
---------+---------+---------+---------+---------+---------+---------+---------
+
DSNT404I SQLCODE = 610, WARNING:  A CREATE/ALTER ON OBJECT ITSODB.TS1 HAS
         PLACED OBJECT IN ADVISORY REORG PENDING
DSNT418I SQLSTATE   = 01566 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXI14 SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 250 0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'000000FA'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

Figure 4-3 ALTER TABLESPACE ... BUFFERPOOL statement and resulting SQL code

As mentioned earlier, the ALTER statement this ALTER statement is not materialized immediately because this is a change to a buffer pool with a different page size. Instead, a row is added to SYSIBM.SYSPENDINGDDL that shows exactly which change is pending, what will be materialized to the catalog and your page set if you ran a REORG table space now. Figure 4-4 shows the row that was inserted when we executed the ALTER statement.

```
DBNAME        TSNAME          DBID    PSID  OBJSCHEMA     OBJNAME       OBJOBID
--------+---------+---------+---------+---------+---------+---------+---------
+
ITSODB        TS1              316       2  DB2R8         TS1                 2

OBJTYPE  STATEMENT_TYPE  OPTION_ENVID  OPTION_KEYWORD       OPTION_VALUE
------+---------+---------+---------+---------+---------+---------+-------
S        A                          4  BUFFERPOOL           BP8K0

OPTION_SEQNO  CREATEDTS                    RELCREATED  IBMREQD
-----+---------+---------+---------+---------+---------+------
           1  2010-08-10-14.57.31.521595  0           0

ROWID
--+---------+---------+---------+---------+--
AF1B52BF31663D0529040178992001000000000202

STATEMENT_TEXT
+---------+---------+---------+---------+-------
ALTER TABLESPACE ITSODB.TS1 BUFFERPOOL BP8K0
```

Figure 4-4 SYSIBM.SYSPENDINGDDL after ALTER

At this time, not much has happened to the table space. Only the advisory state AREOR is set on the table space. AREOR is an advisory state meant to remind you of the pending changes. For example, if you check SYSIBM.SYSPENDINGDDL and you see that you used the wrong BUFFERPOOL or that you did not really want to change the page size but the segment size of your table space, until the changes are pending, you can simply use the following line:

```
ALTER TABLESPACE ITSODB.TS1 DROP PENDING CHANGES;
```

DB2 deletes the pending changes and removes the table space from the AREOR state. Then, you return to where you started.

For this ALTER TABLESPACE ... BUFFERPOOL change to work, the buffer pool that you specify in your statement must be defined (in the DSNTIP2 installation panel or through ALTER BUFFERPOOL). That is, you must assign a certain VPSIZE to it. If you pick a buffer pool that is not activated, you receive the error message shown in Figure 4-5.

Archived

```
ALTER TABLESPACE ITSODB.TS1 BUFFERPOOL BP8K9;                          000060
---------+---------+---------+---------+---------+---------+---------+--------
DSNT408I SQLCODE = -647, ERROR:  BUFFERPOOL BP8K9 FOR IMPLICIT OR EXPLICIT
         TABLESPACE OR INDEXSPACE ITSODB.TS1 HAS NOT BEEN ACTIVATED
DSNT418I SQLSTATE   = 57003 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXIC01 SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 40 0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'00000028'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

Figure 4-5 -647 SQL code after ALTER TABLESPACE ... BUFFERPOOL

If you pick a buffer pool name that is invalid, such as BP8K22, you receive SQLCODE -204:

```
DSNT408I SQLCODE = -204, ERROR:  BP8K22 IS AN UNDEFINED NAME
```

Increasing or decreasing the page size

You might want to change the page size for better space utilization or to accommodate a planned change of the size of a column, which can impact the current page size. The table space for which you increase the page size, depending on the average size of the row, can lead to a reduction or increase of total space needed after REORG is run. If the space increases, you might consider changes to the DSSIZE as well. Refer to “Changing the DSSIZE of a table space” on page 80 for details on changing the DSSIZE of a table space.

If you have LOB columns in the table for which you increase the page size, consider the possible impact on LOB table spaces because auxiliary objects might be implicitly created for the newly-grown partitions. This situation occurs independently of whether you use SQLRULES(DB2) or SQLRULES(STD). Also, it does not matter whether your table space was created implicitly or explicitly. The new LOB objects inherit the buffer pool attribute from the existing LOB objects.

If the space decreases, keep in mind that for partition-by-growth table spaces empty partitions are not removed. After a partition is allocated and registered in the DB2 catalog, it remains defined even after REORG or LOAD. An empty partition is allocated with its primary allocation size.

If you try to ALTER the page size to a value smaller than the row length, you receive SQLCODE -670 and the change request is not honored.

In 4.1.5, “Materialization of pending definition changes” on page 86, we describe in detail what is needed to materialize the pending changes for the catalog and the table space.

This enhancement is in place only for UTS and LOB table spaces. You cannot change the page size for an XML table space.

Up to DB2 9, to change the buffer pool for a table space in a data sharing environment, you must stop the entire table space. With DB2 10, if you change the buffer pool to another buffer pool of the same page size, the change is immediate. If the change is related to a page size change, then it is a pending change just as for non-data sharing objects.

Possible restrictions for ALTER: For a list of possible restrictions for each ALTERs in each scenario, refer to the description for SQLCODE -650. This SQL code lists more than 20 reasons why the ALTER statement failed and is straight forward with the explanation of what exactly caused the problem.

Changing the page size of an index

Prior to DB2 10, you can change page sizes, assigning a different buffer pool with a different page size to an index. However, you cannot implement this change immediately. Therefore, the index is placed in *rebuild pending* (RBPD) state.

With DB2 10, you can change the page size of an index, and the index is no longer placed in an RBPD state if the index is of one of the following types:

- The index is defined on a table that is contained in UTS
- The index is defined on an XML table that is associated with a base UTS
- The index is defined on an AUXILIARY table that is associated with a base universal table space.

If the index is of one of these types, the ALTER is considered as a pending change. Therefore, the index is placed into AREOR state, which improves availability.

The method of changing a page size for an index is the same as for a table space, that is you use the BUFFERPOOL statement on your ALTER TABLESPACE command such as:

```
ALTER INDEX indexname BUFFERPOOL bufferpoolname
```

To materialize the change and, therefore, resolve the AREOR state, you can choose between running REORG INDEX and REORG TABLESPACE. Both can be run with SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

If after materialization of your ALTER you find that you want to revert back to the old page size and assign it to the same or perhaps a different buffer pool, you can do so by issuing another ALTER INDEX ... BUFFERPOOL statement. Of course, your index is again placed in the AREOR state, and you must REORG the index to materialize the change.

For other indexes, such as indexes on classic segmented table spaces, DB2 behavior for ALTER INDEX processing has not changed.

Up to DB2 9, to change the buffer pool for an index space in a data sharing environment, you must stop the entire index space. With DB2 10, if you change the buffer pool to another one of the same page size, the change is immediate. If the change is related to a page size change, then it is a pending change just as for non-data sharing objects.

Changing the SEGSIZE of a UTS

Prior to DB2 10, altering the segment size of a table space was not allowed. With DB2 10, you can alter the segment size of UTS.

When you want to make use of this option, ALTER TABLESPACE ... SEGSIZE does not allow any additional options in this statement. However, you can issue several ALTER TABLESPACE statements in a sequence such as:

```
ALTER TABLESPACE ITSODB.TS1 BUFFERPOOL BP16K0;
ALTER TABLESPACE ITSODB.TS1 DSSIZE 4 G ;
```

Each ALTER then leads to a new row in SYSIBM.SYSPENDINGDDL.

Archived

Upon changing the segment size, if the existing FREEPAGE[1] value is greater than or equal to the new segment size, then the number of pages is adjusted downward to one less than the new segment size. For example:

```
FREEPAGE = 12
SEGSIZE = 4
=> calculated FREEPAGE is 3
```

Figure 4-6 shows what can happen if you ALTER your table space from for example SEGSIZE 16 to SEGSIZE 4. With the FREEPAGE clause, you specify how often DB2 leaves a full page of free space when loading data or when reorganizing data or indexes. For example, if you specify FREEPAGE 12, every twelfth page is left empty.

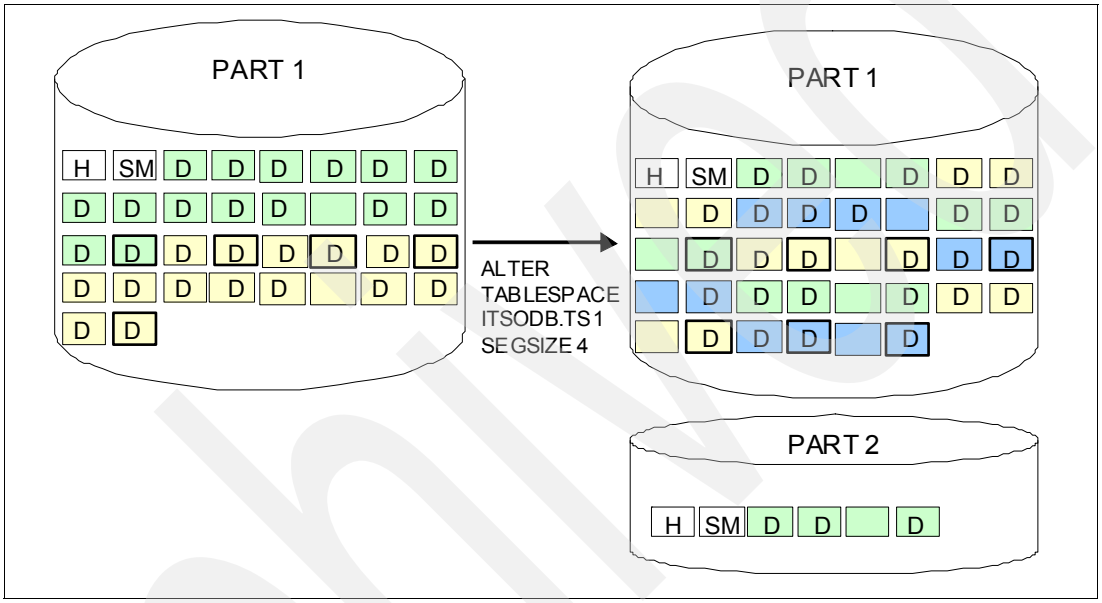


Figure 4-6 ALTER TABLESPACE ... SEGSIZE decreased

Because DB2 reduces the FREEPAGE value to 3 when you reorganize the table space to materialize the changes, you will get many more blank pages. Therefore, the entire table space needs more space. In Figure 4-6, the unaltered table space on the left side of the figure has one almost full partition with 30 data pages in two segments, where each partition has one FREEPAGE. After the REORG, each third page is now a FREEPAGE. The given DSSIZE allows DB2 to store only 27 data pages in the table space. As a result, DB2 must allocate one more segment on an additional table space partition.

As discussed earlier, you can prevent DB2 from allocating new partitions to store the old amount of data by increasing the DSSIZE using the following statement:

```
ALTER TABLESPACE ITSODB.TS1 DSSIZE n G
```

Where *n* is larger than the previous one.

If you use ALTER TABLESPACE ... SEGSIZE *n* to increase your segment size, the FREEPAGE information is not adjusted, because it always fits into your segments in this case.

If you have LOB columns in the table for which you decrease the SEGSIZE and if additional table space partitions are needed due to the described considerations, additional LOB table

[1] The FREEPAGE clause specifies how often DB2 leaves a full page of free space when loading data or when reorganizing data or indexes. The maximum value you can specify for FREEPAGE is 255; however, in a segmented or UTS, the maximum value is 1 less than the number of pages specified for SEGSIZE.

spaces and auxiliary objects are implicitly created for the newly-grown partitions. This creation is independent of whether you use SQLRULES(DB2) or SQLRULES(STD). Also, it does not matter whether you table space was created implicitly or explicitly.

Changing the DSSIZE of a table space

With DB2 10 you can change the data set size of a table space. The DSSIZE keyword on the ALTER TABLESPACE allows you to apply this change to the universal and LOB table spaces. When you use the DSSIZE keyword, no other ALTER TABLESPACE clause is allowed in the same ALTER statement.

The DSSIZE value must be valid, depending on the values in effect for the maximum number of partitions and the page size of the table space. Table 4-3 shows the valid combinations for page size, number of partitions, and DSSIZE.

Table 4-3 Number of partitions by DSSIZE and page size

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

If you have LOB columns in the table for which you decrease the DSSIZE, and additional table space partitions are needed, additional LOB table spaces and auxiliary objects are implicitly created for the newly-grown partitions. This creation is independent of whether you use SQLRULES(DB2) or SQLRULES(STD). Also, it does not matter whether your table space was created implicitly or explicitly.

Convert a partition-by-growth or range-partitioned table space to hash table space

We describe the general functionality of a hash table space, the performance expectations, and how to choose candidate table spaces in detail in 13.15, “Hash access” on page 575. Here, we focus on how to convert from UTS to hash.

If you want to convert your partition-by-growth or range-partitioned table space to hash, there are a few restrictions that apply to both conversions:

- The hash column or columns must be defined as NOT NULL.
- The table to be converted cannot be APPEND(YES) or MEMBER CLUSTER.
- The table to be converted must be in a UTS (partition-by-growth or range-partitioned).
- You cannot hash global temporary tables.

As shown in Figure 4-1 on page 70, you can convert only UTS (either partition-by-growth or range-partitioned) to the hash organization.

Let us assume that your analysis shows that the hash organization is beneficial for your application, and you want to change the organization of your partition-by-growth table space to hash. In this example, you have one partition-by-growth table space with one unique index

on column EMPNO. The table space as it exists now has only 15 tracks allocated to it, because it only stores 42 rows.

Example 4-1 shows the two clusters for the table space and the index space.

Example 4-1 Table space and index space as partition-by-growth

```
Command - Enter "/" to select action                          Tracks %Used   XT
 ------------------------------------------------------------------------------
          DBOBD.DSNDBD.HASHDB.HASHIX.I0001.A001                   15    ?     1
          DBOBD.DSNDBD.HASHDB.HASHTS.I0001.A001                   15    ?     1
```

The ALTER TABLE syntax is enhanced to include the keywords shown in Figure 4-7 and Figure 4-8.

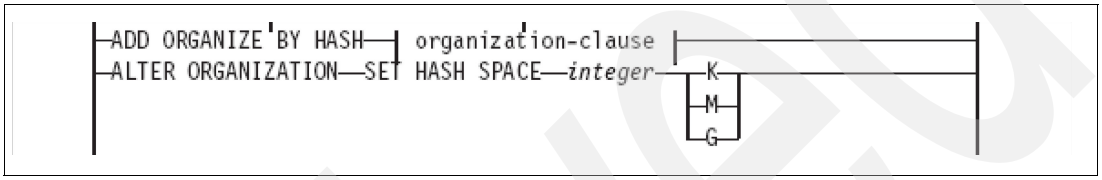


Figure 4-7 ALTER TABLE ADD ORGANIZE BY HASH

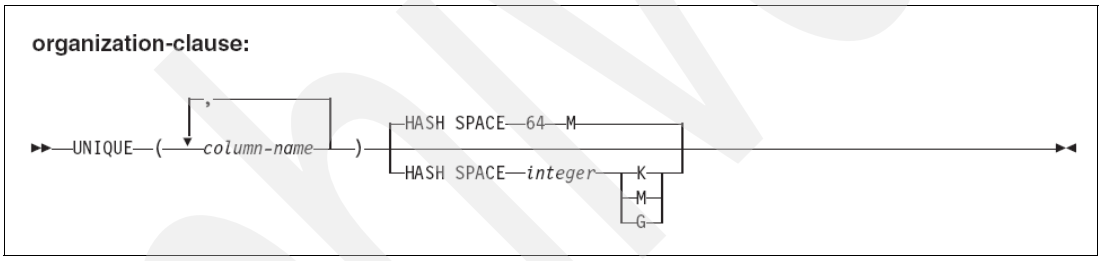


Figure 4-8 ALTER TABLE organization-clause

If you combine these two pieces, the syntax to change the organization of the table space HASHDB.HASHTS from partition-by-growth to hash is as follows:

```
ALTER TABLE HASHTB ADD ORGANIZE BY HASH
 UNIQUE (EMPNO) HASH SPACE 1 M;
```

After issuing this SQL statement, you receive SQL code +610, which informs you that your table space is placed in AREOR state. AREOR is an advisory state and, therefore, does not restrict the use of the underlying object. In addition, a hash *overflow* index is created, which is not built immediately but which is placed into Page Set Rebuild pending[2] (PSRBD) restrictive state.

[2] The entire (non-partitioning) hash index space is in Page Set Rebuild Pending status.

Figure 4-9 shows that the original unique index that you defined on the table is untouched. It is, however, now obsolete, because the hash overflow index is also a unique index that enforces uniqueness over the same column as the original index.

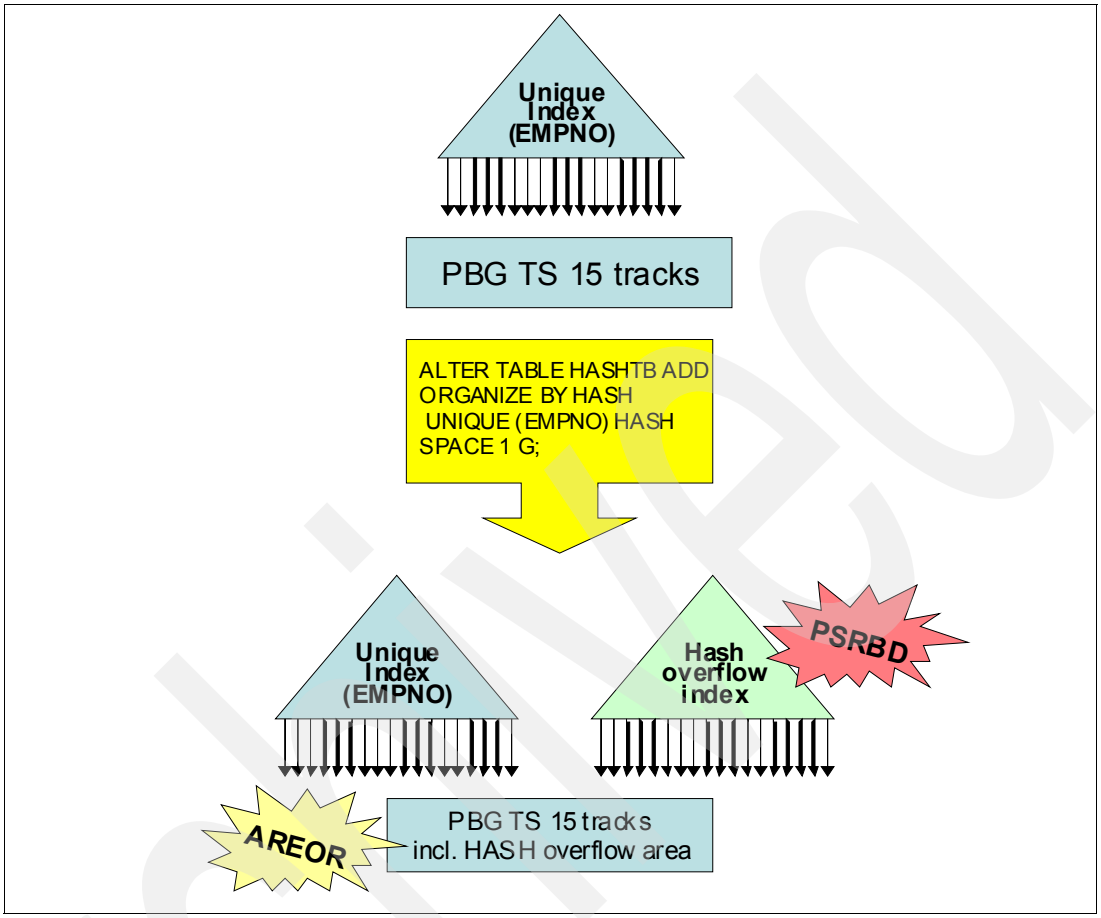


Figure 4-9 Partition-by-growth to hash

In the current situation, you can execute UPDATEs and DELETEs, but INSERTs are not allowed. This situation is caused by the restrictive PSRBD state on the hash overflow index. To remove PSRBD from the index you have two choices:

- REBUILD the hash overflow index.
- REORG the table space.

If you REBUILD the hash overflow index, the restrictive state PSRBD is removed, and your table space is again fully available for read and write access. However, the hash organization is not yet applied, and if you insert rows into the table now, the hash algorithm is not used for it. However, the catalog reflects the new pending structure, so DB2 knows that when you run your next REORG, it has to reload the data into the table space using the hash algorithm.

Tip: Although the REBUILD of the hash overflow index is a theoretical option, practically it is not beneficial. The real hash structure is applied to the table space only after you run REORG TABLESPACE.

You can check the HASH column in SYSIBM.SYSINDEXES. If the value here is set to Y, the index is a hash overflow index, and the next reorganization applies the hash organization to the table space. The HASHSPACE and HASHDATAPAGES columns in the

SYSIBM.SYSTABLESPACE table also provide useful information. After the ALTER and before the actual REORG, HASHSPACE is set to 1048576 and HASHDATAPAGES is set to 0 (zero). A non-zero number in HASHSPACE and 0 in HASHDATAPAGES indicate that an ALTER is submitted for the table space organization, but the REORG that actually applies the new structure has not yet been run.

When you run the REORG TABLESPACE utility to finally apply the hash structure to the table space, you can choose either YES or NO with the AUTOESTSPACE option. If you leave the AUTOESTSPACE parameter untouched, the default setting of YES is applied. YES means that DB2 uses RTS information to calculate the hash space, which is preallocated for you. The needed space depends heavily on the number of rows that fit into one data page. The smaller this number, the more space is needed to efficiently hash the data in the table space.

DB2 ignores the HASHSPACE value in SYSIBM.SYSTABLESPACE during REORG and does not change HASHSPACE to the values that it calculates. However, the HASHDATAPAGES column is adjusted to the real number of pages that DB2 preallocates during the REORG. In our example, we ran the reorganization of our table space using the utility control statement shown in Example 4-2.

Example 4-2 REORG TABLESPACE with AUTOESTSPACE YES

```
REORG TABLESPACE HASHDB.HASHTS SHRLEVEL REFERENCE AUTOESTSPACE YES
```

The number of preallocated data pages is reported in the REORG job output, shown in Example 4-3. In our example we have 61 preallocated pages.

Example 4-3 REORG utility message DSNUGHSH

```
DSNUGHSH - THE HASH SPACE HAS BEEN PREALLOCATED AT 61 FOR TABLESPACE HASHDB.HASHTS
PART 0
```

The result of the REORG is visualized in the upper portion of Figure 4-10. We have a fix sized HASH SPACE, which does not grow if you insert more data than expected in a hash overflow space, which is variable in size. However, as explained in 13.15, “Hash access” on page 575, this insert has a negative performance impact if the hash overflow space goes beyond a certain size.

Archive

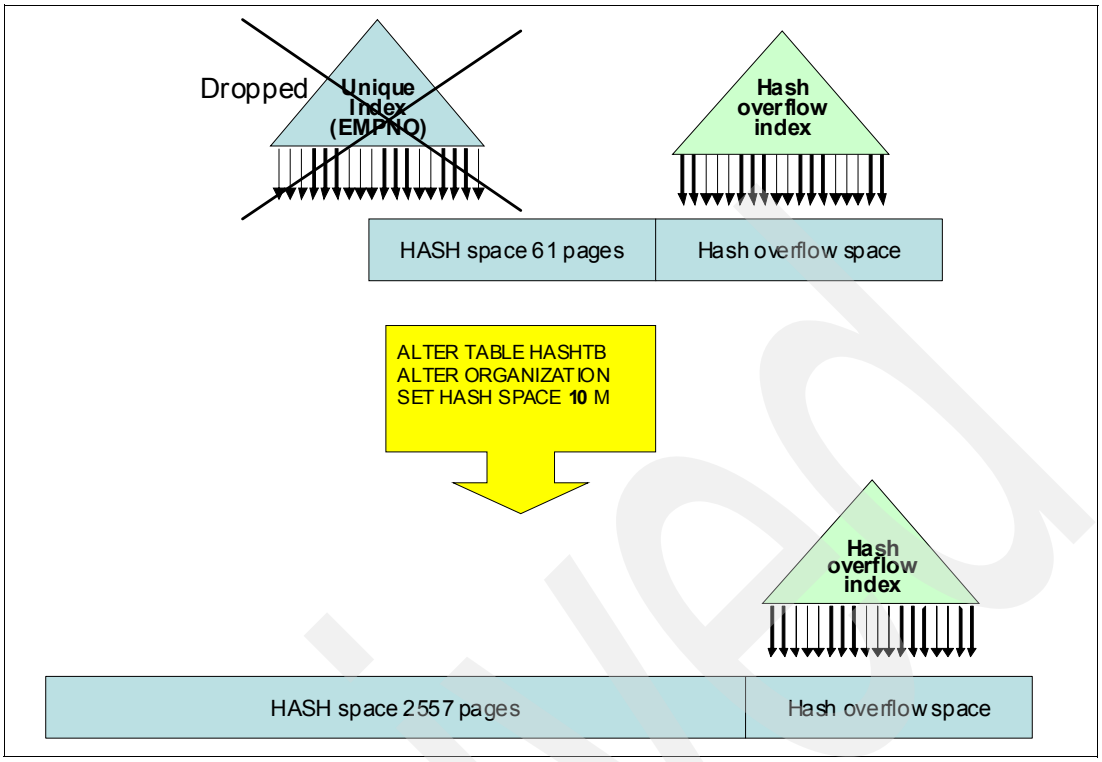


Figure 4-10 REORGed HASH SPACE

If you find that the HASH SPACE is too small because you have too many rows allocated in the hash overflow space, you can use the syntax for ALTER TABLE shown in Figure 4-7 on page 81.

In our example, we used the ALTER TABLESPACE statement shown in Figure 4-10 to increase the HASH SPACE to 10 M. This change is now reflected in the HASHSPACE column in SYSIBM.SYSTABLESPACE. The value is 10240. In fact, in our case, we decreased the number we previously used from 1 GB to 10 MB. DATAPAGES still shows 61, because we did not yet REORG the table space to apply the new size.

For the subsequent REORG, we used the control statement shown in Example 4-4.

Example 4-4 REORG TABLESPACE with AUTOESTSPACE NO

```
REORG TABLESPACE HASHDB.HASHTS SHRLEVEL REFERENCE AUTOESTSPACE NO
```

With this REORG, we ask DB2 to not estimate space for the HASH SPACE but to make the HASH SPACE 10 MB large. The REORG now preallocates 2557 pages and updates column DATAPAGES in SYSIBM.SYSTABLESPACE with this number. You can look at the data set size under TSO as shown in Example 4-5.

Example 4-5 Data set sizes after REORG

```
Command - Enter "/" to select action                    Tracks %Used   XT
------------------------------------------------------------------------------
         DBOBD.DSNDBD.HASHDB.HASHIX.I0001.A001             15    ?      1
         DBOBD.DSNDBD.HASHDB.HASHTBAB.I0001.A001           15    ?      1
         DBOBD.DSNDBD.HASHDB.HASHTS.I0001.A001            315    ?      2
```

Note: The REORG must be SHRLEVEL REFERENCE or SHRLEVEL CHANGE. SHRLEVEL NONE is not allowed.

Converting from hash to partition-by-growth or range-partitioned

If you determine that you do not benefit from the HASH organization, you can convert an existing hash table space to partition-by-growth or range-partitioned. The ALTER TABLE syntax includes the DROP ORGANIZATION key word.

Figure 4-11 shows the simple syntax for converting the hash table space to a partition-by-growth table space or a range-partitioned table space. When you issue the ALTER TABLE statement, the hash overflow index is immediately dropped and the table space is placed in *REORG pending* (REORP) state. This state is a restrictive state. You must run a REORG before you can use it. The REORG must either be SHRLEVEL REFERENCE or SHRLEVEL CHANGE.

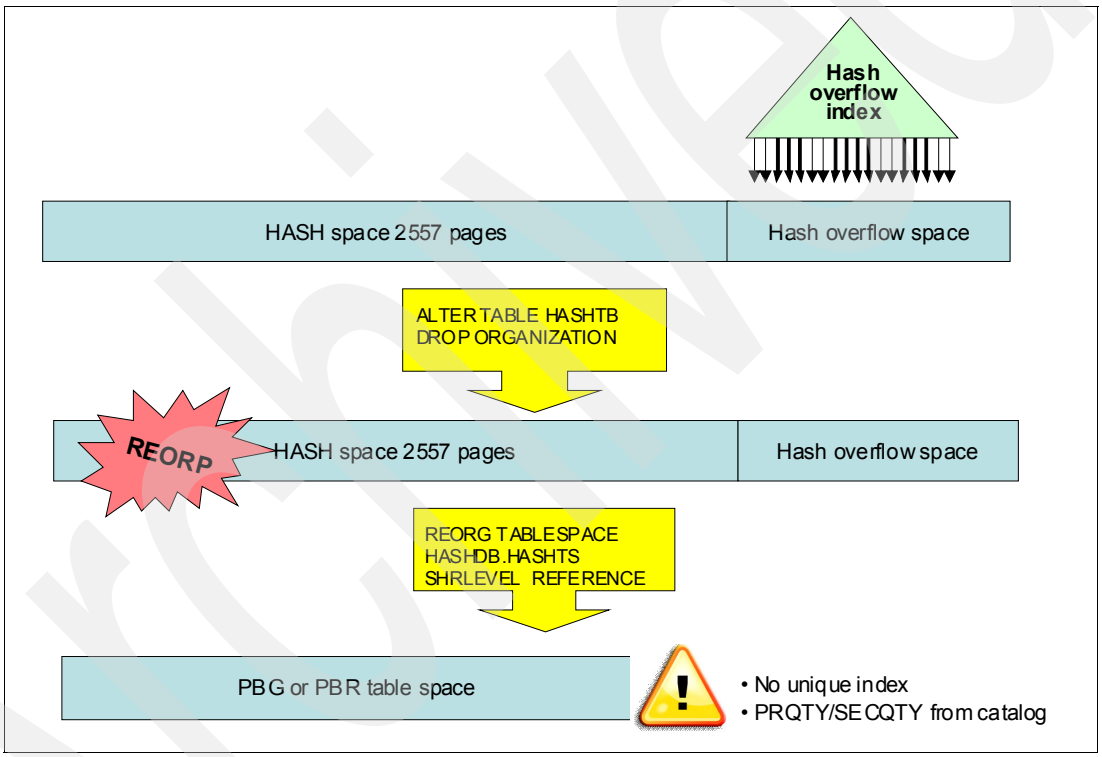


Figure 4-11 DROP ORGANIZATION

As part of the REORG, the table space is reallocated with the PRQTY and SEQTY, which are stored in the DB2 catalog and directory.

Attention: Immediately after you issue the ALTER TABLE DROP ORGANIZATION, the hash index is deleted. If you rely on the uniqueness of those values the hash key was on and if you do not have any additional index taking care of it, you must create a new unique index immediately after dropping the organization and before you run the REORG.

Archive

4.1.5 Materialization of pending definition changes

If you want to materialize the pending changes, you can use REORG TABLESPACE for table spaces and indexes or REORG INDEX for indexes.

Important: You must use either REORG TABLESPACE SHRLEVEL REFERENCE or SHRLEVEL CHANGE. SHRLEVEL NONE will not materialize the pending changes.

The same is true for REORG INDEX.

Running REORG TABLESPACE

The REORG utility is enhanced to handle the associated steps that are necessary to fully accommodate the schema change, which includes the following steps:

1. Let us assume that we have used the following ALTER TABLESPACE statements, which are considered as pending changes:

   ```
   ALTER TABLESPACE ITSODB.TS1 DSSIZE 8 G;
   ALTER TABLESPACE ITSODB.TS1 SEGSIZE 8;
   ```

   With the materialization, DB2 updates the catalog definition with the pending definitions. Changes for the table space and associated indexes are applied in the same order as they were inserted into the SYSIBM.SYSPENDINGDDL catalog table.

   Figure 4-12 shows an extract of the DB2 messages that are part of the REORG TABLESPACE SHRLEVEL REFERENCE job output. Those messages document the catalog changes that REORG initiated.

```
DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1 FOR TABLE DB2R8.ITSODB.TS1
DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=1
DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE ITSODB.TS1
DSNUGPAM - APPLYING PENDING DEFINITION CHANGES COMPLETE FOR ITSODB.TS1
DSNURSWD - SOME PARTITION STATISTICS MAY HAVE BECOME OBSOLETE ON ITSODB.TS1
DSNUSUTP - SYSTABLEPART CATALOG UPDATE FOR ITSODB.TS1 SUCCESSFUL
DSNUSUPT - SYSTABSTATS CATALOG UPDATE FOR DB2R8.ITSOTB1 SUCCESSFUL
DSNUSUPC - SYSCOLSTATS CATALOG UPDATE FOR DB2R8.ITSOTB1 SUCCESSFUL
DSNUSUTB - SYSTABLES CATALOG UPDATE FOR DB2R8.ITSOTB1 SUCCESSFUL
DSNUSUCO - SYSCOLUMNS CATALOG UPDATE FOR DB2R8.ITSOTB1 SUCCESSFUL
DSNUSUTS - SYSTABLESPACE CATALOG UPDATE FOR ITSODB.TS1 SUCCESSFUL
DSNUSEF2 - RUNSTATS CATALOG TIMESTAMP = 2010-08-10-21.01.16.373124
DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
```

Figure 4-12 REORG TABLESPACE messages for materialization of pending changes

2. Update the user data sets with the new table space and or index attributes
3. Collect statistics for the table space and associated indexes with the default options, that is TABLE ALL, INDEX ALL, and HISTORY ALL, if you did not specify the STATISTICS keyword. The last entry in Figure 4-12 shows that REORG invoked the RUNSTATS utility.
4. Invalidate dependent plans, packages and dynamic statements if any materialized change requires it. Refer to 4.1.4, “Online schema changes in detail” on page 73 for details.
5. Regenerate dependent views if any materialized pending change requires it.
6. CREATE SYSCOPY records for the pending definition change that have been materialized.

Table 4-4 shows the new ICTYPE, STYPE and TTYPE columns that have been introduced to table SYSIBM.SYSCOPY for the new online schema changes.

Table 4-4 New SYSIBM.SYSCOPY information

Pending alteration	ICTYPE	STYPE	TTYPE
ALTER TABLESPACE BUFFERPOOL	A (Alter)	F: The page size was altered	Old page size in units of K (4, 8, 16, or 32 KB)
ALTER TABLESPACE DSSIZE	A (Alter)	D: The DSSIZE was altered	Old DSSIZE in units of G, M, or K up to 64 GB
ALTER TABLESPACE SEGSIZE	A (Alter)	S: The SEGSIZE was altered	Old SEGSIZE (n) or the conversion from partitioned to range-partitioned (P)
ALTER TABLESPACE MAXPARTITIONS	A (Alter)	M: The MAXPARTITIONS was altered	*n* if there was an alteration from MAXPARTITIONS n *S* if there was an alteration of MAXPARTITIONS to convert a single-table segmented table space to a range-partitioned table space *I* if there was an alteration of MAXPARTITIONS to convert a single table simple table space to a partition-by-growth table space

7. The entries in SYSIBM.SYSPENDINGDDL, which are related to the reorganized table space, are removed.
8. The utility job will at least encounter return code 4, which is set based on message DSNU1166I, which is also shown in Figure 4-12. This message is to warn you that the following partition statistics might now be obsolete:
   - COLGROUP
   - KEYCARD
   - HISTOGRAM
   - frequency statistics with NUMCOLS > 1
   - statistics for extended indexes

   Use the RUNSTATS utility to correct these wrong statistics.
9. If REORG completes successfully, the table space and associated indexes are removed from the AREOR state.

   Note that only REORG resets the AREOR state, LOAD REPLACE does not.

If the target table space is a partition-by-growth table space, the utility must take a few more steps to support partition growth or reduction before it can perform steps 7 through 9 from the previously described procedure:

1. Determine the actual number of partitions based on the amount of existing user data and the new table space attributes. If the new SEGSIZE or DSSIZE attribute is smaller than the original one, partition growth might happen. Conversely, if the new SEGSIZE or DSSIZE attribute is bigger than the original one, partition reduction might happen.
2. Add additionally needed partitions.

3. Add additional LOB objects for partitions added in step 2. This addition always occurs automatically, independently of the setting for SQLRULES special register or whether the table space was created implicitly or explicitly. During the time that it takes to generate the new LOB objects, add this information to the DB2 catalog, and materialize the LOB objects, rows are added to another new DB2 table SYSIBM.SYSPENDINGOBJECTS. SYSIBM.SYSPENDINGOBJECTS stores information about the name and OBID for the new LOB objects.

   Table 4-5 shows the layout of the SYSIBM.SYSPENDINGOBJECTS catalog table.

Table 4-5 SYSIBM.SYSPENDINGOBJECTS

Column name	Data type	Description
DBNAME	VARCHAR(24) NOT NULL	Name of the database
TSNAME	VARCHAR(24) NOT NULL	Name of the base table space
DBID	SMALLINT NOT NULL	DB2 assigned identifier of the database
PSID	SMALLINT NOT NULL	DB2 assigned identifier of the base table space page set descriptor.
PARTITION	SMALLINT NOT NULL	Partition number with which the object is associated.
COLNAME	VARCHAR(128) NOT NULL	Name of the column contained in the base table space with which the object is associated
OBJSCHEMA	VARCHAR(128) NOT NULL	Identifier of the object
OBJNAME	VARCHAR(128) NOT NULL	Name of the object
OBJTYPE	CHAR(1) NOT NULL	Type of object identified by OBJSCHEMA and OBJNAME. I: The object is an index S: The object is a table space T: The object is a table
INDEXSPACE	VARCHAR(24) NOT NULL	Name of the index space. An empty string if the object is not an index.
OBJOBD	SMALLINT NOT NULL	DB2 assigned identifier of the object
OBJPSID	SMALLINT NOT NULL	DB2 assigned identifier of the object page set descriptor, or 0 if the object does not have a page set descriptor.

4. Move the user data from one partition to another if partition growth has happened.

Running REORG INDEX

If the applied schema changes are related only to indexes and not to table spaces, you do not need to run REORG TABLESPACE to materialize the pending index changes. REORG INDEX SHRLEVEL REFERENCE or CHANGE will apply the changes.

The necessary steps that are added to the REORG INDEX function are similar to the steps that we described in detail for the REORG TABLESPACE utility in “Running REORG TABLESPACE” on page 86.

Archive

Errors associated with the materialization of pending changes

You might receive the following errors that are associated with the materialization of pending changes:

- If the SEGSIZE for a range-partitioned table space, an XML table space using range-partitioning, or a LOB table space is smaller than the original one, it might be that the number of pages that are used for space map pages is increased. In this case it might be that there are not enough pages to accommodate the user data. In this case, the utility fails with error message DSNU1167I. As a result, pending changes are still pending.

  To fix this issue, you can run another ALTER TABLESPACE statement that changes the SEGSIZE back to a bigger one. This must not necessarily be the same big size that you had before you started the online schema change process, but you cannot reduce the current SEGSIZE due to the described potential problem.

- If the new DSSIZE for a range-partitioned table space, an XML table space using range-partitioning, or a LOB table space is smaller than the original one, it might be that the new DSSIZE is too small to accommodate the existing user data. In this case, the utility terminates, and you get message DSNU1167I with a return code 8.

  To fix this problem, you can issue another ALTER TABLESPACE statement with a larger DSSIZE and rerun the job.

  You cannot change DSSIZE to a smaller value for a given table space.

- If you increase SEGSIZE and decrease DSSIZE for a partition-by-growth table space, it might occur that the calculated number of partitions that are needed to store the existing data exceeds the currently defined number for MAXPARTITIONS. In this case the utility terminates and you receive message DSNU1170I with a return code 8.

  You can choose from the following options to fix this issue and then rerun the utility job:

  - Execute an ALTER TABLESPACE statement to change to a bigger SEGSIZE.
  - Execute an ALTER TABLESPACE statement to change to a bigger DSSIZE.
  - Execute an ALTER TABLESPACE statement to change to a bigger MAXPARTITIONS.

4.1.6 Impact on immediate options and DROP PENDING CHANGES

After you execute an ALTER TABLESPACE or ALTER INDEX statement that requires a pending change and before you run REORG TABLESPACE or REORG INDEX to materialize those changes to the catalog and the user page sets, you can deactivate those pending changes using DROP PENDING CHANGES on a ALTER INDEX or ALTER TABLESPACE statement.

For example, if you execute the following SQL statement for a table space that is currently a 4 KB page set, the message shown in Figure 4-13 informs you that table space DB1.TS1 is now in AREOR state:

```
ALTER TABLESPACE DB1.TS1 BUFFERPOOL BP16K0;
```

```
DSNT404I SQLCODE = 610, WARNING:  A CREATE/ALTER ON OBJECT DB1.TS1 HAS PLACED
         OBJECT IN ADVISORY REORG PENDING
DSNT418I SQLSTATE   = 01566 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXI14 SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 250 0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'000000FA'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

Figure 4-13 SQLCODE +610 message

As a consequence, this example inserts the row about a pending change shown in Figure 4-14 into SYSIBM.SYSPENDINGDDL.

```
---------+---------+---------+---------+---------+---------+---------+------
DBNAME     TSNAME       DBID    PSID  OBJSCHEMA  OBJNAME    OBJOBID  OBJTYPE
---------+---------+---------+---------+---------+---------+---------+------
DB1        TS1           286       2  DB2R8      TS1              2  S
... continued
-------+---------+---------+---------+---------+---------+---------+---------
STATEMENT_TYPE  OPTION_ENVID  OPTION_KEYWORD   OPTION_VALUE     OPTION_SEQNO
-------+---------+---------+---------+---------+---------+---------+---------
A                          4  BUFFERPOOL       BP16K0                      1
... continued
CREATEDTS                   RELCREATED  IBMREQD
-+---------+---------+---------+---------+------
2010-08-02-18.55.55.546068  0           0
... continued
--+---------+---------+---------+---------+--------
ROWID
--+---------+---------+---------+---------+--------
7B0FA4C6F7A6331E2904017899200100000000000202
...continued
-+---------+---------+---------+---------+---
 STATEMENT_TEXT
-+---------+---------+---------+---------+---
 ALTER TABLESPACE DB1.TS1 BUFFERPOOL BP16K0
```

Figure 4-14 One wrapped row showing pending change in SYSPENDINGDDL

Let us assume that while your table space is in AREOR, you change the column length for one of the columns of table TB1, which resides in table space DB1.TS1. For this purpose you now issue the following statement:

```
ALTER TABLE TB1 ALTER COLUMN LASTNAME SET DATA TYPE CHAR(30)
```

Unfortunately, because the table space that is holding this column has pending changes waiting to materialize, the normally immediate change of altering the column length is not allowed, and you receive the error message shown in Figure 4-15.

```
DSNT408I SQLCODE = -20385, ERROR:  THE STATEMENT CANNOT BE PROCESSED BECAUSE
         THERE ARE PENDING DEFINITION CHANGES FOR OBJECT DB1.TS1 OF TYPE
         TABLESPACE (REASON 2)
```

Figure 4-15 SQLCODE for normally immediate change due to pending change

Thus, depending on the urgency of the ALTER DATA TYPE change, you now have multiple choices to reenable the table space for immediate changes.

One sequence of steps is as follows:

1. REORG TABLESPACE SHRLEVEL REFERENCE or CHANGE.
2. ALTER TABLE ... SET DATA TYPE.
3. REORG TABLESPACE again because it now is in AREOR, which in terms of performance advises you to reorganize your table space.

Another sequence of steps is as follows:

1. ALTER TABLESPACE DB1.TS1 DROP PENDING CHANGES.
2. ALTER TABLE ... SET DATA TYPE.
3. ALTER TABLESPACE DB2.TS1 BUFFERPOOL BP16K0.
4. REORG TABLESPACE SHRLEVEL CHANGE or REFERENCE.

The second sequence has one step more, but in terms of resource usage, it is the preferred option in this situation.

Attention: If you want to remove the currently pending definition change for a given table space, you must use the DROP PENDING CHANGES keyword on ALTER TABLESPACE or ALTER INDEX command.

Issuing a second ALTER TABLESPACE command that reverses the previous change, that is for example first change from buffer pool BP0 to BP16K0 and then with a second ALTER change from BP16K0 to BP0, is *not* sufficient.

This second ALTER is also added to SYSIBM.SYSPENDINGDDL and executed in the sequence of entering those commands when you run the next REORG TABLESPACE utility.

The immediate ALTER TABLE SET DATA TYPE option is just one sample for the many immediate options that are not immediate if there are pending changes on database objects.

Pending definition changes can have the following impacts on various immediate options:

- The ALTER TABLE statement is not allowed for the table space that has currently pending definition changes.
- The following DDL statements are not allowed if there are any pending definition changes for the index or the table space:
  - ALTER INDEX to change PIECESIZE
  - ALTER INDEX REGENERATE
  - ALTER INDEX ADD COLUMN
- The following DDL statements are not allowed if there are any pending definition changes for the table space and the table space is not a PGB
  - ALTER TABLESPACE to change from a DB2-managed data set to a user-managed data set
  - ALTER TABLESPACE ALTER PARTITION to change from a DB2-managed data set to a user-managed data set
- An ALTER TABLESPACE FREEPAGE or ALTER TABLESPACE ALTER PARTITION FREEPAGE statement is not allowed if there are any pending definition changes for the table space.
- An ALTER TABLESPACE statement to change CCSID is not allowed if there are any pending definition changes for the table space.
- The following DDL statements are not allowed if there are any pending definition changes for the index:
  - ALTER INDEX to change from a DB2-managed data set to a user-managed data set
  - ALTER INDEX ALTER PARTITION to change from a DB2-managed data set to a user-managed data set

  - ALTER INDEX to change from COMPRESS NO to COMPRESS YES
  - RENAME INDEX
- A CREATE INDEX statement is not allowed if there are any pending definition changes for the table space or any objects within the table space.
- A CREATE TABLE statement is not allowed if there are any pending definition changes for the table space
- A DROP INDEX statement is not allowed if the index is an unique index defined on a ROWID column that is defined as GENERATED BY DEFAULT and, there are any pending definition changes for the table space or any objects within the table space that is explicitly created
- A DROP INDEX statement is not allowed if the index is an empty index on an auxiliary table residing in an explicitly-created LOB table space and there are any pending definition changes for the base table space or any objects within the base table space.
- A DROP TABLE statement is not allowed if the table space was explicitly created and there are any pending definition changes for the table space.
- A DROP TABLE statement is not allowed if the table is an empty auxiliary table and there are any pending definition changes for the base table space or any objects within the base table space.
- The following DDL statements are not allowed if the name of the object to be defined or the target of a RENAME STATEMENT is identical to the existing name of an object of the same object type stored in the catalog table SYSPENDINGOBJECTS:
  - ALTER TABLE ADD CLONE
  - CREATE ALIAS
  - CREATE AUXILIARY TABLE
  - CREATE INDEX
  - CREATE GLOBAL TEMPORARY TABLE
  - CREATE SYNONYM
  - CREATE TABLE
  - CREATE VIEW
  - RENAME INDEX
  - RENAME TABLE

4.1.7 UTS ALTER for MEMBER CLUSTER

MEMBER CLUSTER is a table space option that allows reduced contention in a heavy INSERT application in data sharing environment (see 5.4, “Universal table space support for MEMBER CLUSTER”). DB2 9 does not support MEMBER CLUSTER for a UTS.

DB2 10 allows MEMBER CLUSTER for both partition-by-growth UTS and range-partitioned UTS. You can ALTER an existing UTS to add the MEMBER CLUSTER option. For example, the following statement adds a row to SYSIBM.SYSPENDINGDDL and places the table space in AREOR advisory state if it is not already in that state:

```
ALTER TABLESPACE ITSODB.TS2 MEMBER CLUSTER YES;
```

As for all the other online schema changes, which we described in this section, this ALTER TABLESPACE option is considered a pending change. Therefore, you must run REORG TABLESPACE SHRLEVEL CHANGE or REFERENCE to materialize this change.

4.1.8 Utilities support for online schema changes

Utilities are enhanced to support objects with pending definition changes. In this section, we describe this support and list a few considerations for dealing with pending definition changes.

RECOVER TABLESPACE and RECOVER INDEX

The RECOVER TABLESPACE and RECOVER INDEX utilities do not have new parameters for pending definition changes, but there are implications in executing them after the changes are materialized.

After you execute an ALTER command that creates a pending definition change, nothing has really happened to the object. There are no rows in the SYSIBM.SYSCOPY table to indicate that pending changes are waiting for materialization. As a consequence, the two image copies FC1, and FC2 shown in Figure 4-16 are considered good image copies for potential recoveries, such as a RECOVER to currency.

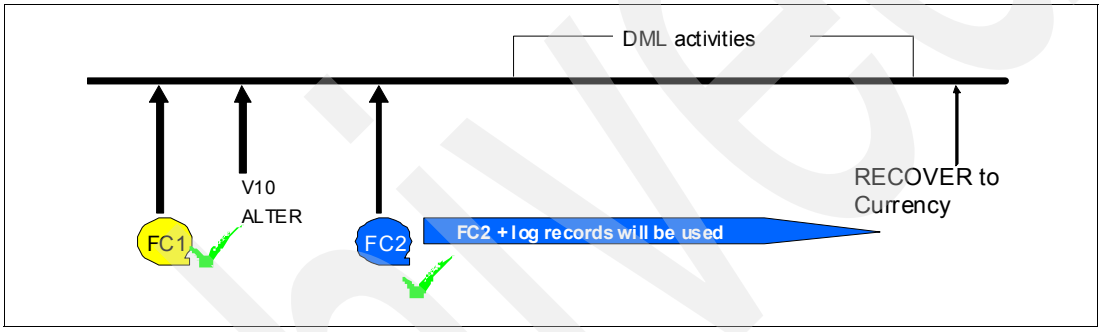


Figure 4-16 Recover after alter before materialization

The situation shown in Figure 4-17 is different. Now, the pending definition changes are materialized. The current page set is different from what it was before, for example because of another page size or another segment size, and the information in the catalog and directory is updated and adjusted to the new definition. The page set that is represented by image copy FC2 has a different structure. Thus, DB2 cannot use any of the image copies that were created before the REORG materialized the change.

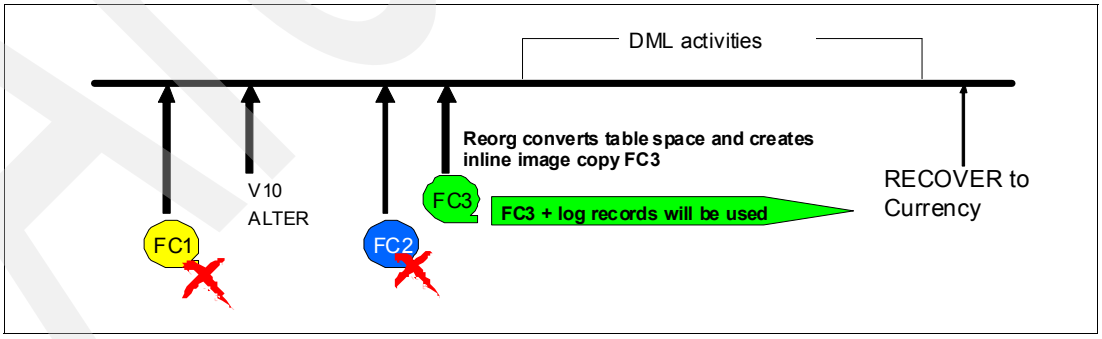


Figure 4-17 Recover to current after materialization of pending definition changes

This situation is also true for a point-in-time recovery. You can recover only to a point in time that is later than or equal to the inline image copy that was taken as part of the reorganization that did the materialization of the pending changes.

At “REPORT RECOVERY” on page 94, we show the enhanced job of the REPORT RECOVERY utility.

As stated, only SHRLEVEL REFERENCE or SHRLEVEL CHANGE are allowed on the REORG TABLESPACE statement. Both options generate inline image copies. You do have one image copy available as recovery base after the schema change. It is in fact the only image copy that RECOVER TABLESPACE can use if needed later. You might want to think about forcing dual copies during the RECOVER TABLESPACE run. The same considerations are true for copy-enabled indexes.

REPORT RECOVERY

The REPORT RECOVERY utility is changed to help to clearly identify SYSIBM.SYSCOPY entries that cannot be considered as recovery base in RECOVER utility runs after materialization of pending definition changes.

The REPORT RECOVERY utility marks the following entries listed from SYSIBM.SYSCOPY to help you identify the type of entry and for which action it can be used as recovery base:

( ) Image copy prior to LOG(NO) event.

<> Image copy prior to rebalancing of table space partitions

* * Non-image copy SYSCOPY entry

(new with DB2 10) SYSCOPY entry created before any alteration was materialized.

The pound signs (#) are the new entries. Any entry in SYSCOPY that uses the pound signs around the image copy type cannot be used by DB2 for RECOVER.

Example 4-6 shows a sample output with eight formatted SYSCOPY entries for table space DB1.TS1. The sort order is from oldest to youngest. Refer to the IC TYPE. The first six entries use pound signs around the letter, which identifies the type of SYSCOPY entry. So, no matter the type of entry, entries with pound signs cannot be used for RECOVER.

Example 4-6 REPORT RECOVERY job output

```
1DSNU000I    215 20:08:17.73 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   215 20:08:17.76 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    215 20:08:17.76 DSNUGUTC -  REPORT RECOVERY TABLESPACE DB1.TS1
 DSNU581I  -DBOB 215 20:08:17.76 DSNUPREC - REPORT RECOVERY TABLESPACE DB1.TS1
 DSNU593I  -DBOB 215 20:08:17.77 DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
             MINIMUM   RBA: 000000000000
             MAXIMUM   RBA: FFFFFFFFFFFF
             MIGRATING RBA: 000000000000
 DSNU582I  -DBOB 215 20:08:17.77 DSNUPPCP - REPORT RECOVERY TABLESPACE DB1.TS1 SYSCOPY ROWS
                          AND SYSTEM LEVEL BACKUPS
 TIMESTAMP = 2010-08-03-15.40.49.663528,  IC TYPE = #C#,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =000008C27CD3
 DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = L,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
 LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED = Y,   TTYPE =
 JOBNAME   =         ,     AUTHID =           ,     COPYPAGESF = -1.0E+00
 NPAGESF   = -1.0E+00            ,            CPAGESF = -1.0E+00
 DSNAME    = DB1.TS1                                    , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M

 TIMESTAMP = 2010-08-03-15.44.45.997506,  IC TYPE = #X#,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =000008C4098C
 DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
 LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED = Y,   TTYPE =
 JOBNAME   = DB2R8L  ,     AUTHID =   DB2R8   ,     COPYPAGESF = -1.0E+00
 NPAGESF   = -1.0E+00            ,            CPAGESF = -1.0E+00
 DSNAME    = DB1.TS1                                    , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M

 TIMESTAMP = 2010-08-03-20.07.20.813333,  IC TYPE = #F#,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =0000090539AE
 DEV TYPE  =         ,                    IC BACK = FC,   STYPE   = T,  FILE SEQ = 0000,  PIT LRSN  = 0000090539E2
 LOW DSNUM = 0001,   HIGH DSNUM = 0001,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED = Y,   TTYPE = C
 JOBNAME   = DB2R8L  ,     AUTHID =   DB2R8   ,     COPYPAGESF = -1.0E+00
 NPAGESF   = -1.0E+00            ,            CPAGESF = -1.0E+00
 DSNAME    = DBOBI.DB1.TS1.N00001.CXZO6WVC               , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M
```

```
TIMESTAMP = 2010-08-03-20.08.04.491833,  IC TYPE = #W#,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =0000090600C4
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED = Y,   TTYPE =  F
JOBNAME   = DB2R8L  ,     AUTHID =   DB2R8   ,     COPYPAGESF = -1.0E+00
NPAGESF   = -1.0E+00            ,                 CPAGESF = -1.0E+00
DSNAME    = DB1.TS1                                      , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M

TIMESTAMP = 2010-08-03-20.08.04.482351,  IC TYPE = #A#,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =000009088F3A
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = F,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED =  ,   TTYPE = 0000004K
JOBNAME   =         ,     AUTHID =           ,     COPYPAGESF = -1.0E+00
NPAGESF   = -1.0E+00            ,                 CPAGESF = -1.0E+00
DSNAME    = DB1.TS1                                      , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M

TIMESTAMP = 2010-08-03-20.08.04.483092,  IC TYPE = #A#,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =000009088F3A
DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = F,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED =  ,   TTYPE = 0000008K
JOBNAME   =         ,     AUTHID =           ,     COPYPAGESF = -1.0E+00
NPAGESF   = -1.0E+00            ,                 CPAGESF = -1.0E+00
DSNAME    = DB1.TS1                                      , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M

TIMESTAMP = 2010-08-03-20.08.04.522143,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =00000908C0D8
DEV TYPE  = 3390    ,                    IC BACK =   ,   STYPE   = W,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
LOW DSNUM = 0001,   HIGH DSNUM = 0001,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED = Y,   TTYPE =
JOBNAME   = DB2R8L  ,     AUTHID =   DB2R8   ,     COPYPAGESF =  2.0E+00
NPAGESF   =  2.0E+00            ,                 CPAGESF =  2.0E+00
DSNAME    = DB2R8.DB1.TS1.FC1                            , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M

TIMESTAMP = 2010-08-03-20.08.05.013301,  IC TYPE =  F ,  SHR LVL =  ,  DSNUM    = 0001,  START LRSN =00000908C0D8
DEV TYPE  =         ,                    IC BACK = FC,   STYPE   = T,  FILE SEQ = 0000,  PIT LRSN  = 00000908C515
LOW DSNUM = 0001,   HIGH DSNUM = 0001,   OLDEST VERSION = 0000,   LOGICAL PART = 0001,   LOGGED = Y,   TTYPE = W
JOBNAME   = DB2R8L  ,     AUTHID =   DB2R8   ,     COPYPAGESF = -1.0E+00
NPAGESF   = -1.0E+00            ,                 CPAGESF = -1.0E+00
DSNAME    = DBOBI.DB1.TS1.NOOOO1.CXZO7W5W                , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M
```

UNLOAD

Although the RECOVER utility cannot use the image copy data sets marked with pound signs in Example 4-6, you can use those data sets as input for the UNLOAD utility. Thus, you can at least bring the data back to that point in time prior to the REORG TABLESPACE utility execution that materialized your pending changes.

REPAIR

You can use the REPAIR utility with the SET NOAREORPEND option to remove the advisory state from you table or index space. As for any situation in which you choose to use the SET option or any other option of the REPAIR utility, this action might result in unwanted consequences.

Figure 4-18 shows a scenario that warns not to use the REPAIR ... SET NOAREORPEND without considering the whole scenario. The figure describes the following situation:

1. You have a table space and an associated image copy coming from regular copy cycles. For some reason, you need to make one or more changes, which place your table space into AREOR state.
2. You also know that if a table space is placed into AREOR state, one or more rows are added to the SYSIBM.SYSPENDINGDDL table. The next copy cycle now produces a second nice copy of your table space. Immediately after this second copy, you create a quiesce point, because it might happen that you have to recover to this point later.
3. So far in this process, nothing has happened and everything is OK. However, someone notices that this table space is in AREOR state and wants to change that state. Instead of

using ALTER TABLESPACE ... DROP PENDING CHANGES, this person decides to remove the AREOR state using the REPAIR TABLESPACE ... SET NOAREORPEND from the table space.

4. Let us assume that some time after you remove the AREOR state from the table space, automation begins and initiates a nightly reorganization.
5. Because the pending changes are still available in SYSIBM.SYSPENDINGDDL, this reorganization materializes the pending changes, and DB2 takes the necessary actions, including the invalidation of all image copies that exist for this table space and that are younger than the inline copy created during the reorganization.
6. At this point, you do not recognize any problems with this process. Later, however, if you have to go back to the quiesce point that you created earlier, you identify the record in SYSIBM.SYSCOPY with ICTYPE 'Q' and submit the following RECOVER:

   ```
   RECOVER TABLESPACE .... TORBA x'...Q...'.
   ```

7. Now, the problem becomes quite obvious. Image copies FC1 and FC2 in Figure 4-18 cannot be used. If you run REPORT RECOVERY utility for the table space, you notice the pound signs (#) around the image copy type information. This indicates that the image copies cannot be used for recover.

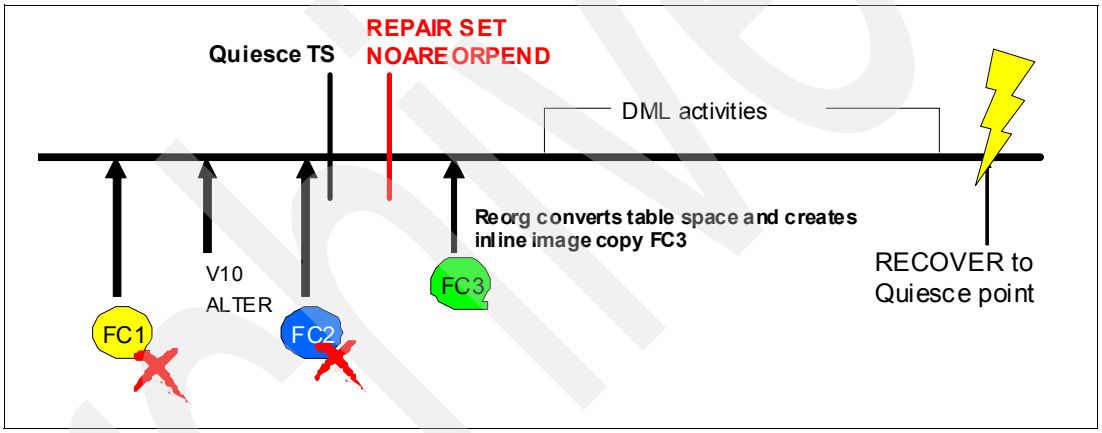


Figure 4-18 Effects of REPAIR SET NOAREORPEND

In this scenario, using REPAIR with the SET NOAREORPEND option is not recommended. If you want to remove the AREOR state, make sure that the pending changes are removed first from SYSIBM.SYSPENDINGDDL. The pending changes are removed when you use ALTER TABLESPACE ... DROP PENDING DDL.

Attention: Always use REPAIR with guidance, as noted in *DB2 10 for z/OS Utility Guide and Reference,* SC19-2984:

"Be very careful when using REPAIR. Improper use can damage the data even further."

4.2 Autonomic checkpoint

DB2 9 for z/OS allows system checkpoints to be scheduled based on either the number of log records created or at a fixed time interval. Both methods have advantages and disadvantages. The advantage for the time based approach is that a system checkpoint is definitely taken at the chosen interval. The disadvantage is that checkpoints might be taken

for low activity periods or the checkpoints might be too far apart during busy periods, which can increase restart time in the event of a failure.

The advantage for checkpoint writing based on the number of log records is, that in case of low workload the checkpoints will be automatically written at larger intervals. Using the number of log records also provides a relatively consistent recovery time. However, at peak workload, the number of log records written between two checkpoints can be much higher than is desirable.

It is desirable to have checkpoints scheduled based on the number of log records when a DB2 subsystem is busy and if a system becomes relatively inactive. Checkpoints should be based on a fixed time minimum interval.

With DB2 10, you can use both values as threshold. The following DSNZPARMs are introduced to control the behavior:

- CHKTYPE (Checkpoint Type)

SINGLE	Either minutes or number of log records is used to determine when checkpoint is taken
BOTH	The new functionality is used. minutes and the number of log records count toward the initiation of the next checkpoint.

- CHKFREQ (Checkpoint Frequency)

1- 60	If CHKTYPE = SINGLE, the checkpoint frequency is set based on the number of set minutes
1000 - 16000000	If CHKTYPE = SINGLE, the checkpoint frequency is set based on the number of log records.
NOTUSED	If CHKTYPE = BOTH, you must specify NOTUSED here.

- CHKLOGR (Records /Checkpoint)

  The RECORDS/CHECKPOINT field determines the number of log records that are created between log checkpoints.

NOTUSED	MUST be specified if CHKTYPE = SINGLE
1000 - 16000000	IF CHECKTYPE is SINGLE, the checkpoint frequency expressed in the number of log records. 1000 - 16000000 is the allowed range.
1 000 - 99999999	IF CHECKTYPE is BOTH, the checkpoint frequency expressed in the number of log records. 1000 - 99999999 is the allowed range.

- CHKMINS (Minutes /Checkpoint)

  The MINUTES/CHECKPOINT field determines the number of minutes between log checkpoints.

NOTUSED	MUST be specified if CHKTYPE = SINGLE
1 - 60	IF CHECKTYPE is SINGLE, the checkpoint frequency expressed in the number of minutes.
1- 1439	IF CHECKTYPE is BOTH, the checkpoint frequency is expressed in the number of minutes.

You can check the current setting with DB2 command -DIS LOG.

Archived

Figure 4-19 is a sample output from our system where we use CHECKTYPE = BOTH and want a checkpoint after 5 minutes or 10000000 log records.

```
DSNJ370I  -DBOB DSNJC00A LOG DISPLAY
CURRENT COPY1 LOG = DBOBL.LOGCOPY1.DS02 IS 13% FULL
CURRENT COPY2 LOG = DBOBL.LOGCOPY2.DS02 IS 13% FULL
          H/W RBA = 000004A8381C
          H/O RBA = 00000464FFFF
          FULL LOGS TO OFFLOAD = 0 OF 6
          OFFLOAD TASK IS (AVAILABLE)
DSNJ371I  -DBOB DB2 RESTARTED 19:52:26 JUL 27, 2010
          RESTART RBA 0000045C5000
          CHECKPOINT FREQUENCY 5 MINUTES OR 10000000 LOGRECORDS
          LAST SYSTEM CHECKPOINT TAKEN 19:51:49 JUL 28, 2010
DSN9022I  -DBOB DSNJC001 '-DIS LOG' NORMAL COMPLETION
```

Figure 4-19 -DIS LOG OUTPUT for CHECKTYPE=BOTH

To support this change, the DB2 command -SET LOG was also changed.

If you want to change from CHECKTYPE = BOTH back to CHECKTYPE =SINGLE, and to set the interval time between two checkpoints to 2 minutes, you can use the command

```
-SET LOG SINGLE CHKTIME(2)
```

The full syntax diagram is shown in Figure 4-20.

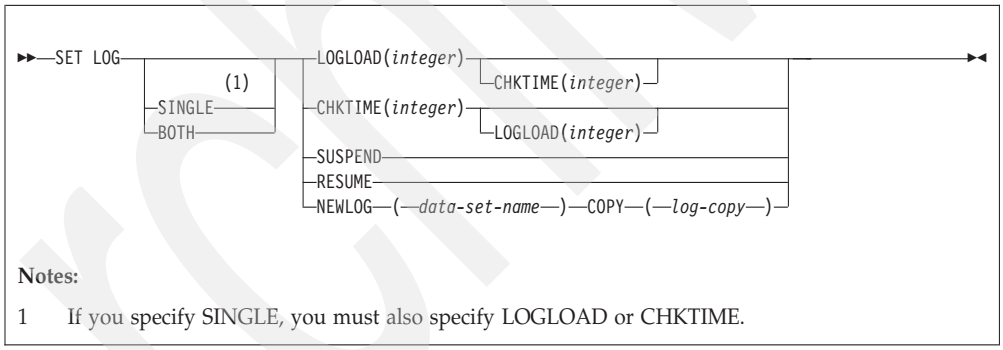


Figure 4-20 -SET LOG syntax

4.3 Dynamically adding an active log data set

With DB2 9, you add a new active log data set to the log inventory using the stand-alone utility DSNJU003 (Change Log Map utility). DSNJU003 is the only utility that requires you to stop DB2 to be able to run the utility. Thus, you have to stop DB2 to add a new active log data set to the bootstrap data set.

If DB2 has a problem with archiving when it runs out of active logs, an availability issue occurs. One problem is that you have to stop DB2. Another problem is that, if you need to add a new active log data set, you cannot properly stop DB2, because as part of the stop process DB2 needs to write at least a few last log records.

DB2 10 in conversion mode solves this kind of issue. The SET LOG command is enhanced to let you dynamically add active log data sets. This feature allows you to give the system some

relief with additional active log space until you can correct the original archiving problem that is causing DB2 to hang.

Figure 4-20 shows the syntax for the -SET LOG NEWLOG keyword. With the new command, the procedure to add a new active log to the log inventory and to have DB2 switch to this new log is as follows:

1. Define the VSAM clusters for the new active log data sets.

   Installation job DSNTIJIN contains the sample JCL and IDCAMS DEFINE CLUSTER statements. In our environment, we used the job listed in Example 4-7.

Example 4-7 Define clusters for the new active log data sets

```
//DSNTIC  PROC
//* ******************************************************************
//* DIRECTORY/CATALOG AMS INVOCATION INSTREAM JCL PROCEDURE
//* ******************************************************************
//DSNTIC  EXEC PGM=IDCAMS,COND=(2,LT)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNTIC  PEND
//*
//DSNTDBL EXEC DSNTIC
//SYSIN    DD  *
DEFINE CLUSTER -
         ( NAME (DBOBL.LOGCOPY1.DS04) -
           KILOBYTES(34560)           -
           LINEAR )                     -
      DATA                            -
         ( NAME (DBOBL.LOGCOPY1.DS04.DATA)  -
                  )    CATALOG(UCAT.DBOBLOGS)

   DEFINE CLUSTER -
         ( NAME (DBOBL.LOGCOPY2.DS04) -
           KILOBYTES(34560)           -
           LINEAR )                     -
      DATA                            -
         ( NAME (DBOBL.LOGCOPY2.DS04.DATA)  -
              )    CATALOG(UCAT.DBOBLOGS)
```

2. Run stand-alone utility DSN1LOGF to format the new active log data sets. This step is not necessary for things to work, but recommended.
3. Execute the -SET LOG command. Make sure you execute it for each log data set you want to add and remember that you must execute it once per log copy. We used:

   ```
   -db0b set log newlog(db0bl.logcopy1.ds04)copy(1)
   -db0b set log newlog(db0bl.logcopy2.ds04)copy(2)
   ```

   Example 4-8 shows the -DIS LOG command output.

Example 4-8 -DIS LOG command output

```
DSNJ370I  -DBOB DSNJC00A LOG DISPLAY 736
CURRENT COPY1 LOG = DBOBL.LOGCOPY1.DS02 IS 20% FULL
CURRENT COPY2 LOG = DBOBL.LOGCOPY2.DS02 IS 20% FULL
          H/W RBA = 000004CEF6C8
          H/O RBA = 00000464FFFF
          FULL LOGS TO OFFLOAD = 0 OF 8
```

Archived

```
          OFFLOAD TASK IS (AVAILABLE)
DSNJ371I  -DBOB DB2 RESTARTED 19:52:26 JUL 27, 2010 737
          RESTART RBA 0000045C5000
          CHECKPOINT FREQUENCY 5 MINUTES OR 10000000 LOGRECORDS
          LAST SYSTEM CHECKPOINT TAKEN 15:56:49 JUL 29, 2010
DSN9022I  -DBOB DSNJC001 '-DIS LOG' NORMAL COMPLETION
```

As a result of the -SET LOG command, the information about the existence of the new active log data set or sets is now stored permanently in the BSDS as though you used DSNJU003.

4. Restart the offload. If you are in a situation where DB2 hangs still trying to archive the data sets that already existed before you added new active log data sets, you let DB2 find the new active log data sets by issuing the following command:

   ```
   -ARCHIVE LOG CANCEL OFFLOAD
   ```

   This command cancels any offloading currently in progress and restarts the off-load process beginning with the oldest active log data set that has not been offloaded and proceeding through all active log data sets that need offloading. Any suspended offload operations are restarted.

4.4 Preemptible backout

When DB2 9 executes a ROLLBACK or ABORT, either due to an explicit request or abnormal termination of a thread, a service request block (SRB) is scheduled to do the backout processing. By default, SRBs that are not part of an enclave are non-preemptible. They can be interrupted, but they are dispatched immediately after the interrupt is serviced.

On a system with a single processor, it might seem that the system is hung or stuck in a loop or responding slowly because the backout operation can take a long time. Furthermore, if you cancel or restart DB2, the backout operation resumes eventually, again making it look as though the system was hung. Generally, if there are n threads being backed out on a n-way system, it will look like the system is hung.

With DB2 10, the SRB that performs the backout operation is scheduled to an enclave so that it can be preempted by higher priority work. The n-thread n-way scenario can cause CPU starvation for lower priority work, but the system is still responsive to operator commands and higher priority work.

4.5 Support for rotating partitions

DB2 helps you drop existing partitions.

DB2 V8 and 9 provide the ALTER TABLE ROTATE option. This option allows you to specify FIRST to LAST ENDING AT. As a result of this statement, DB2 wipes out the information of the *first logical partition* and makes this the new *last logical partition*. The partition range for the new last logical partition must be higher than the old last logical partition if the limit keys are ascending and must be lower if the limit keys are descending.

DB2 10 provides more flexibility. You can choose any partition (not just the first) and rotate it to the logical end of the table space. The same rules apply for the limit keys that you specify in the ENDING AT part of the statement.

The left side of Figure 4-21 shows the original partitioned table space with five partitions. The partitioning key is a date column, and the limit keys are used in an ascending order. DB2 V8 introduced the concept of logical and physical partition numbers. The physical partition number is the number stored in the PARTITION column in SYSIBM.SYSTABLEPART, and the logical partition number is in the LOGICAL_PART column in the same table space. When you create a partitioned table space, if you did not run ROTATE statements, both numbers are the same, as shown in Figure 4-21. The numbers will change when you use ROTATE.

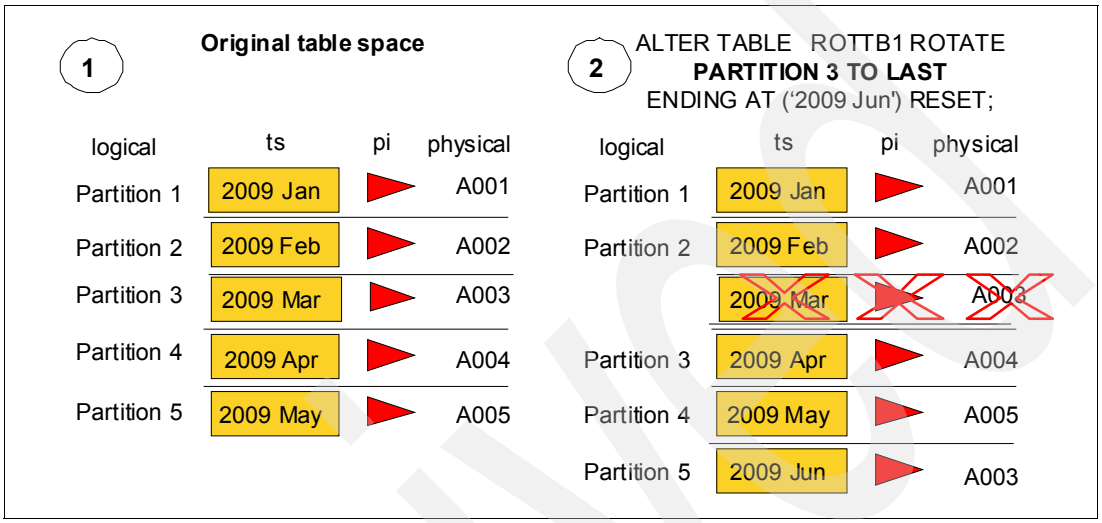


Figure 4-21 ROTATE 3 TO LAST sample

In our example, you want to replace all the data that is currently in Partition 3 (logical and physical still the same) with new data. If you prefer to the used ALTER statement, you can identify Partition 3 as the partition that is being rotated, and as a result it is emptied and goes to the logical end of the table space. The result of this ALTER is that the now empty partition becomes the logical Partition 5 and the physical Partition 3. Note that the ENDING AT date of June 2009 is higher than the ending date for the old logical Partition 5, which was May 2009.

Let us go one step further and assume that now you want to remove all the information for April 2009. April 2009 data is stored in logical Partition 3 and physical Partition 4.

Note: You must specify the physical partition number on your ALTER TABLE ... ROTATE statement.

If you do not have Figure 4-21 in front of you, especially after a few rotates, you have to find the partition number that you must specify now for the ALTER statement querying the catalog table SYSIBM.SYSTABLEPART. Figure 4-22 shows a sample result.

```
 SELECT LIMITKEY, PARTITION, LOGICAL_PART,A.*
  FROM SYSIBM.SYSTABLEPART A WHERE DBNAME = 'ROTATE'
  ORDER BY LIMITKEY       ;
---------+---------+---------+---------
LIMITKEY      PARTITION  LOGICAL_PART
---------+---------+---------+---------
2009-01-31            1             1
2009-02-28            2             2
2009-04-30            4             3
2009-05-31            5             4
2009-06-30            3             5
```

Figure 4-22 SELECT to identify right physical partition for ROTATE

To initiate the rotation, you specify the following ALTER statement:

```
ALTER TABLE   ROTTB1 ROTATE PARTITION 4 TO LAST ENDING AT ('2009 Jul') RESET;
```

Figure 4-23 shows the result of this second ROTATE. The April 2009 data is no longer available in the table space. The new ending limit key is July 2009.

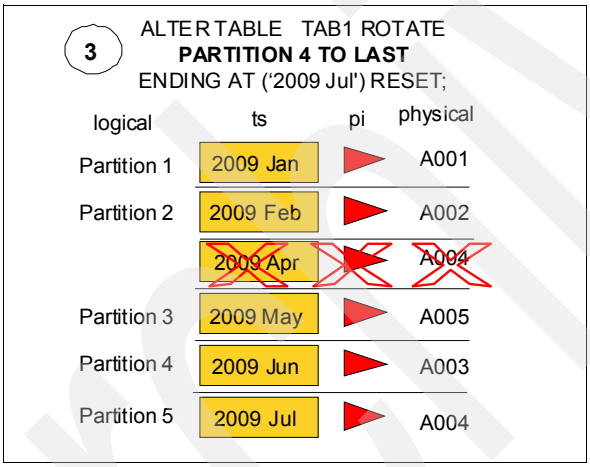


Figure 4-23 ROTATE 4 TO LAST

DB2 must know which physical partition is logically on which position. This logical partition number is for example used when you ask for a -DIS DB(...) SPACE(...) output. If you look at Figure 4-24, at first sight the order the partitions that display seem to be completely unsorted. However, if you compare Figure 4-24 with Figure 4-23, you can see that the -DIS DB command output lists the table space partitions following the sort order of the logical partitions.

```
NAME     TYPE PART  STATUS
-------- ---- ----- -------
TS1      TS    0001 RW
    -THRU      0002
TS1      TS    0005 RW
TS1      TS    0003 RW
    -THRU      0004
TS1      TS         RW
```

Figure 4-24 -DIS DB after two ROTATE table space executions

Remember that we wiped out the data for months March 2009 and April 2009. Figure 4-25 shows the relationship between limit keys and physical and logical partitions after the second rotate. If data was inserted into the table now that has a date of March 2009 or April 2009, that data would all go into physical Partition 5.

This function is not bad, but you want to keep this in mind in case if you plan to use the ROTATE function in DB2 10 NFM.

```
LIMITKEY      PARTITION  LOGICAL_PART
---------+---------+---------+--------
2009-01-31            1             1
2009-02-28            2             2
2009-05-31            5             3
2009-06-30            3             4
2009-07-31            4             5
```

Figure 4-25 LIMITKEY, PARTITION, and LOGICAL_PART after second rotate

DB2 does not allow you to wipe out the last logical partition. You get the error message shown in Figure 4-26. The message does not say clearly that the reason for the failure of the ALTER TABLE statement is the fact that you tried to replace the last logical partition with the same physical partition with which it is currently associated. However, this is the reason for this negative SQL code.

```
DSNT408I SQLCODE = -20183, ERROR:  THE PARTITIONED, ADD PARTITION, ADD
         PARTITIONING KEY, ALTER PARTITION, ROTATE PARTITION, OR PARTITION BY
         RANGE CLAUSE SPECIFIED ON CREATE OR ALTER FOR DB2R8.ROTTB1 IS NOT
         VALID
```

Figure 4-26 Error message you get when trying to rotate the last partition

4.6 Compress on insert

Starting with DB2 V3, data compression has been widely used by DB2 for z/OS customers. In order to manage the compression, DB2 needs a dictionary for every partition and, up to DB2 V7, compression dictionaries were allocated below the bar. Therefore, virtual storage constrained compression to large partitioned table spaces. With DB2 V8, the compression dictionaries are allocated above the bar and, therefore, do not cause storage constraints. As a consequence, you might want to turn on compression for additional table spaces.

Prior to DB2 10, if you turn on compression for a table space using the ALTER TABLESPACE command, DB2 needs to build the compression dictionary. Compression dictionaries are built as part of REORG TABLESPACE or LOAD utility runs. You might not be able to run LOAD or REORG when you decide to turn on compression for a given table space.

With DB2 10 NFM, you can turn on compression with ALTER any time, and the compression dictionary is built when you execute the following statements:

- INSERT statements
- MERGE statements
- LOAD SHRLEVEL CHANGE

Additionally, when you LOAD XML data, a dictionary can be built specifically for the XML table space so that the XML data is compressed in the following circumstances:

- The table space or partition is defined with COMPRESS YES
- The table space or partition has no compression dictionary built yet
- The amount of data in the table space is large enough to build the compression dictionary

The threshold is about 1.2 MB, which is determined by reading the RTS statistics in memory. When the threshold is reached, DB2 builds the dictionary asynchronously and issues the message shown in Figure 4-27.

```
DSNU241I  -DBOB DSNUZLCR - DICTIONARY WITH 4096  755
ENTRIES HAS BEEN SUCCESSFULLY BUILT FROM 598 ROWS FOR TABLE SPACE
SABI6.TS6, PARTITION 1
```

Figure 4-27 Message DSNU241I compression dictionary build

The DSNU241I message is issued in this case because the table space is partitioned. DSNU231I is written out in case of a non-partitioned table space. These messages are issued on the console only by compress on insert. The LOAD and REORG utilities have the same messages, but these messages are written in the utility output, not on the console.

After the dictionary is built, DB2 inserts the data in compressed format. At least 1.2 MB of data in the table space is not compressed. The additional data is compressed.

Data rows that you insert while the dictionary is still under construction are inserted uncompressed. When building the dictionary asynchronously, DB2 reads the existing data with isolation level uncommitted read.

Inserting a dictionary: If the table space is defined as COMPRESS YES and you insert rows, then a dictionary is built. Use COMPRESS NO if you want to insert, but not build, a dictionary. If you ALTER to COMPRESS YES and use LOAD REPLACE or REORG, then you can have the utilities build the dictionary.

If the compression dictionary is built using LOAD REPLACE or REORG TABLESPACE, the dictionary pages follow the system pages (header and space map), which is not the case when the compression dictionary is built on-the-fly. Because there must be at least 1.2 MB worth of data in the table space before the new compression functionality kicks in, the compression dictionary gets any page numbers. Furthermore, the dictionary pages might not be contiguous, as illustrated in Figure 4-28.

The left side of this figure shows the dictionary pages (DI) in the table space. When the table space is image copied with option SYSTEMPAGES YES[3] the dictionary pages are replicated after the header and space map pages.

[3] The SYSTEMPAGES YES option does not apply to FlashCopy image copies.

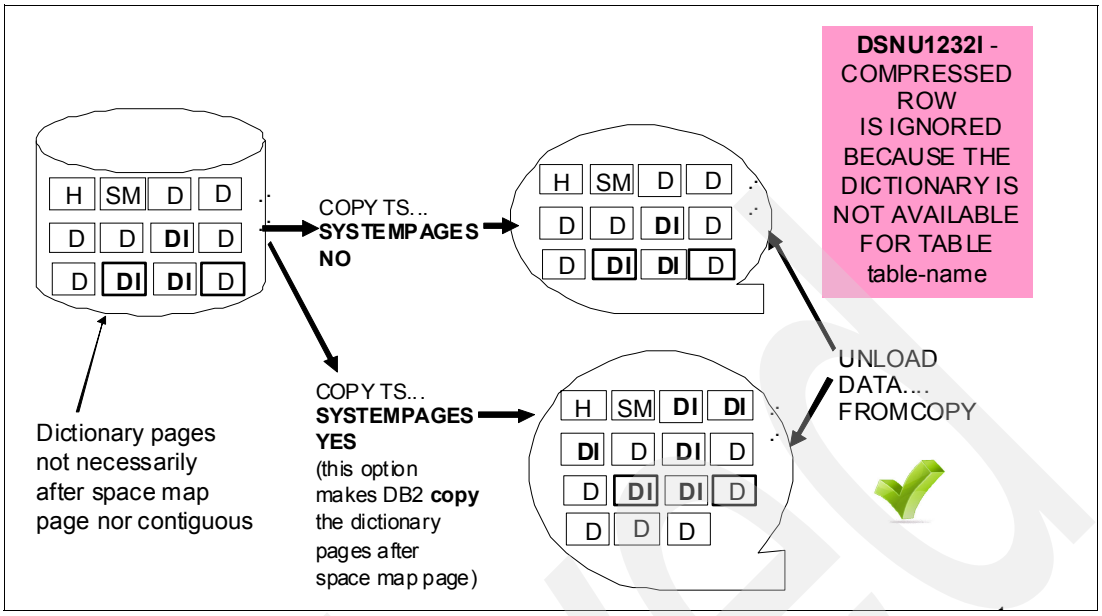


Figure 4-28 Compression dictionary pages spread over table space

If you use image copies as input for the UNLOAD utility and if you plan on using this automatic compression without REORG or LOAD, you must run the COPY utilities with option SYSTEMPAGES YES. This option requests the collection of all SYSTEMPAGES during utility execution and places a copy of those directly behind the first space map page. The original dictionary pages remain at their original location. So, the SYSTEMPAGES YES option on COPY TABLESPACE produces duplicates of those system pages.

If you specify SYSTEMPAGES NO, DB2 copies only the table space as is. When you attempt to unload from this image copy, the UNLOAD utility recognizes that the rows in the table space are compressed but cannot uncompress the rows because it looks only for dictionary pages at the beginning of the table space prior to first data page.

If you run REORG later, depending on your choice regarding the KEEPDICTIONARY utility control keyword, REORG either moves the dictionary pages behind the space map page and before the first data page or creates a new dictionary, which is then located at this point.

4.6.1 DSN1COMP considerations

Generally, you can use the DSN1COMP utility, which is available since DB2 V3, to estimate the space savings that can be achieved by DB2 data compression in table spaces and indexes.

If you run the DSN1COMP utility without any special option, the utility calculates the estimated space savings based on the algorithms that are used for building compression during LOAD. If you use the LOAD utility to build a new compression dictionary, the compression ratio is likely to be a bit less effective than during REORG. If you specify the REORG keyword on the compression utility, DB2 calculates the compression ratio based on what will be accomplished by the REORG utility. There is no specific keyword for the COMPRESS on INSERT method. Compress on INSERT reaches similar compression ratios as the LOAD utility for table spaces, considerably larger than 1.2 MB. As a consequence, if you do not specify REORG for your estimate, the calculated savings is about the same as COMPRESS ON INSERT.

4.6.2 Checking whether the data in a table space is compressed

Let us assume that you have turned on compression for your table space and now you want to know whether COMPRESS on INSERT actually worked and if the data is now compressed. The easiest way to verify whether the table space is compressed is to check the SYSLOG for the following message:

```
DSNU241I  -DBOB DSNUZLCR - DICTIONARY WITH 4096  755
ENTRIES HAS BEEN SUCCESSFULLY BUILT FROM 598 ROWS FOR TABLE SPACE
SABI6.TS6, PARTITION 1
```

If you can find this message, you know that the dictionary was built for a given partition of a table space. For a non-partitioned table space, the message is DSNU231I.

If you cannot find this message in the SYSLOG, check whether the data volume is large enough so that the threshold of 1.2 MB of data is passed and compression can theoretically begin. You can for example use the following SQL query to check the RTS tables:

```
SELECT DATASIZE
FROM SYSIBM.SYSTABLESPACESTATS
WHERE DBNAME='yourdb'
AND NAME='yourts';
```

Remember that column DATASIZE contains the amount of data stored in your table space in bytes, that is you need at least 1,310,720 bytes. However, the time that the current value is reflected in the catalog depends on the interval specified for externalizing RTS statistics (STATSINT DSNZPARM default is 30 minutes).

4.6.3 Data is not compressed

In some cases, it might appear that you turn on compression and that the data volume is larger than 1.2 MB but that COMPRESS on INSERT did not take place. This situation can occur if DB2 cannot build a usable compression dictionary because of the data contents. If this is the case, you do not get an error message. An error message, such as DSNU235I or DSNU245I, displays on the console if there are issues, such as out of space conditions or similar.

4.7 Long-running reader warning message

Although long-running readers cause fewer performance issues than long-running writers, quite often long-running readers also cause problems. One common problem is the SWITCH phase of online reorg, because long-running readers do not allow REORG to break in and gain exclusive control.

DB2 V8 introduced DSNZPARM LRDRTHLD, which provides the opportunity to proactively identify long-running readers. You can specify the number of minutes a reader is allowed to execute without a commit before IFCID 313 record is cut. The problem with this solution is that no DB2 message is produced when the specified number of minutes is exceeded. You have to collect and edit a record trace to get a report about the existence of long-running readers.

In addition, the default value for DSNZPARM LRDRTHLD is zero for DB2 versions 8 and 9. As a consequence many DB2 users have not activated this functionality.

DB2 10 addresses these issues. First, the new default for DSNZPARM LRDRTHLD is 10 minutes. This new default is also applied during release migration. So in case you do not want to turn on this functionality, you must manually overwrite it with zero. Secondly besides the IFID 313, which is still cut when your applications exceed the threshold, DB2 generates the new message DSNB260I shown in Example 4-9. Note that the message is not only popping up with each new threshold being reached, but also each new message tells you for how long in total the reader is already holding resources. In Example 4-9, the first message displays after 10 minutes, and then the second message displays after 20 minutes.

Example 4-9 Message for LRDRTHLD threshold exceeded

```
DSNB260I  -DBOB DSNB1PCK WARNING - A READER HAS BEEN  424
RUNNING FOR 10 MINUTES
           CORRELATION NAME=DB2R8
           CONNECTION ID=TSO
           LUWID=USIBMSC.SCPDBOB.C6615CBED23B=160
           PLAN NAME=DSNESPCS
           AUTHID=DB2R8
           END USER ID=*
           TRANSACTION NAME=*
           WORKSTATION NAME=*

DSNB260I  -DBOB DSNB1PCK WARNING - A READER HAS BEEN  430
RUNNING FOR 20 MINUTES
           CORRELATION NAME=DB2R8
           CONNECTION ID=TSO
           LUWID=USIBMSC.SCPDBOB.C6615CBED23B=160
           PLAN NAME=DSNESPCS
           AUTHID=DB2R8
           END USER ID=*
           TRANSACTION NAME=*
           WORKSTATION NAME=*
```

Depending on how well your applications behaves on your various DB2 systems, the current default setting of 10 minutes can generate a large number or messages. After migrating to DB2 10 conversion mode (CM), check the MSTR address space and change the value to a higher number to prevent the system spool from being flooded by these messages.

Long-running reader warning messages are available starting with DB2 10 CM.

4.8 Online REORG enhancements

DB2 10 improves the usability and performance of online reorganization in several ways.

This release of DB2 for z/OS supports the reorganization of disjoint partition ranges of a partitioned table space, and improves SWITCH phase performance and diagnostics. It also, removes restrictions that are related to the online reorganization of base table spaces that use LOB columns.

In DB2 10 new-function mode, the syntax for the REORG TABLESPACE statement is changed. For partitioned table spaces, the PART specification is extended to allow for multiple parts or part ranges, and the SHRLEVEL REFERENCE and CHANGE specifications are extended to add the new keyword, AUX YES/NO. This new keyword allows for the

Archived

reorganization of LOB and XML table spaces that are associated with the base table. It is also possible to specify SHRLEVEL CHANGE for REORG of a LOB.

This is just a brief overview of how online REORG has been improved. Refer to 11.4, “Online REORG enhancements” on page 452 for more information about these enhancements.

4.9 Increased availability for CHECK utilities

DB2 10 provides increased availability and consistency for the CHECK DATA utility and the CHECK LOB utility.

In previous releases of DB2 for z/OS, the CHECK DATA and CHECK LOB utilities set restrictive states on the table space after completion. With DB2 10, if these utilities find inconsistencies and if the object was not previously restricted, a restrictive state is not set automatically on the table space. The new CHECK_SETCHKP subsystem parameter specifies whether the CHECK DATA and CHECK LOB utilities are to place inconsistent objects in CHECK-pending status.

This is just a brief overview of how CHECK is improved. Refer to 11.5, “Increased availability for CHECK utilities” on page 464 for details about this enhancement.

Archived

5

Data sharing

In this chapter, we describe the improvements to data sharing that are introduced with DB2 10. The main focus of these functions is on availability and performance.

This chapter includes the following topics:

- Subgroup attach name

  You can connect to only a subset of DB2 members, in much the same way as you can connect to a subset of DB2 members through distributed data facility (DDF) using Location Alias names.

- Delete data sharing member

  You can more easily consolidate the number of data sharing members in a group.

- Buffer pool scan avoidance

  DB2 now eliminates buffer pool scans when page sets change group buffer pool dependency, removing some transaction delays.

- Universal table space support for MEMBER CLUSTER

  Universal table spaces now support MEMBER CLUSTER for those heavy INSERT workloads.

- Restart light handles DDF indoubt units of recovery

  DB2 restart light is now able to resolve indoubt DDF units of recovery.

- Auto rebuild coupling facility lock structure on long IRLM waits during restart

  DB2 can recover from some types of restart delays caused by locking delays.

- Log record sequence number spin avoidance for inserts to the same page

  DB2 10 enhances the log record sequence number (LRSN) spin avoidance introduced in DB2 9 to further reduce CPU time for heavy INSERT multi-row applications.

- IFCID 359 for index split

  Tracing is now provided to allow you to monitor where and when index leaves page splits occur. You can then develop strategies to reduce this expensive data sharing activity.

© Copyright IBM Corp. 2010. All rights reserved.

- Avoid cross invalidations

  DB2 10 avoids excessive cross invalidations when a page set is converted from group buffer pool dependent to non-group buffer pool dependent.

- Recent DB2 9 enhancements

  A number of data sharing enhancements are introduced through service to DB2 9 for z/OS.

Several of these topics are also important for non-data sharing environments.

5.1 Subgroup attach name

With DB2 9, you can connect to a data sharing group by specifying either the DB2 subsystem name or the group attach name in CICS, TSO Attach, CAF, RRSAF, JDBC, ODBC, and DB2 utilities connection requests to DB2 for z/OS.

DB2 10 enhances this function through support for subgroup attach. You can connect to only a subset of DB2 members, in much the same way as you can connect to a subset of DB2 members through DDF. You can organize and control the DB2 members that are active and the workload that can be performed on each member.

Consider an environment where you have a two member data sharing group that runs on two separate LPARs and that supports an OLTP workload. To add two new *warehousing* members, both LPAR1 and LPAR2 need to have an OLTP member and a warehousing member. You also need a way for the warehousing applications to connect to the warehousing members while the OLTP applications continue to connect to the OLTP members. For DDF connections, this setup was possible through location alias name support. However, for local z/OS applications, this setup was not possible prior to the sub-group attach name that is introduced in DB2 10.

A *group attach name*, if defined, acts as a generic name for all the members of a data sharing group. You can use a *subgroup attach name* to specify a subset of members within a group attachment. Group and subgroup attachments can be specified in CICS, TSO, CAF, RRSAF, JDBC, ODBC, and DB2 utilities connections to find active DB2 subsystems.

Group attach names and subgroup attach names are defined in the IEFSSNxx parmlib member as follows:

```
SUBSYS SUBNAME(ssname) INITRTN(DSN3INI)
INITPARM('DSN3EPX,cmd-prefix<,scope<,GRP<,SGRP>>>')
```

Where *GRP* is the group attach name and *SGRP* is the subgroup attach name. You can also dynamically add a new DB2 subsystem to a group and subgroup attach with the SETSSI command.

Subgroup attach names follow the same naming rules as group attach names. They must be 1-4 characters in length. They can consist of letters A-Z, numbers 0-9, and the symbols $, #, and @. The names should not be the same as an existing DB2 member name (not enforced).

The following example shows a data sharing group definition that uses subgroup attach:

```
DB1A,DSN3INI,'DSN3EPX,-DB1A,S,DBOA'
DB2A,DSN3INI,'DSN3EPX,-DB2A,S,DBOA,SBG1'
DB3A,DSN3INI,'DSN3EPX,-DB3A,S,DBOA,SBG1'
DB4A,DSN3INI,'DSN3EPX,-DB4A,S,DBOA,SBG2'
```

In this example, you can connect to DB1A by specifying DB1A, you can connect to DB2A or DB3A (whichever is active on the LPAR) by specifying SBG1, you can connect to DB4A by specifying SBG2, and you can connect to DB1A, DB2A, DB3A or DB4A (whichever is active on the LPAR) by specifying DB0A.

Subgroup attach names are subject to certain rules and conditions, which are clarified by new messages added to DB2:

- The subgroup attach name can belong only to one group attach definition.
- The subgroup attach name must have a group attach name.
- The subgroup attach name cannot have the same name as the group attach name.
- A DB2 member can belong only to at most one subgroup attach.
- A DB2 member does not need to belong to a subgroup attach.

The -DISPLAY GROUP DETAIL command is enhanced to display all defined members of all subgroup attach names for group attach names that are defined to the given member.

Dynamic reconfiguration of subgroup attach names are not supported, and an IPL is required to clear subgroup attach data (as is the case with group attach). Subgroup attach names are also subject to the same rules concerning RANDOM ATTACH as group attach names. See 5.10.1, “Random group attach DSNZPARM” on page 120 for a discussion about RANDOM ATTACH.

Using the -SET SYSPARM command to change RANDOM ATTACH NO/YES changes the member information for both group attach and subgroup attach. For example, a member that is defined to group DSNG displays only the subgroups that are defined to DSNG.

The DSNTIPK installation panel is changed to include a SUBGRP ATTACH field that, if specified, identifies the name of a new or existing subgroup attach name within the group attach name that is to be used for this DB2 subsystem.

When you submit a job on a z/OS system, for example, DB2 treats the name that you specified on the DB2 connection request as a subsystem name, a group attach name, or a subgroup attach name. Briefly, DB2 first assumes that the name is a subsystem name and then uses the one of the following process:

- Attaches to that subsystem if it is found and active.
- Assumes the attachment name is either a group attach name or a subgroup attach name, and performs group attach processing if either of the following conditions are met:
  - The name was not found in the subsystem interface list
  - The name was found, but the subsystem was not active, it has a group attach name, and the NOGROUP option was *not* specified on the connection.
- The name was found, but the subsystem was not active and either it lacks a group attach name or the NOGGROUP option was specified on the connection. Thus, the connection is unsuccessful.

For group attach processing, DB2 assumes that the name on the DB2 connection request is a group attach name or subgroup attach name. DB2 creates a list of DB2 subsystems that are defined to this z/OS. DB2 then searches the list first randomly and then sequentially in the order each subsystem was registered to the MSV Subsystem Interface (SSI), depending on the RANDOM ATTACH parameter. Because group attach names and subgroup attach names must be unique, there is no hierarchy, and both types of attachment names are processed together.

Subgroup attach works upon fallback to DB2 Version 8 or Version 9 if the DB2 10 early code is used to define a DB2 subsystem either at IPL or with the z/OS SETSSI command.

Subgroup attach is not available if DB2 Version 8 or Version 9 early code was used to define a DB2 subsystem. There are no coexistence considerations.

The following messages are included:

- DSN3118I csect-name DSN3UR00 - GROUP ATTACH NAME grpn WAS ALREADY DEFINED AS A SUBGROUP ATTACH
- DSN3119I csect-name DSN3UR00 - SUBGROUP ATTACH sgrpn DOES NOT BELONG TO GROUP ATTACH grpn
- DSN3120I csect-name DSN3UR00 - grpn CANNOT BE DEFINED AS BOTH THE GROUP ATTACH NAME AND THE SUBGROUP ATTACH NAME
- DSN3121I csect-name DSN3UR00 - SUBGROUP ATTACH sgrpn WAS ALREADY DEFINED AS A GROUP ATTACH
- DSNT563I AN ENTRY IN field-name-1 REQUIRES AN ENTRY IN field-name-2
- DSNT563I THE ENTRY IN field-name-1 CANNOT BE THE SAME AS THE ENTRY IN field-name-2

5.2 Delete data sharing member

Prior to DB2 10, a *dormant* data sharing member could not be deleted. This has traditionally not been an issue, as there is little or no effort in maintaining these dormant members. However, DB2 10 brings the potential for consolidating the data sharing group to fewer members and being able to delete a member becomes important since you can potentially run the same workload in fewer members. The maintenance stream has provided this function after GA with APARs PM31003, PM31004, PM31006, PM31007, PM31009. PM42528, and PM 51945.

There are several instances where a function DELETE MEMBER can be of help:

- Consolidation to smaller numbers of members, as System z processors have continued to become more and more powerful, coupled with the news that DB2 10 provides the potential to run the same workload with fewer DB2 members.
- DB2 members have been added by mistake.
- You need to maintain an old copy of the BSDS for the member that is no longer needed. (However there is the option to reply QUIESCED to the DB2 restart messages for old members on Group Restart.)
- Possible complications in disaster recovery scenarios. If you do not have the BSDS for the dormant member(s) at the DR site, you then need to reply to the WTOR message on DB2 restart.
- Cloning production systems to create test systems might not need all members in the test system. This of course depends on how you clone your production system.

DB2 10 provides a DELETE MEMBER offline function which you can use to delete dormant DB2 member(s) from a data sharing group.

The Change Log Inventory utility, DSNJU003, has been enhanced to delete *quiesced* members. This is therefore an offline implementation where the entire data sharing group must be brought down so that the member record of the *quiesced, to be deleted* member in the individual member BSDSs can be updated to remove the member details.

Two new operators are introduced to the DSNJU003 utility, DELMBR, to deactivate or destroy a member, and RSTMBR, to restore a deactivated member to the quiesced state.

- DELMBR

  DELMBR operates in two phases: a DEACTIVate and a DESTROY phase.

  DELMBR DEACTIV Must be done first and marks as deactivated the member record of the 'cleanly quiesced, to be deleted member' in the BSDS of the member on which the utility is run. You need to still keep the deactivated member's BSDS and log data sets, should you wish to re-activate the member.
  The result of successfully running this command is that the member record in the BSDS is marked as deactivated. The member is ignored during restart and any attempt to restart the deactivated member fails with an abend.
  The space containing the deactivated member record in the BSDS remains in use, as does the associated record in the SCA. Also, the member name and id cannot be reused while the member is in this state.
  A deactivated member can be restored to the quiesced state via the RSTMBR operator

  DELMBR DESTROY Can subsequently be run to reclaim the space in the BSDS so it can be reused by future new member records. Once this command has been successfully run, the deactivated member cannot be restored. A destroyed member's BSDS and logs can be deleted. You must first deactivate the dormant DB2 member before destroying it.

- RSTMBR

  RSTMBR restores a previously deactivated, but not destroyed member, to the quiesced state.

To deactivate a member, the DSNJU003 utility must be run against the BSDS data sets of all DB2 members, including the member to be deleted and any other inactive members. So, all DB2 members need to be shut down.

You need to run DSNJU003 against the member to be deleted, to define the member as *deactivated* so it cannot be started in error. You also need to run the DSNJU003 against the deactivated member in case you want to restore it.

A member that is to be deactivated must first be cleanly quiesced, with no open SYSLGRNX records, and no:

- URs (inflight, incommit, indoubt, inabort or postponed abort)
- Retained locks
- Active utilities
- Cast out failures
- Re-sync required

Note: It is your responsibility to check for the above conditions when you wish to deactivate a member, as they are not checked by DB2. Potential data integrity problems may occur if you deactivate and destroy a DB2 member which still has outstanding work.

The overall flow you need to follow to delete a dormant DB2 member is:

1. The initial deletion.

Run DSNJU003 with DELMBR DEACTIV against ALL members BSDS data sets in the data sharing group, including the dormant member. This requires all members to be brought down (only the deleted member needs to be quiesced).

2. Restart the group.

   The deleted member is marked as such an can not be restarted.

3. Destroy the deleted member.

   At some point when the deleted member's logs are no longer needed, it can be destroyed and all record of it is gone.

During DB2 restart, a deactivated member is recognized and the DSNR020I WTOR is not issued. However, DB2 restart issues a new informational message:

```
DSNR032I DEACTIVATED MEMBER xx WAS ENCOUNTERED AND IS BEING IGNORED.
IT WILL NOT BE STARTED
```

This message is issued for each deactivated member to indicate that the member's logs are disregarded just as is done today for quiesced members.

Attempts to restart the deactivated member are terminated with a new message:

```
DSNR033I THIS MEMBER BEING STARTED IS MARKED AS DEACTIVATED AND CANNOT BE
STARTED. PROCESSING IS TERMINATED
```

and abend 04E with new reason 00D10130 is issued.

5.3 Buffer pool scan avoidance

Prior to DB2 10, DB2 members needed to execute complete scans of the buffer pool during certain transitions in inter-DB2 Read/Write (RW) interest for a page set/partition (p/p). Examples include (these are not the only cases):

- When a p/p transitions from P-lock S state (inter-DB2 Read Only (RO)) to P-lock IS state (another member has declared RW interest for this p/p) then each member downgrading the P-lock on the p/p from S to IS must scan the buffer pool which this p/p is using to ensure that all the locally cached pages for this p/p are properly registered to the GBP for cross invalidation (XI).
- The same is true for SIX to IX lock transition (this member used to be the sole updater for this p/p, but now another member had declared RW interest).

Usually the buffer pool scan is fast. However, as buffer pool sizes continue to grow larger and larger, the time to execute the buffer pool scan similarly takes longer and longer and can sometimes can cause noticeable transaction delays.

DB2 10 eliminates all of these sequential buffer pool scans. This elimination in turn eliminates the prospects of variable and unpredictable transaction delays for those transactions that cause the changes in the inter-DB2 RW interest levels, especially in the case of larger buffer pools sizes.

5.4 Universal table space support for MEMBER CLUSTER

DB2 typically tries to place data into a table using INSERT, based on the clustering sequence as defined by the implicit clustering index (the first index created) or the explicit clustering

index. This can cause “hot spots” in data page ranges and high update activity in the corresponding space map page or pages. These updates to space map page or pages and data page or pages must be serialized among all members in a data sharing environment, which can adversely affect INSERT/UPDATE performance in data sharing.

DB2 Version 5 introduced the MEMBER CLUSTER option of CREATE TABLESPACE statement of a partitioned table space to address this bottleneck. MEMBER CLUSTER causes DB2 to manage space for inserts on a member-by-member basis instead of by using one centralized space map. The main idea of a MEMBER CLUSTER page set is that each member has exclusive use of a set of data pages and their associated space map page. (Each space map covers 199 data pages.) Each member would INSERT into a different set of data pages and not share data pages, thereby eliminating contention on pages and reducing time spent searching for pages.

MEMBER CLUSTER is used successfully to reduce contention in heavy INSERT applications, such as inserting to journal tables.

DB2 9 introduced universal table spaces (UTS). They combine both the enhanced space search benefits of segmented table spaces with the flexibility in size of partitioned table spaces. You can have two types of UTS (which is the strategic table space for DB2):

- Partition-by-growth
- Partition-by-range

Several functions are supported only through UTS. However, DB2 9 does not support MEMBER CLUSTER in a UTS. So, you have to choose between the benefits of MEMBER CLUSTER (reduced contention in a heavy INSERT environment) with the flexibility of UTS (better space management and new functions).

DB2 10 in new-function mode removes this restriction. MEMBER CLUSTER is supported by both partition-by-growth and range-partitioned UTS.

Each space map of a MEMBER CLUSTER UTS contains 10 segments. So, the number of data pages that are allocated to each DB2 member varies, depending on SEGSIZE. A large SEGSIZE value causes more data pages to be covered by the space map page, which in turn causes the table space to grow in a multiple member data sharing environment. If the SEGSIZE is small, then more space map pages are required. Therefore, we suggest using SEGSIZE of 32 for MEMBER CLUSTER universal table spaces instead of the default 4.

As with prior versions of DB2, a MEMBER CLUSTER table space becomes unclustered quickly. You need to REORG the table space to bring the data back into clustering sequence.

To create a MEMBER CLUSTER partition-by-range UTS:

```
CREATE TABLESPACE MySpace IN MyDB
MEMBER CLUSTER
MUNPARTS 3;
```

To create a MEMBER CLUSTER partition-by-growth UTS:

```
CREATE TABLESPACE MySpace in MyDB
MEMBER CLUSTER
MSAXPARTITIONS 10;
```

You can also implement MEMBER CLUSTER using an ALTER, without having to drop and recreate the table space:

```
ALTER TABLESPACE MyDB.MySpace .......... MEMBER CLUSTER YES/NO
```

The need to use MEMBER CLUSTER can change over time because the data access requirements can change from a heavy INSERT to a more query base. Therefore, DB2 10 also provides the capability to both enable and disable MEMBER CLUSTER for universal table space.

The ALTER statement updates the catalog table SYSIBM.SYSPENDINGDDL to indicate there is a pending change and places the table space in Advisory REORG-pending state (AREOR). DB2 also issues the new SQLCODE +610 to indicate this fact. You are then advised to schedule the REORG utility on the entire table space to materialize the pending ALTER. (The REORG also restores the data clustering.) A LOAD on the table space level also materializes the pending ALTER and honor the data clustering.

The MEMBER_CLUSTER column in the SYSIBM.SYSTABLESPACE catalog table is set for MEMBER CLUSTER table spaces in DB2 10 NFM, after the pending ALTER is resolved by LOAD or REORG. Existing MEMBER CLUSTER table spaces were created prior to DB2 10 in NFM have the TYPE column in the SYSIBM.SYSTABLESPACE catalog table set. During the DB2 10 enabling-new-function mode (ENFM) process, these values are moved to the MEMBER_CLUSTER column.

You can use the ALTER TABLESPACE statement to migrate a single table MEMBER CLUSTER table space to a MEMBER CLUSTER universal table space.

If you use the same SQL syntax for a classic partitioned table space in DB2 10 NFM, DB2 creates a partition-by-range UTS. However, you can use SEGSIZE 0 with the MEMBER CLUSTER clause and the NUMPARTS clause of the CREATE TABLESPACE statement to create a classic partitioned table space with the MEMBER CLUSTER structure, which can be useful if you use the DSN1COPY utility to copy one classic partitioned table space to another.

To create a MEMBER CLUSTER classic partitioned table space;

```
CREATE TABLESPACE MySpace in MyDB
MEMBER CLUSTER SEGSIZE 0
NUMPARTS 3;
```

Figure 5-1 shows the implication to RECOVER on making structure changes, such as ALTER MEMBER CLUSTER. When the ALTER MEMBER CLUSTER change is materialized by a REORG, DB2 prohibits recovery to any point in time prior to the REORG. However, you can still recover to a point in time before the REORG, including after the ALTER statement was run, provided that the REORG has not been run. This process is true for any deferred ALTER introduced in V10, with the following exceptions:

- ALTER TABLE ADD HASH and ALTER TABLE HASH SPACE
- ALTER the length of an inline LOB column

These exceptions allow a recover to a point in time that is prior to the materialization.

Note also that any image copy taken prior to when the change is materialized (for example the blue image copy), cannot be used to recover to any point in time after the REORG was run, including to current. DB2 cannot restore a full image copy taken prior to the REORG then apply logs across the REORG, which has always been the case.

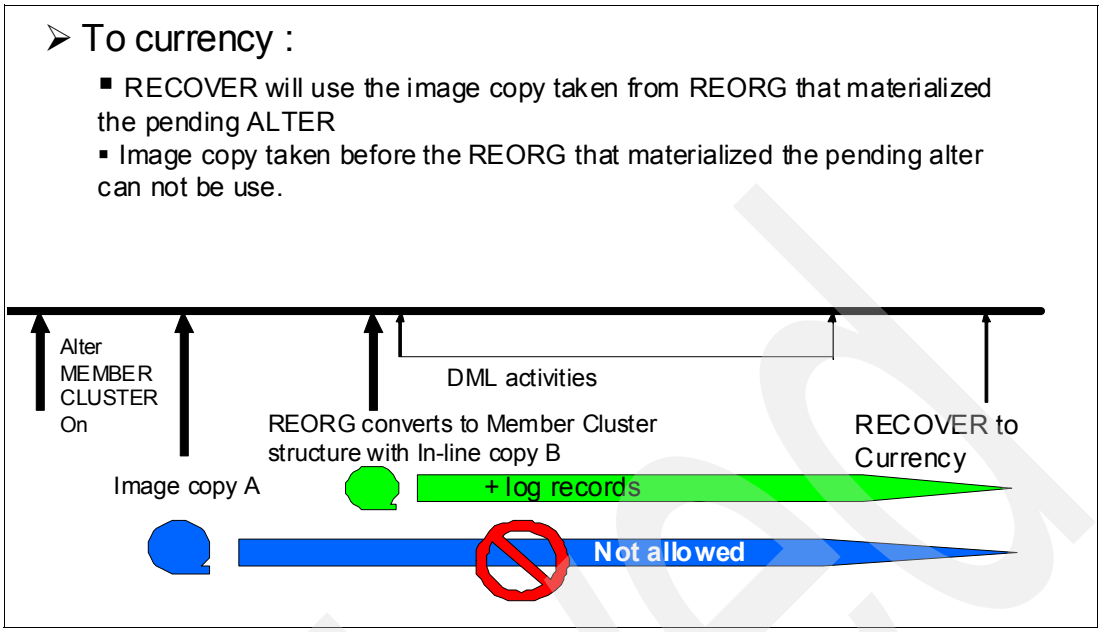


Figure 5-1 MEMBER CLUSTER with RECOVER

5.5 Restart light handles DDF indoubt units of recovery

When a DB2 9 data sharing member is started in light mode (restart light), retained locks cannot be freed for an indoubt unit of recovery if the coordinator of the unit of recovery was a remote system. This situation occurs because DDF automatic indoubt thread resolution (resync) functionality is required to determine the unit of recovery decision but DDF is unavailable in light mode. In fact, the ssnmDIST address space is not even started in light mode.

DB2 10 implements DDF restart light support to automatically resolve indoubt units of recovery if the coordinator of the unit of recovery is a remote system.

Note that most DB2 commands are restricted in restart light mode. The only commands available are Start DB2 or Stop DB2, Display Thread (Indoubt), and Recover (Indoubt). With the implementation of DDF restart light support for remote indoubt units of recovery, the START and STOP DDF commands are also available in restart light mode.

DB2 for z/OS in light mode is concerned only with indoubt units of recovery where DB2 for z/OS is a participant relative to a local or remote coordinator that owns the decision. DB2 for z/OS can also be a coordinator relative to local or remote participants, and decisions can be provided to remote participants as long as DB2 and DDF remain started while resolving its own participant related indoubt units of recovery. When all indoubt units of recovery are resolved, DB2 terminates automatically even if DB2 still has coordinator responsibility with respect to local or remote indoubt participants.

XA recover processing for a remote XA Transaction Manager requires that XA Transaction Manager first connect to the group's SQL port to retrieve a list of transactions (XIDs) that are indoubt. Because the SQL port is unavailable for the light member, another member of the group must be available and be accessible through a group DVIPA. After this other member is contacted, it returns the member DVIPA and resync port for any indoubt work (XIDs) owned by the light member. XA Transaction Manager can then resolve work at the light member because its resynch port is available. Thus, another DB2 member must be active for the

remote XA Transaction Manager to resolve any indoubt units of recovery on the member that is started in light mode.

5.6 Auto rebuild coupling facility lock structure on long IRLM waits during restart

In the past, users initiated a group restart when DB2 appeared to have “stalled” on restart. In one case that was investigated, a stall occurred during DB2 restart after an internal resource lock manager (IRLM) failure. Rather than initiating an IRLM lock structure rebuild, both a lock and shared communications area (SCA) rebuild were started. The SCA rebuild stalled, and the user then shut down all members and performed a group restart. Instead, only the lock structure rebuild should have been started. The action that was taken was disruptive, because all DB2 members must be down. A group restart should be initiated only as a last resort.

In many cases, normal restart stalls are due to some incompatibility on a specific resource that can be resolved with a simple lock structure rebuild. DB2 10 provides a function to recover from these stalls automatically on restart. The IRLM now initiates an automatic rebuild of the lock structure into the same coupling facility (CF) if it detects long waits from the CF.

DB2 has a restart monitor that periodically checks the progress of DB2 restart. If the restart monitor detects that restart was suspended for between 4-6 minutes on an IRLM request (usually a lock request), DB2 notifies IRLM, and IRLM initiates the lock structure rebuild in an attempt to allow DB2 restart to proceed.

A message is presented to the console to record this event:

```
DXR184I irlmx REBUILDING LOCK STRUCTURE AT REQUEST OF DBMS IRLM QUERY
```

The event is also recorded by a new reason codes in IFCID 267 record as shown in Example A-6 on page 621.

5.7 Log record sequence number spin avoidance for inserts to the same page

DB2 9 in NFM provided a function called *LRSN spin avoidance* that allows for duplicate log record sequence number (LRSN) values for consecutive log records on a given member. Consecutive log records that are written for updates to different pages (for example, a data page and an index page, which is a common scenario) can share the same LRSN value. However, in DB2 9, consecutive log records for the same index or data page must still be unique.

The DB2 member does not need to “spin” consuming CPU under the log latch to wait for the next LRSN increment. This function can avoid significant CPU overhead and log latch contention (LC19) in data sharing environments with heavy logging. See *DB2 9 for z/OS Performance Topics*, SG24-7473 for details. This spin might still be an issue with multi-row INSERT.

DB2 10 further extends LRSN spin avoidance. In DB2 10 NFM, consecutive log records for inserts to the same data page can now have the same LRSN value. If consecutive log records are to the same data page, then DB2 no longer continues to “spin” waiting for the LRSN to

increment. The log apply process of recovery has been enhanced to accommodate these duplicate LRSN values.

Duplicate LRSN values for consecutive log records for the same data page set are allowed only for INSERT type log records. DELETE and UPDATE log records still require unique LRSN values.

This enhancement helps applications, such as those applications that use multi-row INSERT where the table or tables that are inserted into have none or only a few indexes, by further reducing CPU utilization.

5.8 IFCID 359 for index split

Index leaf page splits can be expensive in data sharing and can impact application performance. The DB2 member that is performing the index page split needs to execute two synchronous writes to the log during the split. This process can elongate transaction response time in some cases. The index tree P-lock can also become a source for delays if lots of index splitting is occurring between members. So, many data sharing users care about monitoring index page splits.

Monitoring index leaf page splits might be less important in non-data sharing environments, but there are still benefits. To achieve optimal performance of index access, you need to keep the index organized efficiently. However, running frequent REORGs can increase the system maintenance costs and at time be disruptive to your application workload or workloads. Monitoring index leaf page splits can help you to decide when and how often to reorganize your key indexes.

DB2 10 provides some tracing through the new IFCID 359, to help you to identify index leaf page split activity. The new IFCID 359 records when index page splits occur and on what indexes. Example A-10 on page 622 shows the format of IFCID 359. A X'80' in fields QW0359FL indicates the index was GBP Dependent during the index leaf page split.

IFCID 359 record: The IFCID 359 record is not included with any trace class. You need to start it explicitly. A record is written at the end of every leaf page split that lists the index and that page that was split.

5.9 Avoid cross invalidations

DB2 10 avoids excessive cross invalidations when a page set is converted to from group buffer pool dependent to non-group buffer pool dependent.

When a page set is converted to non-group buffer pool dependent, (which is typically driven by pseudo close or physical close processing when the close activity results in the removal of inter-DB2 R/W interest), DB2 first takes a serialization lock on the page set then performs castout processing for all the relevant pages in the group buffer pool. When castout is complete, DB2 then issues a *delete name* request to the CF.

The delete name request instructs the CF to free the pages in the CF occupied by the page set and to deregister the pages from the group buffer pool directory. Deregistering the pages from the directory also causes cross invalidation requests to be percolated to all the DB2 members that also have cached pages for this page set.

DB2 10 instructs the delete name request to only delete the data entries from the group buffer pool for the nominated page set or sets. Directory entries are left untouched, which in turn avoids cross invalidation requests that are percolated to the members. In time, the directory entries are reused. Pages for this nominated page set can also reside in member or members buffer pools, but these pages should have already been made invalid if a more recent version of the page resides in the group buffer pool. So, there is no need to send cross invalidation requests to all the DB2 members.

This enhancement avoids situations such as IRLM lock timeouts for online transactions due to a long running CF *delete name* process in DB2. Consider the situation where a DB2 member driving the *delete name* requests is situated in an LPAR that is a long distance away from the CF that contains the primary GBP. There can many thousands of directory entries to be deleted, and for every entry deleted, the CF must send cross invalidation requests through XCF to individual DB2 members that also contain pages for this deleted page set. This is the CF architecture. If the XCF requests are delayed, for example because the signals had to travel a long distance, the *delete name* request might also be able to only handle a few pages at a time. So, to delete all of the entries for the page set, the delete name request might need to be retried many times, requiring several tens of seconds. During this time, the DB2 member holds locks on the page set (for data integrity reasons), and this serialization can cause IRLM timeouts for some transactions.

5.10 Recent DB2 9 enhancements

DB2 9 introduced a number of data sharing enhancements for z/OS through the service stream. We list them in this section for completeness.

5.10.1 Random group attach DSNZPARM

DB2 9 introduced a behavior to the way DB2 subsystems are selected during group attach processing. DB2 subsystems are chosen at random, rather than the DB2 V8 behavior where group attachment requests are always attempted in the order of DB2 subsystem initialization. Some customers relied on DB2 V8 behavior to isolate batch processing from transaction processing.

APAR PK79327 introduces the DSNZPARM parameter, RANDOMATT=YES/NO, to restore the DB2 V8 group attachment behavior after migrating to DB2 9 and to provide additional flexibility.

You can specify RANDOMATT=NO to exclude a DB2 member from being selected at random. To satisfy a group attachment request, DB2 first randomly searches for an active DB2 subsystem, excluding those specifying RANDOMATT=NO. If all such subsystems are inactive, DB2 searches for any active DB2 subsystem within the group, in the order that is defined by the z/OS subsystem initialization, regardless of the setting of RANDOMATT.

To achieve non-randomized group attachments for all members, as in DB2 Version 8, specify RANDOMATT=NO for all members of the data sharing group.

5.10.2 Automatic GRECP recovery functions

DB2 9 tries to handle objects that are in GRECP during restart. However, automatic GRECP recovery can be delayed for Lock timeouts or a deadlock with another member.

The automatic GRECP recovery process first acquires drains on all of the objects being recovered, then performs physical opens. During the physical open, the down-level detection code attempts to retrieve the current level-ID from SYSLGRNX. This is done using an execution unit switch to an unrelated task, so if the main task has already drained SYSLGRNX, the other task ends up timing out on a drain lock while trying to acquire a read claim.

Additionally, if another member is restarting, it might be suspended on a drain lock for DBD01, if the automatic GRECP recovery process has drained DBD01, which might untimately cause recovery to fail for that object.

The down-level detection logic is modified with APAR PK80320 to bypass objects on restart that are in GRECP state. Automatic GRECP processing has also been modified to not be suspended or timed out if another member is in restart processing.

Table 5-1 lists other recommended maintenance for GRECP/LPL that should be applied to your DB2 9 systems.

Table 5-1 DB2 9 GRECP/LPL maintenance

APAR	Brief description
PK80320	Bypass objects that are in GRECP state on restart
PM02631	Utilities in data sharing when target object exceeds 200 partitions
PM03795	Delay during startup when several DB2 members start concurrently if any deadlocks occur
PM05255	Add DBET diagnostic data for data sharing
PM06324	Problems during GRECP/LPL recovery, or Recovery utility if there are two CLR log records with the same LRSN value against the same page
PM06760	During restart AUTO-GRECP recovery processing takes too long
PM06933	The DBET GRECP derived flags are not correctly set with the GRECP data at the partition level.
PM06972	AUTO-GRECP recovery failed on SYSLGRNX or takes a long time to recover SYSLGRNX
PM07357	Open-ended log range for GRECP/LPL recovery
PM11441	Long running backout recovery during LPL/GRECP recovery
PM11446	DSNI021I incorrectly issued when GRECP/LPL was abnormally terminated

5.10.3 The -ACCESS DATABASE command enhancements

DB2 9 introduced the -ACCESS DATABASE command to force a table space, index space, or partition to be physically opened or to remove the GBP-dependent status for a table space, index space, or partition.

Archived

The command is enhanced with APAR PK80925 to support the use of wildcards and subset ranges in the DATABASE and SPACENAM keywords. This enhancement is a major usability and manageability benefit, especially for large shops who need to manage thousands of objects.

Figure 5-2 shows the syntax of the -ACCESS DATABASE command with the wild carding that is now supported.

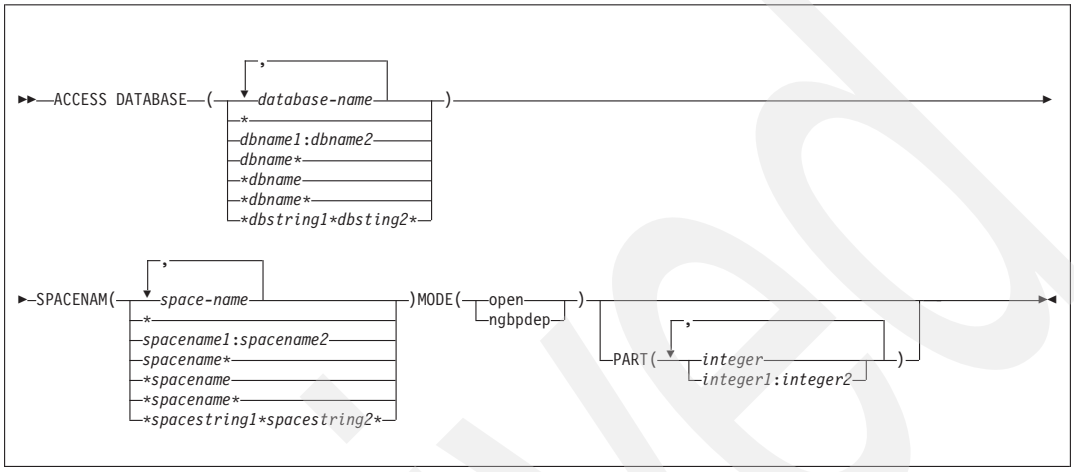


Figure 5-2 ACCESS DATABASE command

5.10.4 Reduction in forced log writes

DB2 performs forced log writes after inserting into a newly formatted or allocated page on a GBP dependent segmented or UTS. The forced log writes lead to a performance degradation for insert operations and inconsistent log write I/O times (both elapsed time and events) for the same job running on two different days.

APARs PK83735 and PK94122 changed DB2 9 so that the forced log writes are no longer done when inserting into a newly formatted or allocated page for a GBP dependent segmented or universal table space.

Archived

Part 2

Application functions

In this part, we describe functions that help to enable the application development with DB2 10 for z/OS.

DB2 10 delivers several SQL enhancements that can help applications to ease development and porting. DB2 10 also provide several enhancements to the support of pureXML, allowing companies to seamlessly integrate their XML data with relational data in an efficient and high performance manner.

DB2 10 native SQL stored procedures again improve performance and manageability.

DB2 for z/OS data today is accessed though modern application development environments such as Java. Application developers do not need to be aware that the database resides on the mainframe, because the application languages that are supported by DB2 are fully portable using the DB2 Universal JDBC Driver.

Because of the large amount of application-related enhancements, we have divided them across several chapters The SQL contents have been split between two chapters. One chapter is more related to the SQL language, and the other chapter to application-enabling infrastructure changes. XML and e-business have each a dedicated chapter.

This part contains the following chapters:

- Chapter 6, “SQL” on page 125
- Chapter 7, “Application enablement” on page 203
- Chapter 8, “XML” on page 243
- Chapter 9, “Connectivity and administration routines” on page 309

© Copyright IBM Corp. 2010. All rights reserved.

Archived

6

SQL

In this chapter, we describe SQL-related functionality that is introduced in DB2 10 for z/OS. We provide examples that demonstrate how these new features work, and we discuss nuances or functional behavior that are helpful to understand. The examples can help you in adapting and using these features in your applications.

The dominant theme in SQL-related changes in DB2 10 for z/OS is the move towards simplification of application development and porting. Thus, the majority of features and enhancements to existing functionality in DB2 10 for z/OS are geared towards capabilities that are present in other members of the DB2 family and in competitive databases. For example, the TIMESTAMP WITH TIME ZONE data type improves the support for applications that deal with multiple time zones. User-defined SQL routines are enhanced to allow greater facilitation of SQL Procedural Language. OLAP specifications are implemented that support moving aggregates, which are common in warehousing workloads.

In this chapter, we specifically discuss the following topics:

- Enhanced support for SQL scalar functions
- Support for SQL table functions
- Enhanced support for native SQL procedures
- Extended support for implicit casting
- Support for datetime constants
- Variable timestamp precision
- Support for TIMESTAMP WITH TIME ZONE
- Support for OLAP aggregation specification

© Copyright IBM Corp. 2010. All rights reserved.

6.1 Enhanced support for SQL scalar functions

As a quick background information summary, a *user-defined function* provides a mechanism for extending the functionality of the database by adding a function that can be evaluated in SQL statements. The DB2 CREATE FUNCTION statement registers the following types of user-defined functions with the database:

- External scalar

  The function is written in a programming language and returns a scalar value. The external executable is registered with a database server along with various attributes of the function.

- External table

  The function is written in a programming language and returns a complete table. The external executable is registered with a database server along with various attributes of the function.

- Sourced

  The function is implemented by invoking another function (either built-in, external, SQL, or sourced) that already exists at the server. The function inherits the attributes of the underlying source function.

- SQL scalar

  The function is written exclusively in SQL statements and returns a *scalar value*. The body of a SQL scalar function is written in the SQL procedural language and is defined at the current server along with various attributes of the function. DB2 10 for z/OS enhances SQL scalar functions. We discuss the enhancements in this section.

- SQL table

  The function is written exclusively in SQL statements and returns a *set of rows*. The body of a SQL table function is written in the SQL procedural language and is defined at the current server along with various attributes of the function. DB2 10 for z/OS adds support for SQL table functions. We discuss the enhancements at 6.2, “Support for SQL table functions” on page 144.

Additional information: For more information about the DB2 CREATE FUNCTION statement, refer to *DB2 10 for z/OS SQL Reference,* SC19-2983.

Starting with DB2 8 for z/OS, a user-defined SQL scalar function was limited to a single RETURN statement that either specified an expression (excluding scalar fullselect) or specified NULL. If additional capability was needed by the function, an external scalar function was required instead. Because the body of the external function must be written in a host language (COBOL, PL/I, or other programming language), this requirement posed usability and performance concerns and inhibited porting functions from other database products.

To address these concerns, DB2 10 for z/OS enhances SQL scalar functions by allowing the use of SQL Procedural Language (SQL PL) in the function body. SQL scalar functions that are created in DB2 10 for z/OS new-function mode are enhanced by:

- Including SQL PL control statements
- Including scalar fullselect in the RETURN statement
- Using distinct type in the definition of a parameter or SQL variable
- Defining a parameter for a transition table

- Using MODIFIES SQL DATA attribute on the CREATE FUNCTION (SQL scalar) statement
- Using the DB2 Unified Debugger for non-inline SQL scalar functions

DB2 10 for z/OS classifies user-defined SQL scalar functions into the following types:

- Inline SQL scalar function

  This type of function supports the same capability as in prior releases. Thus, if the CREATE FUNCTION (SQL scalar) statement that is used to create the function can execute successfully in a prior DB2 release, the function is considered *inline*.

  No package is created for an inline SQL scalar function, and any reference to the function within a SQL statement is replaced with the expression specified on the simple form of the RETURN statement of the CREATE FUNCTION (SQL scalar).

  Allowable options: You can specify only the following options with the CREATE FUNCTION (SQL scalar) statement for a SQL scalar function to be considered *inline*:

  - LANGUAGE SQL
  - SPECIFIC
  - NOT DETERMINISTIC
  - DETERMINISTIC
  - EXTERNAL ACTION
  - NO EXTERNAL ACTION
  - READS SQL DATA
  - CONTAINS SQL
  - CALLED ON NULL INPUT
  - STATIC DISPATCH
  - PARAMETER CCSID

  If you use other options, the function is considered a *non-inline SQL scalar function*.

- Non-inline SQL scalar function

  This type of function does not qualify as an inline SQL scalar function. Typically, this type of function uses one or more of the enhancements that are not available prior to DB2 10 for z/OS new-function mode; however, the function can also be considered non-inline because it fails to qualify as an inline (per the note regarding allowable options previously).

  DB2 creates a package that is associated with the non-inline SQL scalar function. Any procedural statements are converted to a representation that is stored in the database directory. When a function is invoked, the native representation is loaded from the directory, and the DB2 engine executes the routine.

You can verify the value of the new column INLINE in the SYSIBM.SYSROUTINES catalog table to determine whether the given SQL scalar function is inline. INLINE contains a value of Y for an inline SQL scalar function.

You can query the SYSIBM.SYSPACKAGE catalog table to find packages that are associated with non-inline SQL scalar function. The column TYPE has a value of F for the package that is associated with non-inline SQL scalar function.

Non-inline SQL scalar functions use the following package naming conventions:

- LOCATION is set to the value of the CURRENT SERVER special register
- COLLID is set to the schema qualifier of the function
- NAME is set to the specific name of the function
- VERSION is set to the version identifier of the function

The package is generated using bind options that correspond to the implicitly or explicitly specified function options. In addition to the corresponding bind options, the package is generated using the following bind options:

- FLAG(I)
- SQLERROR(NOPACKAGE)
- ENABLE(*)

Rebinding: The package that is associated with non-inline SQL scalar function might contain SQL control statements (the logic part that was converted to the DB2 native representation and stored in the package) and the SQL statements that were specified in the function body. The REBIND PACKAGE command rebinds only the SQL statements that were specified in the function body. The SQL control statements are *not* rebound. The ALTER FUNCTION REGENERATE statement rebinds both SQL statements specified in the function body and the SQL control statements.

You might encounter a situation where you need to rebind the SQL control statements after applying DB2 maintenance so that the DB2 representation of the logic can be refreshed. Be aware that instead of executing the REBIND PACKAGE command, you need to use either the ALTER FUNCTION REGENERATE command or the ALTER FUNCTION REPLACE command, or you can alternatively drop and re-create the function.

6.1.1 SQL scalar functions syntax changes

Here, we describe relevant changes to the SQL syntax. For complete information about the syntax, refer to *DB2 10 for z/OS SQL Reference,* SC19-2983.

CREATE FUNCTION (SQL scalar)

Figure 6-1 shows the main portion of the CREATE FUNCTION (SQL scalar) syntax.

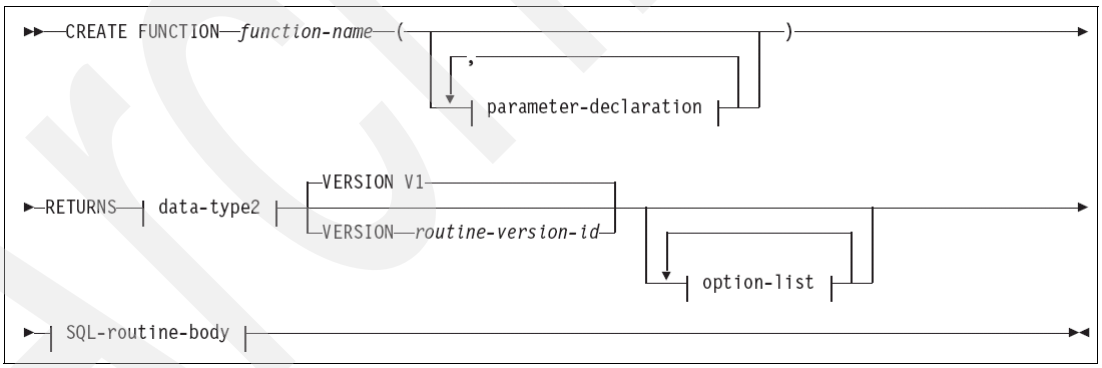


Figure 6-1 CREATE FUNCTION (SQL scalar) syntax diagram: Main

The VERSION clause is used for function versioning. It specifies the version identifier for the first version of the function that is to be generated. You can use an ALTER FUNCTION statement with the ADD VERSION clause or the BIND DEPLOY command to create additional versions of the function. The *routine-version-id* is a SQL identifier that can be up to 64 EBCDIC bytes, and its UTF-8 representation must not exceed 122 bytes. The default version identifier is V1.

Figure 6-2 shows the parameter declaration part of the syntax. Enhancements include the ability to declare parameter as a transition table and XML data type support.

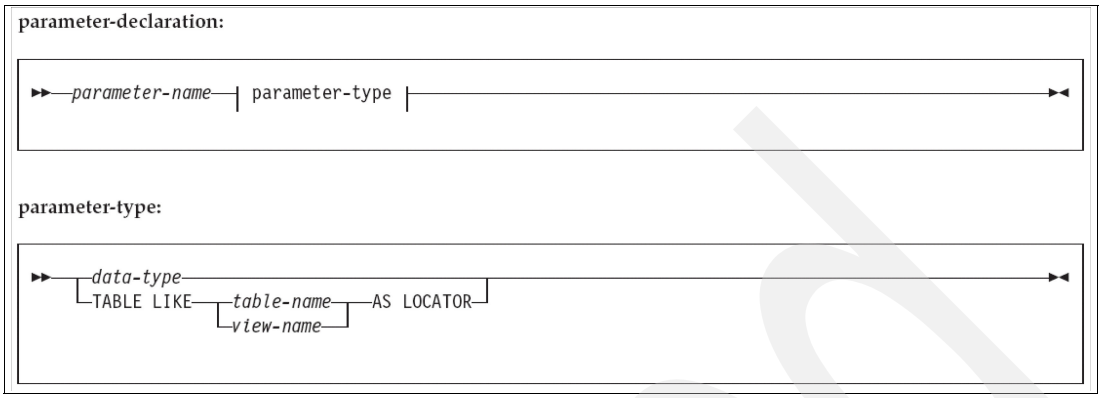


Figure 6-2 CREATE FUNCTION (SQL scalar) syntax diagram: parameter-declaration

The *SQL-routine-body* is either the simple RETURN statement (as supported in prior DB2 releases) or a SQL control statement, as shown in Figure 6-3.

Figure 6-3 CREATE FUNCTION (SQL scalar) syntax diagram: SQL-routine-body

To provide compatibility with previous releases of DB2 or other products in the DB2 family, DB2 supports the following keywords:

- VARIANT as a synonym for NOT DETERMINISTIC
- NOT VARIANT as a synonym for DETERMINISTIC
- NOT NULL CALL as a synonym for RETURNS NULL ON NULL INPUT
- NULL CALL as a synonym for CALLED ON NULL INPUT
- The RETURNS clause, the RETURN-statement, and the clauses in the option-list can be specified in any order. However, the RETURN-statement MUST be the last clause in the CREATE statement for a SQL scalar function
- TIMEZONE can be specified as an alternative to TIME ZONE in the DDL. LOAD and UNLOAD utilities use only TIME ZONE.

Archived

The option-list clause supports the options that we list here. Because the meaning and the default values for these options are the same as the options for native SQL procedures that were introduced in DB2 9 for z/OS, we mention only the options names along with a brief description. For a complete description of these options, refer to *DB2 10 for z/OS SQL Reference,* SC19-2983.

- MODIFIES SQL DATA

  Specifies that the function can execute any SQL statement except the statements that are not supported in functions. Do not specify MODIFIES SQL DATA when ALLOW PARALLEL is in effect.

- RETURNS NULL ON NULL INPUT

  Specifies that the function is not invoked and returns the null value if any of the input argument values is null.

- ALLOW PARALLEL or DISALLOW PARALLEL

  Specifies if the function can be run in parallel. The default is DISALLOW PARALLEL, if you specify one or more of the following clauses:

  – NOT DETERMINISTIC
  – EXTERNAL ACTION
  – MODIFIES SQL DATA

  Otherwise, ALLOW PARALLEL is the default.

- ALLOW DEBUG MODE, DISALLOW DEBUG MODE, or DISABLE DEBUG MODE

  Specifies whether this version of the routine can be run in debugging mode. The default is determined using the value of the CURRENT DEBUG MODE special register.

  – ALLOW DEBUG MODE

    Specifies that this version of the routine can be run in debugging mode. When this version of the routine is invoked and debugging is attempted, a WLM environment must be available.

  – DISALLOW DEBUG MODE

    Specifies that this version of the routine cannot be run in debugging mode. You can use an ALTER statement to change this option to ALLOW DEBUG MODE for this initial version of the routine.

  – DISABLE DEBUG MODE

    Specifies that this version of the routine can never be run in debugging mode. This version of the routine cannot be changed to specify ALLOW DEBUG MODE or DISALLOW DEBUG MODE after this version of the routine has been created or altered to use DISABLE DEBUG MODE. To change this option, drop the routine, and create it again using the desired option. An alternative to dropping and recreating the routine is to create a version of the routine that uses the desired option and make that version the active version. When DISABLE DEBUG MODE is in effect, the WLM ENVIRONMENT FOR DEBUG MODE is ignored.

- QUALIFIER schema-name

  Specifies the implicit qualifier that is used for unqualified names of tables, views, indexes, and aliases that are referenced in the routine body. The default value is the same as the default schema.

- PACKAGE OWNER authorization-name

  Specifies the owner of the package that is associated with the first version of the routine. The default value is the SQL authorization ID of the process.

Archived

- ASUTIME

  Specifies the total amount of processor time, in CPU service units, that a single invocation of a routine can run. The default is NO LIMIT. You can specify the limit as an integer value from 1 to 2 147 483 647.

- INHERIT SPECIAL REGISTERS or DEFAULT SPECIAL REGISTERS

  Specifies how special registers are set on entry to the routine. The default is INHERIT SPECIAL REGISTERS.

- WLM ENVIRONMENT FOR DEBUG MODE name

  Specifies the WLM (workload manager) application environment that is used by DB2 when debugging the routine. The default is the WLM-established stored procedure address space specified at installation time.

- CURRENT DATA YES or CURRENT DATA NO

  Specifies whether to require data currency for read-only and ambiguous cursors when the isolation level of cursor stability is in effect. It also controls whether block fetch can be used for distributed, ambiguous cursors. The default is CURRENT DATA NO.

- DEGREE

  Specifies whether to attempt to run a query using parallel processing to maximize performance. The default value is 1 (parallel processing not used).

- DYNAMICRULES

  Specifies the values that apply, at run time, for the following dynamic SQL attributes:

  – The authorization ID that is used to check authorization
  – The qualifier that is used for unqualified objects
  – The source for application programming options that DB2 uses to parse and semantically verify dynamic SQL statements
  – Whether dynamic SQL statements can include GRANT, REVOKE, ALTER, CREATE, DROP, and RENAME statements

  In addition to the value of the DYNAMICRULES clause, the runtime environment of a routine controls how dynamic SQL statements behave at run time. The combination of the DYNAMICRULES value and the runtime environment determines the value for the dynamic SQL attributes. That set of attribute values is called the *dynamic SQL statement behavior*. You can specify the following values:

  – RUN

    Specifies that dynamic SQL statements are to be processed using run behavior. This value is the default value.

  – BIND

    Specifies that dynamic SQL statements are to be processed using bind behavior.

  – DEFINEBIND

    Specifies that dynamic SQL statements are to be processed using either define behavior or bind behavior.

  – INVOKEBIND

    Specifies that dynamic SQL statements are to be processed using either invoke behavior or bind behavior.

Archived

– INVOKERUN

Specifies that dynamic SQL statements are to be processed using either invoke behavior or run behavior.

- APPLICATION ENCODING SCHEME

  Specifies the default encoding scheme for SQL variables in static SQL statements in the routine body. The value is used for defining a SQL variable in a compound statement if the CCSID clause is not specified as part of the data type, and the PARAMETER CCSID routine option is not specified.

- WITH EXPLAIN or WITHOUT EXPLAIN

  Specifies whether information will be provided about how SQL statements in the routine will execute. The default value is WITHOUT EXPLAIN.

- WITH IMMEDIATE WRITE or WITHOUT IMMEDIATE WRITE

  Specifies whether immediate writes are to be done for updates that are made to group buffer pool dependent page sets or partitions. This option is applicable only for data sharing environments. The IMMEDWRITE subsystem parameter has no affect on this option. The default value is WITHOUT IMMEDIATE WRITE.

- ISOLATION LEVEL RR, RS, CS, or UR

  Specifies how far to isolate the routine from the effects of other running applications. The default value is ISOLATION LEVEL CS.

- OPTHINT string-constant

  Specifies whether query optimization hints are used for static SQL statements that are contained within the body of the routine. The default value is an empty string, which indicates that the DB2 subsystem does not use optimization hints for static SQL statements.

- SQL PATH

  Specifies the SQL path that the DB2 subsystem uses to resolve unqualified user-defined distinct types, functions, and procedure names (in CALL statements) in the body of the routine.

- REOPT

  Specifies if DB2 will determine the access path at run time by using the values of SQL variables or SQL parameters, parameter markers, and special registers. The default value is REOPT NONE.

- VALIDATE RUN or VALIDATE BIND

  Specifies whether to recheck, at run time, errors of the type OBJECT NOT FOUND and NOT AUTHORIZED that are found during bind or rebind. The option has no effect if all objects and needed privileges exist. The default value is VALIDATE RUN.

- ROUNDING

  Specifies the desired rounding mode for manipulation of DECFLOAT data. The default value is taken from the DEFAULT DECIMAL FLOATING POINT ROUNDING MODE in DECP.

- DATE FORMAT ISO, EUR, USA, JIS, or LOCAL

  Specifies the date format for result values that are string representations of date or time values.

- DECIMAL(15), DECIMAL(31), DECIMAL(15,s), or DECIMAL(31,s)

  Specifies the maximum precision that is to be used for decimal arithmetic operations.

- FOR UPDATE CLAUSE OPTIONAL or FOR UPDATE CLAUSE REQUIRED

  Specifies whether the FOR UPDATE clause is required for a DECLARE CURSOR statement if the cursor is to be used to perform positioned updates. The default is FOR UPDATE CLAUSE REQUIRED.

- TIME FORMAT ISO, EUR, USA, JIS, or LOCAL

  Specifies the time format for result values that are string representations of date or time values.

- NOT SECURED or SECURED

  Specifies if the function is considered secure for row access control and column access control. The default value is NOT SECURED.

See 6.1.2, “Examples of SQL scalar functions” on page 136 for examples of CREATE FUNCTION.

ALTER FUNCTION (SQL scalar)

Figure 6-4 shows the main portion of the ALTER FUNCTION (SQL scalar) syntax.

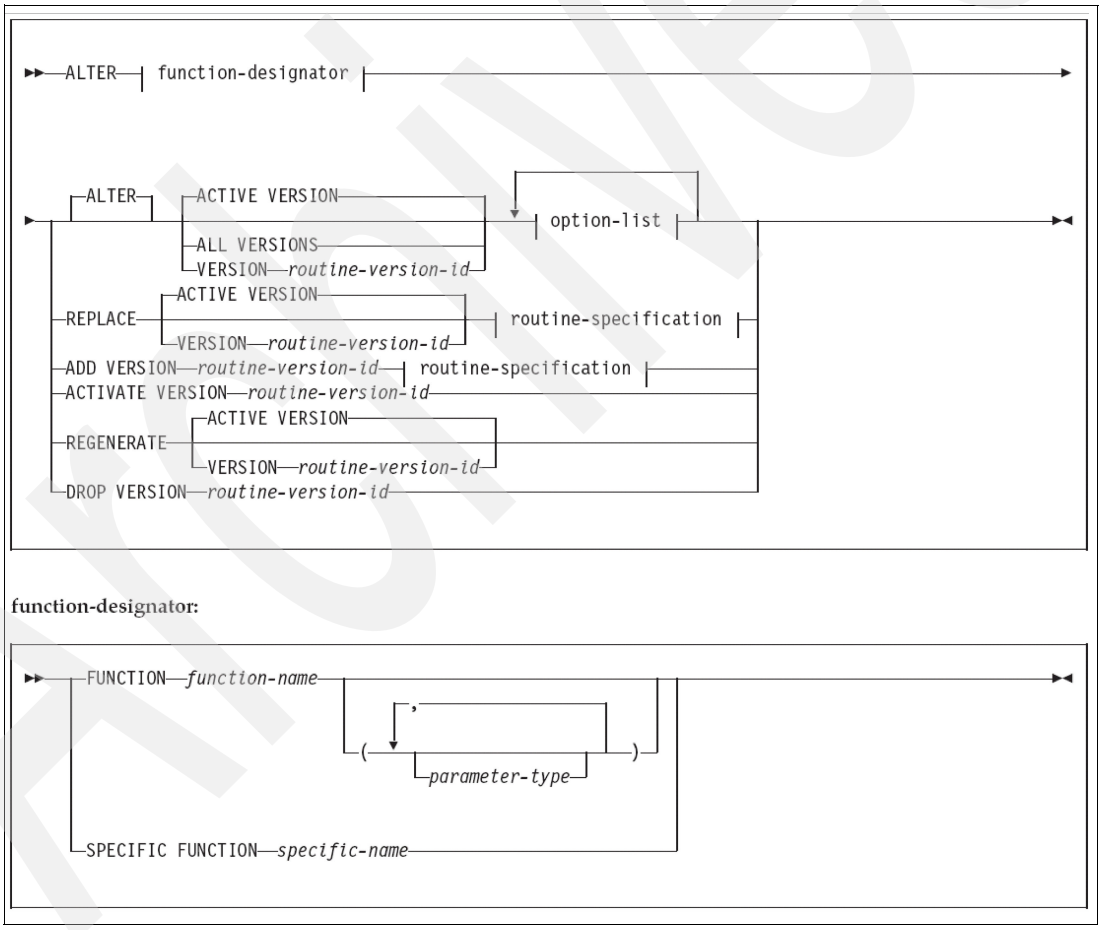


Figure 6-4 ALTER FUNCTION (SQL scalar) syntax diagram: Main

Main enhancements to the ALTER FUNCTION (SQL scalar) statement are related to the function versioning. You can use the ALTER FUNCTION for the following tasks:

- Add new function version
- Replace existing function version

- Regenerate existing function version
- Drop existing function version
- Designate which version should be currently active
- Alter options for all function versions or for a specific function version only

The remaining portion of the ALTER FUNCTION (SQL scalar) syntax, such as option-list, parameter-type, and routine-specification, are the same as for CREATE FUNCTION (SQL scalar) statement. Here, we describe only the changes that we did not discuss in “CREATE FUNCTION (SQL scalar)” on page 128.

When you specify the function to be altered, note that if If the function was defined with a table parameter (such as using the LIKE TABLE name AS LOCATOR clause in the CREATE FUNCTION statement to indicate that one of the input parameters is a transition table), the function signature cannot be used to uniquely identify the function. Instead, you need to use one of the other syntax variations to identify the function with its function name, if unique, or its specific name.

Several ALTER FUNCTION (SQL scalar) clauses allow you to specify the version that you want to change:

- ACTIVE VERSION

  Specifies that the currently active version of the function is to be changed, replaced, or regenerated. If the function is secure, the changed, replaced, or regenerated version remains secure. ACTIVE VERSION is the default.

- VERSION *routine-version-id*

  Identifies the version of the function that is to be changed, replaced, or regenerated. The *routine-version-id* is the version identifier that is assigned when the version of the function is defined. The *routine-version-id* must identify a version of the specified function that exists at the current server. If the function is secure, the changed, replaced, or regenerated version remains secure.

  For the DROP clause this is the only applicable option.

- ALL VERSIONS

  This option is applicable only to the ALTER clause. It specifies that all of the versions of the function are to be changed. SECURED and NOT SECURED are the only options that can be changed when ALL VERSIONS is specified.

The following ALTER FUNCTION (SQL scalar) clauses are supported:

- ALTER

  Specifies that existing version or versions of the function is to be changed. When you change a function using ALTER option-list, any option that is not explicitly specified uses the existing value from the version of the function that is being changed.

- REPLACE

  Specifies that a version of the function is to be replaced. Binding the replaced version of the function might result in a new access path even if the routine body is not being changed.

  When you replace a function, the data types, CCSID specifications, and character data attributes (FOR BIT/SBCS/MIXED DATA) of the parameters must be the same as the attributes of the corresponding parameters for the currently active version of the function. For options that are not explicitly specified, the system default values for those options are used, even if those options were explicitly specified for the version of the function that is being replaced. This is not the case for versions of the function that specified DISABLE

DEBUG MODE. If DISABLE DEBUG MODE is specified for a version of a function, it cannot be changed by using the REPLACE clause.

- ADD VERSION *routine-version-id*

  Specifies that a new version of the function is to be created. The *routine-version-id* is the version identifier for the new version of the function. The *routine-version-id* must not identify a version of the specified function that already exists at the current server.

  When a new version of a function is created, the comment that is recorded in the catalog for the new version is the same as the comment that is in the catalog for the currently active version. When you add a new version of a function, the data types, CCSID specifications, and character data attributes (FOR BIT/SBCS/MIXED DATA) of the parameters must be the same as the attributes of the corresponding parameters for the currently active version of the function. The parameter names can differ from the other versions of the function. For options that are not explicitly specified, the system default values are used. If the function is secure, the new version is considered secure.

- ACTIVATE VERSION *routine-version-id*

  Specifies the version of the function that is to be the currently active version. The *routine-version-id* is the version identifier that is assigned when the version of the function is defined. The version that is specified with the *routine-version-id* is the version that is invoked by a function invocation. The *routine-version-id* value must identify a version of the function that exists at the current server.

- REGENERATE

  Specifies that a version of the function is to be regenerated. When DB2 maintenance is applied that changes how a SQL function is generated, you might need to regenerate the function to process the changes from applying the maintenance.

  REGENERATE rebinds the package (at the current server) that is associated with the non-inline SQL scalar function, including the section that contains the SQL control statement (the SQL PL) and the section or sections for any SQL statements that are contained in the body of the function.

  Note that this rebind is different from doing a REBIND PACKAGE command against the package that is associated with the function. The REBIND PACKAGE rebinds only the section or sections for the SQL statement or statements that are contained in the body of the function and does not modify the section that contains the SQL control statement (the SQL PL).

- DROP

  Drops the version of the function that is identified with the *routine-version-id*. The *routine-version-id* is the version identifier that is assigned when the version is defined. The *routine-version-id* value must identify a version of the function that exists at the current server and must not identify the currently active version of the function. Only the identified version of the function is dropped.

  When only a single version of the function exists at the current server, use the DROP FUNCTION statement to drop the function.

See 6.1.5, “ALTER actions for the SQL scalar functions” on page 141 for examples of ALTER FUNCTION.

6.1.2 Examples of SQL scalar functions

Example 6-1 shows an inline SQL scalar function, KM2MILES, that converts kilometers to miles. It is an inline SQL scalar function because it can be created successfully in prior DB2 releases.

Example 6-1 inline SQL scalar function KM2MILES

```
CREATE FUNCTION KM2MILES(KM DOUBLE)
RETURNS DOUBLE
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
CONTAINS SQL
RETURN KM * 0.621371192;
```

Example 6-2 shows a non-inline function, KM2MILES_SLOW, that converts kilometers to miles. It is a non-inline SQL scalar function because it uses the ALLOW PARALLEL option, which is not available in prior DB2 releases.

Example 6-2 Non-inline function KM2MILES_SLOW using option not previously available

```
CREATE FUNCTION KM2MILES_SLOW(KM DOUBLE)
RETURNS DOUBLE
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
CONTAINS SQL
ALLOW PARALLEL
RETURN KM * 0.621371192;
```

Example 6-3 shows an inline function, TRYXML. Although this function cannot be created successfully in prior DB2 releases, it uses the XML data type. By the exception rule, this SQL scalar function qualifies as inline.

Example 6-3 Inline SQL scalar function TRYXML using XML data type

```
CREATE FUNCTION TRYXML(P1 XML)
RETURNS XML
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
CONTAINS SQL
RETURN XMLCOMMENT('XML Comment');
```

Archived

Example 6-4 shows a non-inline SQL scalar function, TRYSFS. It is a non-inline function because it uses scalar fullselect in the RETURN statement expression, which was not allowed prior to DB2 10.

Example 6-4 Non-inline SQL scalar function TRYSFS using scalar fullselect

```
CREATE FUNCTION TRYSFS(P1 CHAR(6))
RETURNS CHAR(3)
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
CONTAINS SQL
RETURN (SELECT WORKDEPT FROM DSN8A10.EMP WHERE EMPNO = P1);
```

Example 6-5 shows a non-inline SQL scalar function that uses the transition table parameter.

Example 6-5 Non-inline SQL scalar function TRYTBL with transition table parameter

```
CREATE FUNCTION TRYTBL(P1 TABLE LIKE DSN8A10.EMP AS LOCATOR)
RETURNS INT
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
READS SQL DATA
RETURN (SELECT COUNT(*) FROM TABLE(P1 LIKE DSN8A10.EMP));
```

If you use SQL PL in the SQL scalar function, you can specify only a single SQL control statement in the function body; however, that statement can be a compound statement. If you specify a compound statement, it must contain at least one RETURN statement, and a RETURN statement must be executed when the function is invoked. The scoping rules for labels, SQL variables, cursor declarations, and condition names are the same as for native SQL procedures that were introduced with DB2 9 for z/OS. An error is issued if a SQL function calls a procedure and the procedure issues a COMMIT, ROLLBACK, CONNECT, RELEASE, or SET CONNECTION statement.

Considerations for use of terminator characters: SQL processor programs, such as SPUFI, the command line processor, and DSNTEP2, might not correctly parse SQL statements in the routine body that end with semicolons. These processor programs accept multiple SQL statements as input, with each statement separated with a terminator character. Processor programs that use a semicolon as the SQL statement terminator can truncate a CREATE FUNCTION statement with embedded semicolons and pass only a portion of it to DB2. Therefore, you might need to change the SQL terminator character for these processor programs.

If a function is specified in the select-list of a select-statement and if the function specifies EXTERNAL ACTION or MODIFIES SQL DATA, the function is invoked only for each row that is returned. Otherwise, the function might be invoked for rows that are not selected.

Be aware of the usage of date-time special registers in the SQL scalar function body. For inline SQL scalar function (and also for SQL table function), all references to any date-time special register return the same value, and this value is also the same value that is retuned by the special register reference in the statement that invoked the function. Example 6-6 demonstrates this behavior.

Archived

Example 6-6 Special register behavior within the SQL scalar function body

```
-- create inline SQL scalar function that references CURRENT TIMESTAMP
CREATE FUNCTION TMS_INLINE()
RETURNS CHAR(56)
LANGUAGE SQL
NO EXTERNAL ACTION
RETURN
 CHAR(CURRENT TIMESTAMP) ||                          -- get timestamp
 UPPER(LOWER(SPACE(POWER(1,2))||'-- ' )) ||          -- waste some time
 CHAR(CURRENT TIMESTAMP)#                            -- get timestamp
-- create non-inline SQL scalar function that references CURRENT TIMESTAMP
CREATE FUNCTION TMS_NONINLINE()
RETURNS CHAR(56)
LANGUAGE SQL
NO EXTERNAL ACTION
BEGIN
 DECLARE VAR1,VAR2 CHAR(26);
 SET VAR1 = CHAR(CURRENT TIMESTAMP);                 -- get timestamp
 -- some time will pass before the next statement is processed
 SET VAR2 = CHAR(CURRENT TIMESTAMP);                 -- get timestamp
 RETURN VAR1 || ' -- ' || VAR2;
END#
-- now invoke both functions
SELECT CURRENT TIMESTAMP AS Invoker
      ,TMS_INLINE()      AS Inline
      ,TMS_NONINLINE()   AS NonInline
FROM  SYSIBM.SYSDUMMY1#
-- result:
-- Invoker:   2010-08-01-17.30.01.295158
-- Inline:    2010-08-01-17.30.01.295158 -- 2010-08-01-17.30.01.295158
-- NonInline: 2010-08-01-17.30.01.296383 -- 2010-08-01-17.30.01.296468
```

Both references to CURRENT TIMESTAMP within the body of the inline SQL scalar function return the same value as the CURRENT TIMESTAMP reference in the invoking statement. In addition, the two references to CURRENT TIMESTAMP within the body of the non-inline SQL scalar function return different values, which are also different from the CURRENT TIMESTAMP reference in the invoking statement.

6.1.3 SQL scalar function versioning

The extensions for SQL scalar functions include support for versioning and source code management. For the non-inline SQL scalar function, you can:

- Define multiple versions of a SQL scalar function, where one version is considered the active version
- Activate a particular version of a SQL scalar function
- Alter the routine options associated with a version of a SQL scalar function
- Define a new version by specifying the parameter list, routine options, and function body
- Replace the definition of an existing version by specifying the parameter list, routine options, and function body

Archive

- Drop a version of a SQL scalar function
- Fall back to previous version instantly requiring no explicit rebind/recompile

All of these functions can be accomplished using *function versioning* (similar to the SQL native procedure versioning that was introduced in DB2 9 for z/OS). With function versioning, the function can have more than one variation (version), with the function signature being identical across all function versions. Thus, the schema name, the function name, and the number of parameters and their associated data types are the same for all versions of the function. (The names of the parameters do not have to be the same.)

Note that because the specific name of a function refers to a unique function signature, all versions of the function share the same specific name. However, a unique routine identifier is associated with each version of a SQL function, so all versions of a particular function have different routine identifiers.

With function versioning, you use CREATE FUNCTION statement to establish a SQL scalar function as a new object (with initial version) and then use ALTER FUNCTION to define, change, or replace a version of the SQL scalar function. The parameter list, function options, and a function body can be specified as part of the definition of a version of a function (recall that all versions of a SQL scalar function must have identical signature). When adding or modifying a SQL scalar function version you can change just the function options or you can change the entire definition (function options and function body). Note that changing certain attributes of a SQL scalar function can cause packages that refer to the function (that is, invoking applications) to be marked invalid. Additionally, when certain attributes of a SQL scalar function are altered, you might need to rebind or recompile the body of the function.

There can be several versions of a SQL scalar function at any given time at the current server; however, there can be only one *active* version. When you first create a SQL scalar function, that initial version is considered the active version of the function. You can then use the ADD VERSION clause of the ALTER FUNCTION statement to define additional versions and you can use ACTIVATE VERSION clause of the ALTER FUNCTION statement to make any one of those versions the current active version.

When the current active version of the SQL scalar function is changed (using ALTER FUNCTION ACTIVATE VERSION), the new active version is used on the next invocation of the function. For the user issuing the ALTER FUNCTION statement this would be the next invocation of that function. For another user this would be the next invocation of that function after the COMMIT of the ALTER FUNCTION statement that activated the new version of the function.

However, if the invocation of the function is a recursive invocation inside the body of the same function and if the initial invocation used a version that is different from the currently active version, the active version is not used. The version from the initial invocation is used for any recursive invocation until the entire function finishes executing, which preserves the semantics of the version that is used by the original invocation, including the case where the recursive invocation is indirect. For example, if function X invokes function Y, which then invokes function X, the second invocation of function X uses the same version of the function that was active at the time of the original invocation of function X.

The current active version for a SQL scalar function is reflected in the catalog. You can verify the column ACTIVE in SYSIBM.SYSROUTINES table because it has a value of Y for the active version.

Using CURRENT ROUTINE VERSION to override the active version of a SQL scalar function: You can use special register CURRENT ROUTINE VERSION to interactively ask DB2 to execute a routine version other than the one marked as active in the DB2 catalog. However, the enhancements to SQL scalar functions introduced in DB2 10 for z/OS do not make use of this special register. Thus, unlike native SQL procedures, you cannot use the CURRENT ROUTINE VERSION special register to override the active version of a SQL scalar function.

Example 6-7 demonstrates function versioning.

Example 6-7 SQL scalar function versioning example

```
-- Create a non-inline SQL scalar function with initial version V1
CREATE FUNCTION TRYVER()
RETURNS VARCHAR(20)
VERSION V1
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
CONTAINS SQL
RETURN 'Running version1';
-- Add a second version V2 of the function created above
ALTER FUNCTION TRYVER ADD VERSION V2 ()
RETURNS VARCHAR(20)
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
CONTAINS SQL
RETURN 'Running version2';
-- Invoke the function
SELECT TRYVER() FROM SYSIBM.SYSDUMMY1;
-- result: Running version1
-- Now make V2 version the active version
ALTER FUNCTION TRYVER ACTIVATE VERSION V2;
-- Invoke the function again
SELECT TRYVER() FROM SYSIBM.SYSDUMMY1;
-- result: Running version2
```

6.1.4 Deploying non-inline SQL scalar functions

Extensions to the DB2 for z/OS bind commands are provided to help with the deployment of non-inline SQL scalar functions to multiple remote servers or within the same server. This function allows you to introduce a thoroughly tested SQL scalar function to a wider community while keeping the same generated logic in the function. You can change certain options on the newly created SQL scalar function to better match the deploying environment. You can use the BIND DEPLOY command to deploy a non-inline SQL scalar function to a target location.

When deploying a non-inline SQL scalar function, if a function with the same target name does not exist at the target location, the deployed function is created as a new function at the target location. If a function with the same target name already exists at the target location, the deployed function is either added as a new version of the function, or the deployed function is used to replace an existing version of the function, depending on the ACTION option of the BIND DEPLOY command used.

Options for the BIND DEPLOY command: If you do not specify the QUALIFIER and OWNER options with the BIND DEPLOY command, the default values are the value of the authorization ID that issues the BIND DEPLOY command.

Example 6-8 shows a deployment scenario where a non-inline SQL scalar function TEST.TRYDEP is created on a local server and then deployed to another server at location EKATERINBURG. After the BIND DEPLOY command is completed, the function TEST.TRYDEP is available at location EKATERINBURG.

Example 6-8 Deployment of a SQL scalar function

```
CREATE FUNCTION TEST.TRYDEP()
RETURNS INT
LANGUAGE SQL
VERSION V1
DETERMINISTIC
NO EXTERNAL ACTION
RETURN 42;
-- issue BIND DEPLOY command to deploy the function
BIND PACKAGE(EKATERINBURG.TEST) DEPLOY(TEST.TRYDEP) COPYVER(V1) ACTION(ADD)
```

6.1.5 ALTER actions for the SQL scalar functions

This section includes examples of the different ALTER actions for the SQL scalar functions. Example 6-9 shows the initial definition of the SQL scalar function PRIVET that we use in the examples that follow.

Example 6-9 Initial definition of the function for the running example

```
CREATE FUNCTION PRIVET()
  RETURNS VARCHAR(26)
  LANGUAGE SQL
  VERSION V1
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURN 'Hello to Alina';
```

Example 6-10 shows modification of the option-list clause for the function, changing the DEBUG MODE option.

Example 6-10 ALTER FUNCTION (SQL scalar) ALTER option-list

```
ALTER FUNCTION PRIVET() ALTER DISABLE DEBUG MODE;
```

Archive

Example 6-11 shows addition of the function version.

Example 6-11 ALTER FUNCTION (SQL scalar) ADD VERSION

```
ALTER FUNCTION PRIVET() ADD VERSION V2
  ()
  RETURNS VARCHAR(26)
  LANGUAGE SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURN 'Hello to Bridget';
```

Example 6-12 shows activation of the function version.

Example 6-12 ALTER FUNCTION (SQL scalar) ACTIVATE VERSION

```
ALTER FUNCTION PRIVET() ACTIVATE VERSION V2;
```

Example 6-13 shows replacement of the function version.

Example 6-13 ALTER FUNCTION (SQL scalar) REPLACE

```
ALTER FUNCTION PRIVET() REPLACE VERSION V2
  ()
  RETURNS VARCHAR(26)
  LANGUAGE SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURN 'Hello to Nicole';
```

Example 6-14 shows regeneration of the currently active version of the function.

Example 6-14 ALTER FUNCTION (SQL scalar) REGENERATE

```
ALTER FUNCTION PRIVET() REGENERATE VERSION V2;
```

Finally, Example 6-15 shows removal of one of the versions of the function.

Example 6-15 ALTER FUNCTION (SQL scalar) DROP

```
ALTER FUNCTION PRIVET() DROP VERSION V1;
```

Typically, the changes made by ALTER FUNCTION take effect immediately. That is, the changed function definition is used the next time the function is invoked. However, ALTER FUNCTION is locked out from making the changes if the function is in use. In that case, the ALTER FUNCTION takes effect after the query that references the function is completed. Note that the query can have multiple references to the function or can invoke the function multiple times for processing multiple rows. Therefore, ALTER FUNCTION waits until the entire query completes, not an individual function invocation.

When a version of a non-inline SQL scalar function is altered to change any option that is specified for the active version, all the packages that refer to that function are marked invalid. Additionally, when certain attributes of the function are changed, the body of the function might be rebound or regenerated.

Archived

Table 6-1 summarizes the effect of changing the specific function option. A value of *Y* in a row indicates that a rebind or regeneration occurs if the option is changed.

Table 6-1 CREATE and ALTER FUNCTION options resulting in rebind when changed

CREATE FUNCTION or ALTER FUNCTION option	Change requires rebind of invoking applications	Change results in implicit rebind of the non-control statements of the function body	Change results in implicit regeneration of the entire function body
ALLOW DEBUG MODE, DISALLOW DEBUG MODE or DISABLE DEBUG MODE [a] [b]	Y	Y	Y
ALLOW PARALLEL, DISALLOW PARALLEL		Y	
APPLICATION ENCODING SCHEME	Y	Y	Y
ASUTIME	Y		
CURRENTDATA		Y	
DATE FORMAT	Y	Y	Y
DECIMAL	Y	Y	Y
DEGREE		Y	
DYNAMICRULES		Y	
FOR UPDATE CLAUSE OPTIONAL or FOR UPDATE CLAUSE REQUIRED	Y	Y	Y
INHERIT SPECIAL REGISTERS, or DEFAULT SPECIAL REGISTERS	Y		
ISOLATION LEVEL		Y	
MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL	Y	Y	Y
NOT DETERMINISTIC, or DETERMINISTIC			
OPTHINT		Y	
PACKAGE OWNER		Y	
QUALIFIER		Y	
REOPT		Y	
ROUNDING	Y	Y	Y
SQL PATH		Y	
TIME FORMAT	Y	Y	Y
VALIDATE RUN or VALIDATE BIND		Y	
WITH EXPLAIN, or WITHOUT EXPLAIN		Y	
WITH IMMEDIATE WRITE, or WITHOUT IMMEDIATE WRITE		Y	
WLM ENVIRONMENT FOR DEBUG MODE	Y		

a. Invoking applications are invalidated if a value of DISALLOW DEBUG MODE is changed to DISABLE DEBUG MODE.
b. The function package is rebound or regenerated if a value of ALLOW DEBUG MODE is changed to DISALLOW DEBUG MODE.

Archived

6.2 Support for SQL table functions

Prior DB2 releases included table function support for *external* table functions only. With DB2 10 for z/OS new-function mode, you can create SQL table functions. With external table function, the body of the function is written in the host language (such as COBOL, PL/I, or another programming language. With new-function mode, the body of the SQL table function is written entirely in SQL PL. DB2 10 for z/OS introduces support for simple SQL table functions in the sense that you can use a single RETURN control statement in the function body. When you define a SQL table function you can:

- Define parameter as a distinct type
- Define parameter for a transition table (TABLE LIKE ... AS LOCATOR)
- Include a single RETURN SQL control statement that returns a result table

6.2.1 SQL table functions syntax changes

Here, we describe some relevant changes to the SQL syntax. For complete information about the syntax, refer to *DB2 10 for z/OS SQL Reference,* SC19-2983.

CREATE FUNCTION (SQL table)

Figure 6-5 shows the main portion of the CREATE FUNCTION (SQL scalar) syntax.

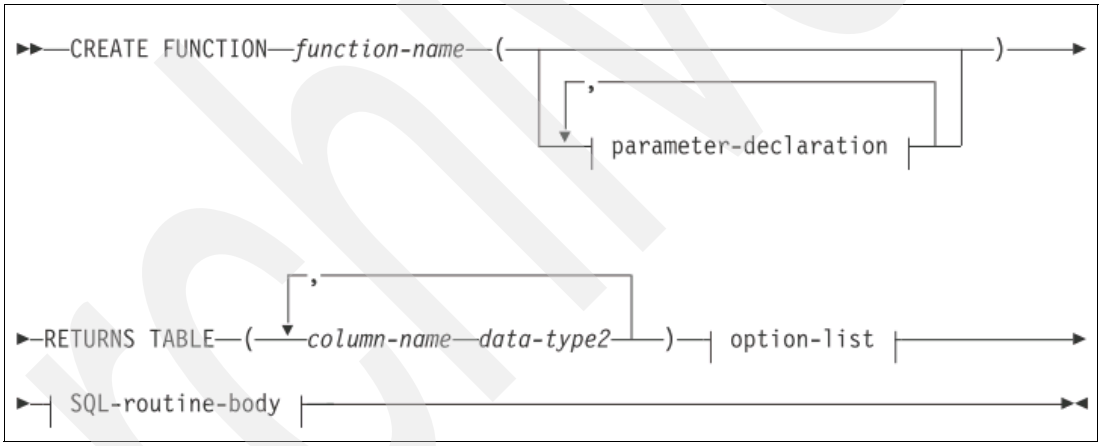


Figure 6-5 CREATE FUNCTION (SQL table) syntax diagram: Main

Archived

The parameter-declaration is the same as for “CREATE FUNCTION (SQL scalar)”. Here we describe only the syntax that is specific to SQL table functions. The RETURNS TABLE clause specifies that the result of the function is a table and lists the output table column names and data types. All columns of the output table are nullable. Figure 6-6 shows the option-list.

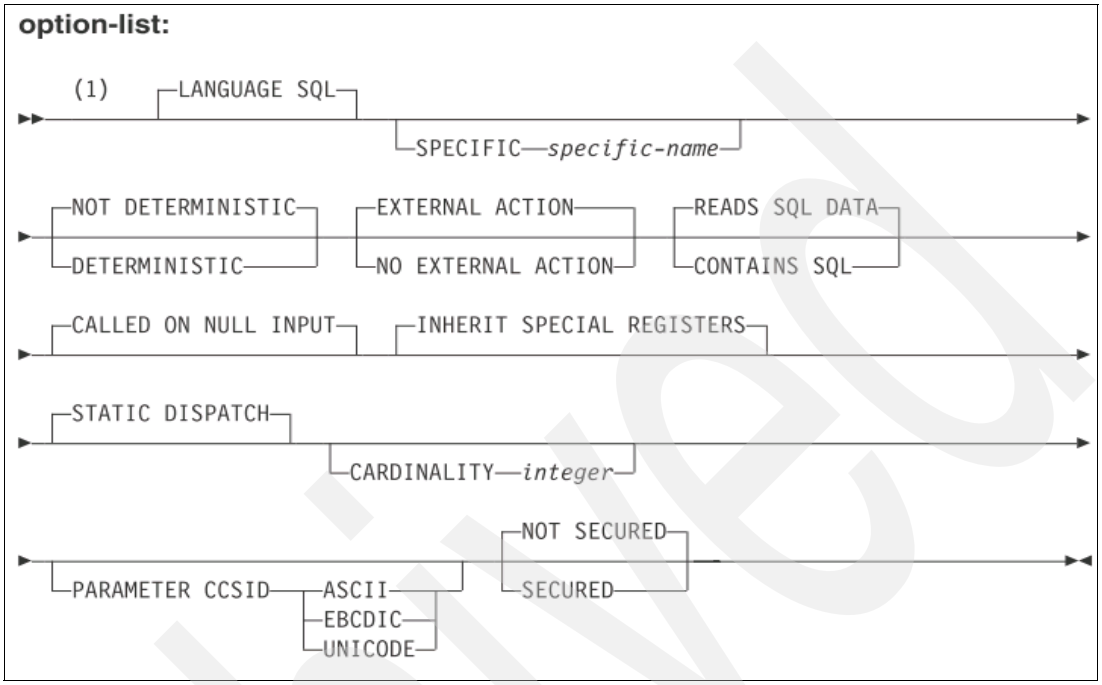


Figure 6-6 CREATE FUNCTION (SQL table): Options

CARDINALITY specifies an estimate of the expected number of rows that the function returns. The number is used for optimization purposes. The value of integer must be between 0 and 2147483647. The default value is the same value that DB2 uses when the RUNSTATS utility does not generate statistics for a table. Figure 6-7 shows the SQL-routine-body part of the CREATE FUNCTION (SQL table).

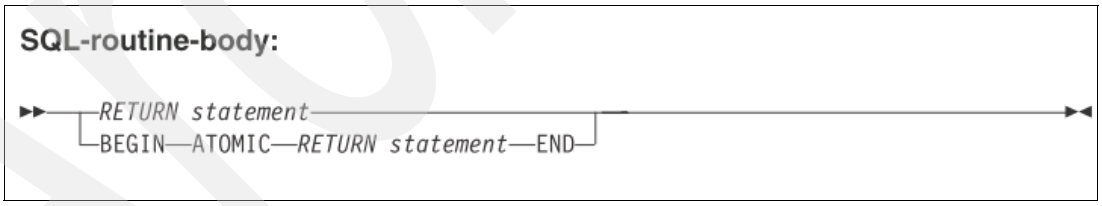


Figure 6-7 CREATE FUNCTION (SQL table) syntax: SQL-routine-body

The BEGIN ATOMIC ... END variation is supported for compatibility; however, even with this syntax variation, you can specify nothing but the RETURN statement. The RETURN statement is the RETURN SQL control statement, and Figure 6-8 shows the corresponding syntax. Note that because a SQL table function must return a table, only the fullselect option of the RETURN SQL control statement is allowed. That is, you cannot specify NULL or an expression for the RETURN of the SQL table function. (You can specify scalar fullselect.)

Archived

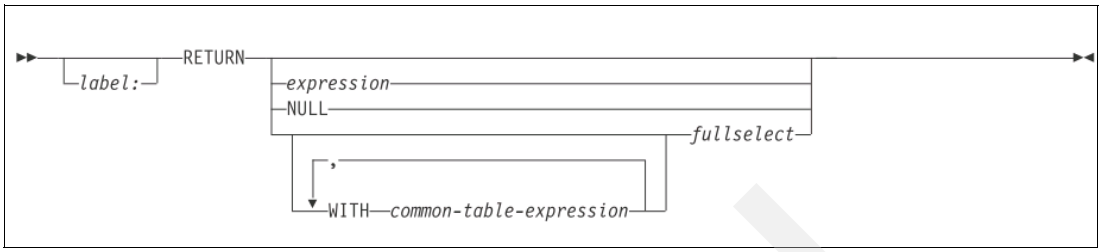


Figure 6-8 RETURN SQL control statement

ALTER FUNCTION (SQL table)

Figure 6-9 shows the main portion of the ALTER FUNCTION (SQL table) statement.

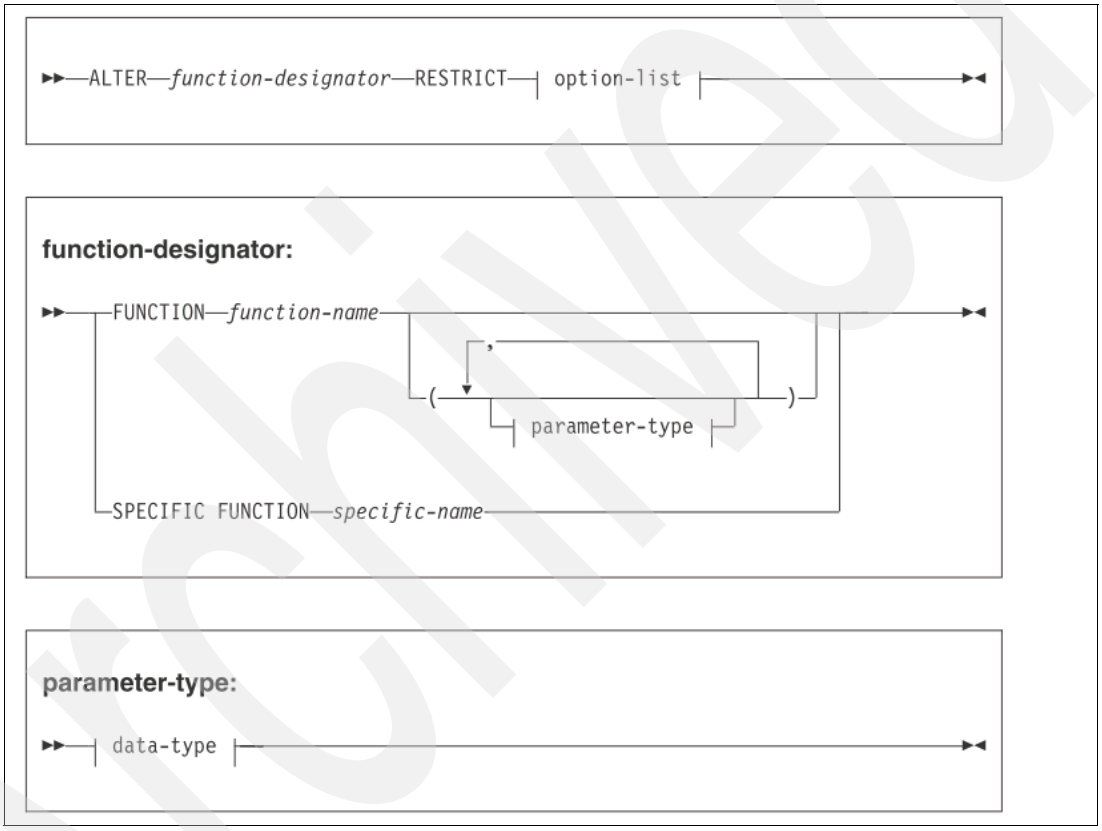


Figure 6-9 ALTER FUNCTION (SQL table): Main

When you specify the function to be altered, note that if If the function was defined with a table parameter (such as the LIKE TABLE name AS LOCATOR clause was specified in the CREATE FUNCTION statement to indicate that one of the input parameters is a transition table), you cannot use the function signature to uniquely identify the function. Instead, you need to use one of the other syntax variations to identify the function with its function name, if unique, or its specific name.

RESTRICT indicates that the function is not altered or replaced it if is referenced by any function, materialized query table, procedure, trigger, or view.

You can specify the following options on the ALTER FUNCTION (SQL table):

- NOT DETERMINISTIC or DETERMINISTIC
- EXTERNAL ACTION or NO EXTERNAL ACTION

Archived

- READS SQL DATA or CONTAINS SQL
- CALLED ON NULL INPUT
- INHERIT SPECIAL REGISTERS
- STATIC DISPATCH
- CARDINALITY integer
- SECURED or NOT SECURED

Note that there are no defaults for these options. The default is the prior value of the option.

When a SQL table function is altered, all packages that refer to that function are marked invalid.

To provide compatibility with previous releases of DB2 or other products in the DB2 family, DB2 supports the following keywords:

- VARIANT as a synonym for NOT DETERMINISTIC
- NOT VARIANT as a synonym for DETERMINISTIC
- NULL CALL as a synonym for CALLED ON NULL INPUT

6.2.2 Examples of CREATE and ALTER SQL table functions

Example 6-16 shows a SQL table function TRYTABLE that returns a table.

Example 6-16 SQL table function definition and invocation

```
CREATE FUNCTION TRYTABLE(P1 CHAR(3))
RETURNS TABLE(FIRSTNAME VARCHAR(12), LASTNAME VARCHAR(15))
RETURN SELECT FIRSTNME, LASTNAME FROM DSN8A10.EMP WHERE WORKDEPT = P1;

SELECT * FROM TABLE(TRYTABLE('A00')) X;
-- result:
+-------------------------------+
|  FIRSTNAME   |    LASTNAME    |
+-------------------------------+
| CHRISTINE    | HAAS           |
| VINCENZO     | LUCCHESI       |
| SEAN         | O'CONNELL      |
| DIAN         | HEMMINGER      |
| GREG         | ORLANDO        |
+-------------------------------+
```

No package is created for a SQL table function, and any reference to the function within a SQL statement is replaced with the RETURN-statement specified on the CREATE FUNCTION (SQL table), which is similar to inline SQL scalar function processing. In fact, the INLINE column in the SYSIBM.SYSROUTINES catalog table contains a value of Y for the row that is associated with SQL table function.

Example 6-17 shows a modification to the TRYTABLE function definition (from Example 6-16) to set the estimated number of rows that are returned to 100 and to designate the function as deterministic.

Example 6-17 ALTER FUNCTION (SQL table)

```
ALTER FUNCTION TRYTABLE(P1 CHAR(3)) RESTRICT
 CARDINALITY 100
 DETERMINISTIC;
```

6.3 Enhanced support for native SQL procedures

DB2 9 for z/OS introduced native SQL procedures. DB2 10 for z/OS new-function mode provides the following enhancements:

- Ability to define a parameter or SQL variable as a distinct type (same capability exists for SQL functions)
- CREATE and ALTER statements for a SQL procedure can be embedded in an application program
- There are also general performance improvements for SQL PL processing, which can provide up to 20% CPU time improvement for certain workloads compared to DB2 9.

DB2 10 for z/OS also enhances the SQL control assignment statement by allowing multiple assignments. Figure 6-10 shows the assignment clause syntax.

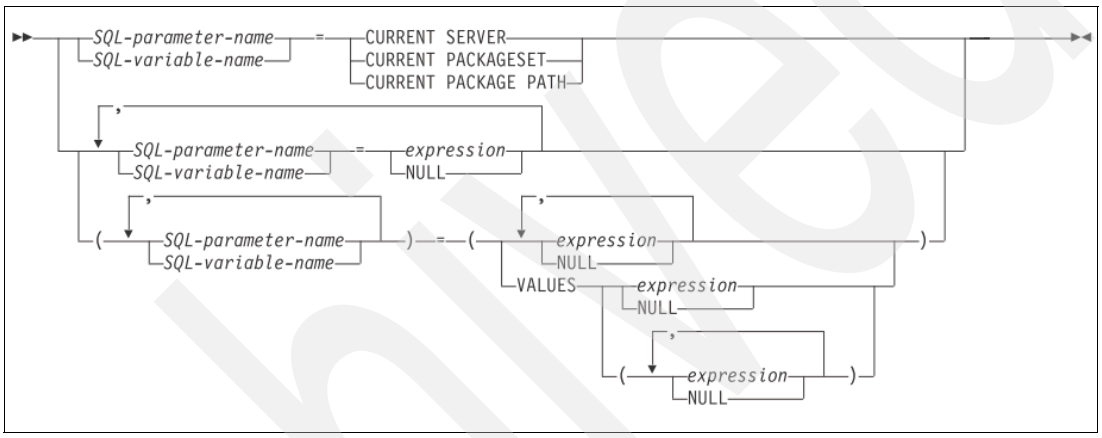


Figure 6-10 Enhanced assignment (SET) statement

Example 6-18 shows use of the distinct type for the parameter and SQL variable.

Example 6-18 Distinct type in SQL procedure

```
CREATE TYPE TEMPERATURE AS DECIMAL(5,2)#
CREATE PROCEDURE TRYPROC(IN P1 TEMPERATURE, OUT P2 TEMPERATURE)
LANGUAGE SQL
  BEGIN
    DECLARE VAR1 TEMPERATURE;
    SET VAR1 = P1, P2 = VAR1;
  END#
```

APARs PM13525 and PM13631 provide functions for SQL procedures.

6.4 Extended support for implicit casting

DB2 10 for z/OS extends the compatibility rule for character or graphic string and numeric data types. Starting with DB2 10 new-function mode, the character or graphic strings and numeric data types are compatible, and DB2 performs an implicit cast between the values of those data types.

In this section, we provide information about the implicit casting between numeric data types and character or graphic strings and illustrate this behavior with examples.

DB2 performs implicit casting in many cases. For example, when inserting an INTEGER value into a DECIMAL column, the INTEGER value is implicitly cast to DECIMAL data type. Therefore, you can assign character or graphic sting value directly to a target with numeric data type or compare it with numeric value. Conversely, you can assign numeric value directly to a target of a character or graphic string data type.

Implicit casting note: An *explicit* cast allows casting between graphic strings and numeric data types, whereas an *implicit* cast between graphic strings and numeric data types supports only Unicode graphic strings.

Also note that LOB data types are not supported for implicit cast with numeric data types (just as the explicit cast between LOBs and numerics is not supported). Thus, in this book when we say *character* or *graphic string*, it is implied no LOBs and only Unicode for the graphic string.

When DB2 implicitly casts a numeric value to a string value, the target data type is VARCHAR, which is then compatible with other character or graphic string data types. The length attribute and the CCSID attribute of the result are determined the same way as the VARCHAR built-in scalar function.

When DB2 implicitly casts a character or graphic string to a numeric value, the target data type is DECFLOAT(34), which is then compatible with other numeric data types. The reason DECFLOAT(34) was chosen for the target data type is that a valid string representation of any numeric value (integer, decimal, floating-point, or other numeric value) is also a valid representation of a DECFLOAT(34) number, but not the other way around. In other words, the set of all string representations of a DECFLOAT(34) is a proper superset of the set of all string representations of other numeric data types.

Table 6-2 shows the implicit cast target data types and length.

Table 6-2 Implicit cast target data types and length

Source data type	Target data type
SMALLINT	VARCHAR(6)
INTEGER	VARCHAR(11)
BIGINT	VARCHAR(20)
DECIMAL(p,s)	VARCHAR(p+2)
REAL, FLOAT, DOUBLE	VARCHAR(24)
DECFLOAT	VARCHAR(42)
CHAR, VARCHAR, GRAPHIC, VARGRAPHIC	DECFLOAT(34)

Implicit cast from numeric value to a string value can happen during the following operations:

- Assignment (where the source value is a number and the target operand is a character string or graphic string data type)

  Among assignment statements, implicit casting is not supported for SET statements for special registers, RETURNS clause and RETURN statement for functions, and the SQL following control statements:

  - RETURN
  - SIGNAL
  - RESIGNAL

- Processing of concatenation operators (CONCAT and ||).

6.4.1 Examples of implicit casting

Example 6-19 demonstrates how the values of different numeric data types are cast to a character string during concatenation operation.

Example 6-19 Implicit cast from numeric to string

```
SELECT 'The answer is: ' || INT(42)
       || ' = ' || DOUBLE(42)
       || ' = ' || DEC(42, 8,3)
       || ' = ' || DECFLOAT(42)
FROM   SYSIBM.SYSDUMMY1;
-- result: The answer is: 42 = 4.2E1 = 42.000 = 42
```

Implicit cast from a string value to a numeric value can happen during the following operations:

- Assignments

  Where the source value is a character string or graphic string, and the target operand is a numeric data type.

- Comparisons

  When a character string or graphic string value is compared with a numeric value, DB2 implicitly converts the string value to DECFLOAT(34) and applies numeric comparison rule between the DECFLOAT(34) value and the other numeric value.

- Arithmetic operators (unary[1] arithmetic operators + and - and infix[2] arithmetic operators +, -, *, and /)

  If the operand of unary arithmetic operators is of a character string or graphic string data type, that operand is implicitly cast to DECFLOAT(34). For infix arithmetic operators, if one operand is a numeric value or both operands are character or graphic string values, DB2 implicitly casts the character string or graphic string operand to the DECFLOAT data type.

Example 6-20 demonstrates how the values of character string are cast to numeric values. For the first query, the string '1' is converted to DECFLOAT(34), so the operands for the arithmetic operation become DECFLOAT(34) and INTEGER. According to the numeric operation rules, the integer operand is converted to DECFLOAT, and the result data type of the arithmetic operation is DECFLOAT(34).

[1] Unary operators require only one operand. They perform operations such as incrementing or decrementing a value by one, negating an expression, or inverting the value of a boolean.

[2] In terms of operator position, an operator can be prefix, a postfix, or an infix, depending on whether it precedes or succeeds its operand or sits between its left and right operands.

For the second query, comparison between CHAR(6) column EMPNO and an integer constant 10 results in the character string value of the column to be converted to DECFLOAT. Thus, the comparison becomes between DECFLOAT and INTEGER, which leads to the integer 10 being converted to DECFLOAT, and then a comparison of two DECFLOAT values is performed.

For the third query, there are two integer operands and one string operand, the result data type is, therefore, a DECFLOAT.

The third query demonstrates the ability to provide string operands for built-in functions that expect integer operands.

Example 6-20 Implicit cast from string to numeric

```
SELECT '1' + 1
FROM   SYSIBM.SYSDUMMY1;
-- result: +2.000000000000000000000000E+0000
SELECT FIRSTNME, LASTNAME, EMPNO
FROM   DSN8A10.EMP
WHERE  EMPNO = 10;                    -- CHAR(6) = INT
-- result: CHRISTINE HAAS 000010
SELECT MAX(1, 2, '3')
FROM   SYSIBM.SYSDUMMY1;
-- result: +3.000000000000000000000000E+0000
SELECT SUBSTR('MALINKA', '2', '6')
FROM   SYSIBM.SYSDUMMY1;
-- result: ALINKA
```

Using implicit cast from strings to DECIMAL data: Exercise caution when dealing with implicit cast from strings to DECIMAL data. Conversion from DECFLOAT to DECIMAL performs rounding, if necessary. This function is different from implicit casting to other numeric data types where truncation is performed instead and is even different from the explicit cast from string to DECIMAL where truncation is performed. Furthermore, the rounding depends on the CURRENT DECFLOAT ROUNDING MODE special register setting.

Thus, keep in mind that an implicit cast from a string value to DECIMAL target is essentially equivalent to the explicit cast from string to DECFLOAT(34) followed by the explicit cast from DECFLOAT(34) to DECIMAL. For example, an implicit cast from '1.9' to DECIMAL(3,0) is equivalent to expression DECIMAL(DECFLOAT('1.9'), 3, 0), which is semantically not equivalent to DECIMAL('1.9', 3, 0). A value produced in the first case depends on the setting of the DECFLOAT rounding mode, and the value produced in the second case is always 1. By default, the DECFLOAT rounding mode is ROUND_HALF_EVEN, which rounds the value from '1.9' to 2.

Similar to the explicit cast between string and DECIMAL, the casts (implicit or explicit) between DECIMAL and other non-DECFLOAT numeric data types follow the truncation semantics. For example, DECIMAL(19.2, 3, 0) results in 1.

If you need the implicit cast behavior between strings and DECIMAL to match the truncation semantics of the explicit cast between strings and DECIMAL (or to match the truncation semantics of the casts between DECIMAL and other non-DECFLOAT numeric data types), ensure that the DECFLOAT rounding mode is set to ROUND_DOWN (which means truncation).

Example 6-21 demonstrates this aspect. We insert the same values into both integer and zero-scale decimal columns and we observe truncation for the integer and rounding for the decimal. Furthermore, rounding for decimal depends on the CURRENT DECFLOAT ROUNDING MODE special register setting. For example, when ROUND_HALF_DOWN mode is in effect, both values '1.9' and '2.5' are rounded to 2. When ROUND_HALF_UP mode is in effect, the value '1.9' is rounded to 2 and the value of '2.5' is rounded to 3. When ROUND_DOWN mode is in effect, the value '1.9' is rounded (truncated) to 1and the value '2.5' is rounded (truncated) to 2.

Example 6-21 Rounding with implicit cast from string to decimal

```
CREATE TABLE TRYIMPCAST(ID INT, C1 SMALLINT, C2 DEC(5,0));

SET CURRENT DECFLOAT ROUNDING MODE = ROUND_HALF_DOWN;
INSERT INTO TRYIMPCAST VALUES(1,'1.9','1.9'); -- will insert 1, 1, 2
INSERT INTO TRYIMPCAST VALUES(2,'2.5','2.5'); -- will insert 2, 2, 2
-- change rounding mode
SET CURRENT DECFLOAT ROUNDING MODE = ROUND_HALF_UP;
INSERT INTO TRYIMPCAST VALUES(3,'1.9','1.9'); -- will insert 3, 1, 2
INSERT INTO TRYIMPCAST VALUES(4,'2.5','2.5'); -- will insert 4, 2, 3
-- change rounding mode
SET CURRENT DECFLOAT ROUNDING MODE = ROUND_DOWN;
INSERT INTO TRYIMPCAST VALUES(5,'1.9','1.9'); -- will insert 5, 1, 1
INSERT INTO TRYIMPCAST VALUES(6,'2.5','2.5'); -- will insert 6, 2, 2

SELECT * FROM TRYIMPCAST;
-- result: ID  C1  C2
--         1   1   2
--         2   2   2
--         3   1   2
--         4   2   3
--         5   1   1
--         6   2   2
```

6.4.2 Rules for result data types

Numeric is the dominant data type. If one operand is a numeric data type and the other operand is a character or graphic string data type, the data type of the result is DECFLOAT(34). The string operand is implicitly cast to DECFLOAT(34). The rules apply to set operations (UNION, INTERSECT, and EXCEPT), CASE expressions, scalar functions (COALESCE, VALUE, IFNULL, NULLIF, scalar MAX, and scalar MIN), and IN list of IN predicates.

6.4.3 Function resolution

Because numeric data types and character or graphic strings are now compatible, you can invoke a function that expects a numeric value with a character or graphic string argument. Conversely, you can invoke a function that expects a character or graphic string with a numeric argument. Therefore, DB2 includes additional changes to the function resolution rules.

For example, if you invoke a function that expects an INTEGER with a character string argument, DB2 casts down DECFLOAT(34) to an INTEGER, because during an implicit cast, character strings are converted to DECFLOAT(34). Thus, the following implicit casting is supported for function resolution:

- A numeric data type can be cast to a character or graphic string data type.
- A character or graphic string data type can be cast to a numeric data type.
- A numeric value can be cast to a value of another numeric data type that is not in the data type promotion chain list for the source data type, allowing for casting a numeric value to a numeric data type that is lower in the promotion chain. This extension for implicit casting applies to both built-in and user-defined functions.

  For example, built-in scalar function LEFT expects second argument to be integer. You can invoke this function with a character string, for example LEFT('ABCDEFG', '3'), in which case, the string '3' is implicitly cast to DECFLOAT(34) and then assigned to an INTEGER value.
- A varying length character string data type can be cast to a fixed length character data type.
- A character or graphic string data type can be cast to a date, time, or timestamp data type.

Example 6-22 illustrates these implicit casts for function resolution. This example creates simple SQL scalar functions and then invokes those functions with arguments of various data types. None of these invocations would successfully resolve to the desired functions in prior releases (resulting in a SQLCODE -440); however, all of these invocations resolve successfully in DB2 10 for z/OS new-function mode.

Example 6-22 Implicit cast with function resolution

```
CREATE FUNCTION F1(P1 INT)
RETURNS INT
LANGUAGE SQL
RETURN P1;
SELECT F1(BIGINT(1))                -- BIGINT to INTEGER
FROM   SYSIBM.SYSDUMMY1;
-- result: 1
SELECT F1('201')                    -- CHAR to DECFLOAT to INTEGER
FROM   SYSIBM.SYSDUMMY1;
-- result: 201

CREATE FUNCTION F2(P1 TIMESTAMP)
RETURNS TIMESTAMP
LANGUAGE SQL
RETURN P1;
SELECT F2('2010-08-06 12:11:00')    -- VARCHAR to TIMESTAMP
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-06-12.11.00.000000

CREATE FUNCTION F3(P1 CHAR(3))
RETURNS CHAR(3)
LANGUAGE SQL
RETURN P1;
SELECT F3('201')                    -- VARCHAR to CHAR
FROM   SYSIBM.SYSDUMMY1;
-- result: 201
SELECT F3(201)                      -- INTEGER to VARCHAR to CHAR
FROM   SYSIBM.SYSDUMMY1;
-- result: 201
```

To avoid costly DECFLOAT processing, DB2 optimizes certain implicit casts by casting directly to the target data type (without casting to intermediate DECFLOAT value first). For example, when inserting a character string into an INTEGER column, DB2 first attempts to cast the string value directly to INTEGER and only if it fails does the cast to DECFLOAT occur. For example, when inserting a string value '1.0' into DECIMAL(5,0) column, the value is cast to DECIMAL(5,0) directly. The cast succeeds because '1.0' is a valid representation for DECIMAL(5,0), and a decimal value 1.0 is inserted. In addition, when inserting a string value '1.9' into DECIMAL(5,0) column, the value is cast to DECIMAL(5,0) directly. The cast fails because '1.9' is not a valid representation for DECIMAL(5,0), and a string value of '1.9' is cast to DECFLOAT(34). Then, the DECFLOAT value 1.9 is assigned to DECIMAL(5,0), resulting in 2.0 being inserted.

6.5 Support for datetime constants

DB2 10 for z/OS introduces support for the ANSI/ISO SQL standard form of a datetime constant. Prior to DB2 10 for z/OS, you had to use a character string constant of a particular format to specify a datetime constant. In other words, there was no concept of a datetime constant per se. For example, a date constant '1977-08-01' is really the character string constant that just happened to have a valid date representation.

With DB2 10 for z/OS new-function mode, you can specifically denote the constant as a datetime constant instead of a character string constant. This enhancement helps when porting existing applications from other database systems that support datetime constants. Table 6-3 shows the format for the ANSI/ISO SQL standard datetime constants.

Table 6-3 Datetime constant formats

Data type	Format
DATE	DATE *string-constant* The *string-constant* value must conform to one of the formats listed in Table 6-4 on page 155, subject to the following rules: ► Leading blanks are not allowed, but trailing blanks can be included. ► Leading zeros can be omitted from the month and day elements of the date. An implicit specification of 0 is assumed for any digit that is omitted. ► Leading zeros must be included for the year element of the date.
TIME	TIME *string-constant* The *string-constant* value must conform to one of the formats listed in Table 6-5 on page 155, subject to the following rules: ► Leading blanks are not allowed, but trailing blanks can be included. ► Leading zeros can be omitted from the hour elements of the time. An implicit specification of 0 is assumed for any digit that is omitted. ► The seconds element of the time can be omitted. An implicit specification of 0 seconds is assumed when the seconds element is omitted ► The hour can be '24' if the USA format is not used, and the minutes and seconds are all zeros. ► Using the USA format, the following additional rules apply: – The minutes element of the time can be omitted. An implicit specification of 0 minutes is assumed when the minutes element is omitted. For example, 1 PM is equivalent to 1:00 PM. – The letters A, M, and P can be specified in lowercase. – A single blank must precede the AM or PM. – The hour must not be greater than 12 and cannot be 0 except for the special case of 00:00 AM. ► Using the ISO format of the 24-hour clock, the correspondence between the USA format and the 24-hour clock is as follows: – 12:01 AM through 12:59 AM correspond to 00.01.00 through 00.59.00 – 01:00 AM through 11:59 AM correspond to 01.00.00 through 11.59.00 – 12:00 PM (noon) through 11:59 PM correspond to 12.00.00 through 23.59.00 – 12:00 AM (midnight) corresponds to 24.00.00 – 00:00 AM (midnight) corresponds to 00.00.00

Data type	Format
TIMESTAMP	TIMESTAMP *string-constant* [a] The *string-constant* value is subject to the following rules: ► Leading blanks are not allowed, but trailing blanks can be included. ► Leading zeros can be omitted from the month, day, and hour elements of the timestamp. If any digit is omitted, 0 is assumed. Leading zeros must be included for the minute and second elements of the timestamp ► The hour can be '24' if the minutes, seconds, and any fractional seconds are all zeros. ► The period separator character following the seconds element can be omitted if fractional seconds are not included. ► The number of digits of fractional seconds can vary from 0 to 12. An implicit specification of 0 is assumed if fractional seconds are omitted. The number of digits of fractional seconds determines the precision of the timestamp value. ► The separator between date and time elements can be either a blank or a hyphen-minus or a 'T'. If the separator between date and time elements is a blank or a 'T', the separator between hour and minutes and between minutes and seconds can be either a period or a colon, otherwise only period can be used as the separator between hour and minutes and between minutes and seconds.

a. The format rules described here do not cover TIMESTAMP WITH TIME ZONE. Refer to 6.7, "Support for TIMESTAMP WITH TIME ZONE" on page 166 for related coverage.

Table 6-4 shows the standard formats for the string representation of date.

Table 6-4 Formats for string representation of dates

Format name	Abbreviation	Date format	Example
International Standards Organization	ISO	yyyy-mm-dd	2010-08-06
IBM USA standard	USA	mm/dd/yyyy	08/06/2010
IBM European standard	EUR	dd.mm.yyyy	06.08.2010
Japanese industrial standard Christian era	JIS	yyyy-mm-dd	2010-08-06

Table 6-5 shows the standard formats for the string representation of time.

Table 6-5 Formats for string representation of times

Format name	Abbreviation	Date format	Example
International Standards Organization	ISO	hh.mm.ss [a]	17.30.00
IBM USA standard	USA	hh:mm AM or PM	5:30 PM
IBM European standard	EUR	hh.mm.ss	17.30.00
Japanese industrial standard Christian era	JIS	hh:mm:ss	17:30:00

a. This is an earlier version of the ISO format. JIS can be used to get the current ISO format.

Example 6-23 shows some valid and invalid datetime constants.

Example 6-23 Examples of datetime constants

```
DATE '1977-01-18'                     -- valid date (ISO format)
DATE '18.01.1977'                     -- valid date (EUR format)
DATE '0000-01-18'                     -- invalid date (bad year)
DATE '2010-02-29'                     -- invalid date (bad day)
TIME '24:00:00'                       -- valid time (JIS, current ISO format)
TIME '24:00:01'                       -- invalid time (bad hour)
TIME '00.00.00'                       -- valid time (EUR format)
TIME '12:01 AM'                       -- valid time (USA format)
TIMESTAMP '2007-05-14 11:55:00.1234'  -- valid (space and colons)
```

```
TIMESTAMP '2007-05-14 11.55.00.1234'  -- valid (space and periods)
TIMESTAMP '2007-05-14-11:55:00.1234'  -- invalid (hyphen-minus and colons)
TIMESTAMP '2007-05-14-11.55.00.1234'  -- valid (hyphen-minus and periods)
TIMESTAMP '2007-05-14 11.55.00.'      -- valid (period after ss)
TIMESTAMP '2007-05-14 11.55.00'       -- valid (period after ss omitted)
TIMESTAMP ' 2007-05-14 11.55.00'      -- invalid (leading blanks)
TIMESTAMP '2007-05-14 11.55.00.   '   -- valid (trailing blanks ok)
TIMESTAMP '2007-05-14T11:55:00.1234'  -- valid (T and colons)
TIMESTAMP '2007-05-14T11.55.00.1234'  -- valid (T and periods)
```

To demonstrate that constants declared in this manner are not character strings but are really datetime, you can use the HEX built-in scalar function to see the stored representation. Example 6-24 illustrates this suggestion. We select HEX of the date constant and observe that the stored representation is a date (packed decimal format of the form *yyyymmdd*). Then we select HEX of the string constant representation of a date and observe that the stored representation is a string (EBCDIC string of the form '*yyyy-mm-dd*').

Example 6-24 Difference between datetime constant and character string constant

```
SELECT HEX( DATE '2010-08-06' )       -- display datetime
FROM   SYSIBM.SYSDUMMY1;
-- result: 20100806

SELECT HEX(      '2010-08-06' )       -- display character string
FROM   SYSIBM.SYSDUMMY1;
-- result: F2F0F1F0604F0F864F0F6
```

The new datetime constants are a better way of specifying an explicit cast of a character string constant to a datetime data type. Instead of specifying DATE('2010-08-06') in your query (which is an expression with a constant input), you can specify DATE '2010-08-06' (which is just a constant). This method improves portability and can improve performance if you use a constant instead of an expression with a constant.

6.6 Variable timestamp precision

Prior to DB2 10 for z/OS, the format of the timestamp data type was fixed and provided the precision of one microsecond (10^{-6} of a second). The complete string representation of a timestamp had the form *yyyy-mm-dd-hh.mm.ss.nnnnnn*, where *nnnnnn* represents the fractional specification of microseconds. Some applications (such as Java or .NET) and other database systems support a greater level of precision for the timestamp data type. In addition, with faster machines, the timestamp values returned by DB2 in the parallel sysplex environment might no longer be unique. To address these issues, DB2 10 for z/OS enhances the timestamp data type by providing a greater timestamp precision and making the timestamp precision varying.

Enhancement to timestamp to support TIME ZONE: DB2 10 for z/OS also enhances the timestamp data type by providing a TIME ZONE support. This enhancement is independent from the greater precision enhancement. We discuss this enhancement in 6.7, "Support for TIMESTAMP WITH TIME ZONE" on page 166. In this section, unless otherwise noted, all references to a *timestamp* imply the TIMESTAMP WITHOUT TIME ZONE.

With DB2 10 for z/OS new-function mode, timestamp data type can provide precision up to a picosecond (10^{-12} of a second). You can control the number of digits for the fractional second using the optional timestamp precision attribute. The precision attribute range is 0 to 12, and the default value is 6 because only six digits for the fractional second were supported in prior releases. Example 6-25 illustrates various timestamp precisions that could be declared.

Example 6-25 Examples of timestamp precision

```
CREATE TABLE TRYTMSPREC(
 ID INT
,TMS_DFLT TIMESTAMP               -- precision 6  (default )
,TMS_0    TIMESTAMP(0)            -- precision 0  (minimum supported)
,TMS_6    TIMESTAMP(6)            -- precision 6  (same as default)
,TMS_9    TIMESTAMP(9)            -- precision 9
,TMS_12   TIMESTAMP(12)           -- precision 12 (maximum supported)
);
```

As in prior DB2 releases, the stored representation of a timestamp is a string of packed decimal digits. However, unlike prior releases, where the stored length was always 10 bytes, the stored length of a timestamp in DB2 10 for z/OS varies from 7 bytes (for timestamp with precision 0) to 13 bytes (for timestamp with precision 12). Figure 6-11 shows the formula for determining the stored length L_{int} based on the timestamp precision p.

$$L_{int} = 7 + \left\lceil \frac{p}{2} \right\rceil$$

Figure 6-11 Stored length of the timestamp based on its precision

Note that based on this formula, the data for a timestamp with an odd precision occupies the same amount of storage as the data for a timestamp with the next even precision. For example, timestamp with precision 9 occupies the same amount of storage (12 bytes) as the timestamp with precision 10. Thus, you might want to always use even precision unless there is some requirement to use a particular precision. You can use LENGTH built-in scalar function to see the number of bytes needed to represent a given timestamp value.

The external length (as described in the SQLDA) was 26 bytes in prior DB2 releases and now varies from 19 bytes (for timestamp with precision 0) to 32 bytes (for timestamp with precision 12). Figure 6-12 shows formula for determining the external length L_{ext} based on the timestamp precision *p.*

$$L_{ext} = 19 + p + \left\lceil \frac{p}{p+1} \right\rceil$$

Figure 6-12 Described length of the timestamp based on its precision

6.6.1 String representation of timestamp values

The external representation of a timestamp is a string representation of the form *yyyy-mm-dd-hh.mm.ss* (19 bytes, when precision is 0) or *yyyy-mm-dd-hh.mm.ss.nnnnnnnnnnnn* (where the number of digits *n* for the fractional second can vary from 1 to 12, resulting in external length of 21 to 32 accordingly).

To ensure compatibility with existing applications, if a string representation of a timestamp is implicitly cast to a value with a timestamp data type, the timestamp precision is assumed to

be 6 regardless of the number of digits for fractional seconds in the string! Any fractional second digits beyond the sixth digit are truncated, and if there are less than 6 digits, the missing digits are assumed to be zero. For example, 2010-08-06-12.11.00.7 is treated as 2010-08-06-12.11.00.700000 and 2007-05-14-11.55.00.123456789 is treated as 2007-05-14-11.55.00.123456.

If you need a string representation of a timestamp to be given a different timestamp precision, you can explicitly cast the value to a timestamp with a specified precision or, in the case of a constant, you can use the timestamp constant instead of a character-string constant by using TIMESTAMP keyword (for example TIMESTAMP '2007-05-14 11:55:00.123456789' has the precision of 9).

6.6.2 Timestamp assignment and comparison

A value that is assigned to a timestamp column, a timestamp variable, or a timestamp parameter must be a timestamp, a TIMESTAMP constant, or a valid string representation of a timestamp. A timestamp can be assigned only to a timestamp column, a character string or graphic-string column, a timestamp variable, or a character string or graphic-string variable. If the timestamp precision of the target is less than the timestamp precision of the assigned value, the excessive fractional seconds are truncated. If the timestamp precision of the target is greater than the timestamp precision of the assigned value, the missing digits for fractional seconds are assumed to be zero.

Example 6-26 shows some timestamp assignment cases. The first insert statement assigns timestamp value with precision 12 to columns of table TRYTMSPREC (declared in Example 6-25). Because some columns of that table are declared with smaller precision, truncation occurs. The second insert statement assigns timestamp value with precision 3 to columns of table TRYTMSPREC. Because some columns of that table are declared with greater precision, zeros are appended to get the target precision.

Example 6-26 Timestamp assignment example

```
INSERT INTO TRYTMSPREC VALUES(1
 ,'2010-08-06 12:11:30.123456789999'   -- precision 12 into 6
 ,'2010-08-06 12:11:30.123456789999'   -- precision 12 into 0
 ,'2010-08-06 12:11:30.123456789999'   -- precision 12 into 6
 ,'2010-08-06 12:11:30.123456789999'   -- precision 12 into 9
 ,'2010-08-06 12:11:30.123456789999'   -- precision 12 into 12
);
SELECT TMS_DFLT, TMS_0, TMS_6, TMS_9, TMS_12 FROM TRYTMSPREC WHERE ID = 1;
-- result:  2010-08-06-12.11.30.123456
--          2010-08-06-12.11.30
--          2010-08-06-12.11.30.123456
--          2010-08-06-12.11.30.123456789
--          2010-08-06-12.11.30.123456789999
INSERT INTO TRYTMSPREC VALUES(2
 ,'2010-08-06 12:11:30.123'            -- precision 3 into 6
 ,'2010-08-06 12:11:30.123'            -- precision 3 into 0
 ,'2010-08-06 12:11:30.123'            -- precision 3 into 6
 ,'2010-08-06 12:11:30.123'            -- precision 3 into 9
 ,'2010-08-06 12:11:30.123'            -- precision 3 into 12
);
SELECT TMS_DFLT, TMS_0, TMS_6, TMS_9, TMS_12 FROM TRYTMSPREC WHERE ID = 2;
-- result:  2010-08-06-12.11.30.123000
--          2010-08-06-12.11.30
--          2010-08-06-12.11.30.123000
```

```
--          2010-08-06-12.11.30.123000000
--          2010-08-06-12.11.30.123000000000
```

When comparing timestamp values with different precisions, the operand with smaller precision is extended by appending zeros to its fractional seconds to make it the same precision as the other operand. Example 6-27 demonstrates this concept by comparing timstamp(6) column with timestamp(9) column. Example 6-26 inserts two rows into TRYTMSPREC table.

For the first row, the value in a timestamp(6) column TMS_6 is 2010-08-06-12.11.00.123456. When compared to the timestamp(9) column TMS_9, it is extended to 2010-08-06-12.11.00.123456000, and it does not match the value 2010-08-06-12.11.00.123456789 in TMS_9. However, for the second row, the value in TMS_6 is 2010-08-06-12.11.00.123000, and it is extended to 2010-08-06-12.11.00.123000000, which matches the value in TMS_9.

Example 6-27 Timestamp comparison example

```
SELECT TMS_6, TMS_9
FROM   TRYTMSPREC
WHERE TMS_6 = TMS_9;                    -- precision 6 compared with precision 9
-- result: 2010-08-06-12.11.30.123000
--         2010-08-06-12.11.30.123000000
```

Rules for result timestamp precision

When an operation involves a timestamp operand or operands, the following rules are observed:

- If one operand is a timestamp of precision *x* and the other operand is a timestamp of precision *y*, the result data type of an operation is a timestamp of precision MAX(*x*,*y*).
- If one operand is a timestamp of precision *x* and the other operand is a character string, the result data type of an operation is a timestamp of precision *x*, even though the character string might contain a timestamp representation with a higher precision. In other words, DB2 does not scan the character string operand in an attempt to detect the precision of the timestamp value in it.

Example 6-28 demonstrates the rules for determining the result precision of a timestamp.

Example 6-28 Example of timestamp precision of the result

```
SELECT VALUE(TMS_6, TMS_9)                -- precision 3 and 9
FROM   TRYTMSPREC
WHERE  ID = 1;
-- result: 2010-08-06-12.11.30.123456000  -- precision 9

SELECT VALUE('2010-08-06-12:11:30.1239'   -- string has precision 4
            ,TMS_0                        -- precision 0
           )
FROM   TRYTMSPREC
WHERE  ID = 1;
-- result: 2010-08-06-12.11.30            -- precision 0
```

CURRENT TIMESTAMP special register

When referencing CURRENT TIMESTAMP special register, if a timestamp with a specific precision is desired, the special register can be referenced as CURRENT

TIMESTAMP(integer), where integer can range from 0 to 12. The default precision is 6. SYSDATE can also be specified as a synonym for CURRENT TIMESTAMP(0). Example 6-29 shows some possible references to CURRENT TIMESTAMP.

Example 6-29 CURRENT TIMESTAMP reference examples

```
SELECT CURRENT_TIMESTAMP              -- default (precision 6)
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-18-03.17.07.716218

SELECT CURRENT_TIMESTAMP(12)          -- precision 12
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-18-03.17.07.723155367187

SELECT CURRENT_TIMESTAMP(0)           -- precision 0
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-18-03.17.07

SELECT SYSDATE                        -- precision 0
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-18-03.17.07
```

Datetime durations

A labeled duration represents a specific unit of time as expressed by a number (which can be the result of an expression) followed by one of the seven duration keywords. The number specified is converted as though it were assigned to a DECIMAL(15,0) number, except for SECONDS, which uses DECIMAL(27,12) to allow 0 to 12 fractional second digits to be included.

A timestamp duration represents a number of years, months, days, hours, minutes, seconds, and fractional seconds expressed as a DECIMAL(14+s,s) number, where s is the number of fractional seconds in the range from 0 to 12. To be interpreted properly, the number must have the format *yyyyxxddhhmmss.zzzzzzzzzzzz*, where *yyyy*, *xx*, *dd*, *hh*, *mm*, *ss*, and *zzzzzzzzzzzz* represent, respectively, the number of years, months, days, hours, minutes, seconds, and fractional seconds. The result of subtracting one timestamp value from another is a timestamp duration with a scale that matches the maximum timestamp precision of the timestamp operands.

Timestamp arithmetic

Timestamps can be subtracted, incremented, or decremented.

The result of subtracting one timestamp from another is a timestamp duration that specifies the number of years, months, days, hours, minutes, seconds, and fractional seconds between the two timestamps.The data type of the result is DECIMAL(14+s,s), where s is the maximum timestamp precision of the two operands.

The result of incrementing (adding a duration to) a timestamp or decrementing (subtracting a duration from) a timestamp is itself a timestamp. The precision of the result timestamp matches the precision of the timestamp operand.

Archived

6.6.3 Scalar function changes

The EXTRACT scalar function is enhanced so that the SECOND option returns fractional seconds in addition to the seconds. Example 6-30 shows some sample invocations of EXTRACT scalar function.

Example 6-30 EXTRACT scalar function example for timestamp

```
SELECT EXTRACT(SECOND FROM TMS_DFLT)           -- default precision 6
      ,EXTRACT(SECOND FROM TMS_12  )           -- precision 12
      ,EXTRACT(SECOND FROM TMS_0   )           -- precision 0
FROM  TRYTMSPREC
WHERE ID = 1;
-- result: 30.123456
--         30.123456789999
--         30
```

The SECOND scalar function is enhanced so that an optional second argument can be specified to request fractional seconds to be returned. The optional second argument must be an integer constant in the range of 0 through 12. When this optional argument is specified, the result of the function is DECIMAL(2+*s*, *s*) where *s* is the value of the argument. Example 6-31 shows some sample invocations of SECOND scalar function.

Example 6-31 SECOND scalar function example for timestamp

```
SELECT SECOND(TMS_12  , 12)
      ,SECOND(TMS_12  ,  1)
      ,SECOND(TMS_DFLT    )
      ,SECOND(TMS_DFLT, 12)
      ,SECOND(TMS_0   ,  3)
      ,SECOND(TMS_12  ,  6)
      ,SECOND(TMS_12  ,  0)
FROM  TRYTMSPREC
WHERE ID = 1;
-- result: 30.123456789999
--         30.1
--         30
--         30.123456000000
--         30.000
--         30.123456
--         30
```

The LENGTH result is the number of bytes that are needed to represent the timestamp value as documented in the formula in Figure 6-11 on page 157. Example 6-32 shows sample invocations of LENGTH scalar function.

Example 6-32 LENGTH scalar function example for timestamp

```
SELECT LENGTH(CURRENT_TIMESTAMP    )
      ,LENGTH(CURRENT_TIMESTAMP( 0))
      ,LENGTH(CURRENT_TIMESTAMP( 6))
      ,LENGTH(CURRENT_TIMESTAMP( 9))
      ,LENGTH(CURRENT_TIMESTAMP(10))
      ,LENGTH(CURRENT_TIMESTAMP(12))
FROM  SYSIBM.SYSDUMMY1;
-- result: 10
--          7
--         10
```

Archive

```
--         12
--         12
--         13
```

The MICROSECOND is not enhanced for greater timestamp precision. The result is still an integer between 0 and 999999. If the precision of the timestamp exceeds 6, the value is truncated. Example 6-33 shows sample invocations of MICROSECOND scalar function.

Example 6-33 MICROSECOND scalar function example for timestamp

```
SELECT MICROSECOND(TMS_DFLT)
      ,MICROSECOND(TMS_0   )
      ,MICROSECOND(TMS_6   )
      ,MICROSECOND(TMS_12  )
FROM  TRYTMSPREC
WHERE ID = 1;
-- result: 123456
--              0
--         123456
--         123456
```

The TIMESTAMP second argument can now be an integer constant that specifies the result timestamp precision and must be in the range of 0 to 12.

If the second argument is not an integer constant, the result timestamp precision is 6. If the second argument is not specified, the result timestamp precision depends on the first argument:

- If it is a character or graphic string, the result timestamp precision is 6.
- If it is a date, the result timestamp precision is 0.
- If it is a timestamp with precision p, the result timestamp precision is p.

Example 6-34 shows some sample invocations of TIMESTAMP scalar function.

Example 6-34 TIMESTAMP scalar function example for timestamp

```
SELECT TIMESTAMP('2010-08-06 12:11:30.123456789'    )
      ,TIMESTAMP('2010-08-06 12:11:30.123      '    )
      ,TIMESTAMP('2010-08-06 12:11:30.123456789', 12)
      ,TIMESTAMP(TMS_9                             )
      ,TIMESTAMP(TMS_9                           ,  6)
      ,TIMESTAMP(TMS_9                           , 12)
FROM  TRYTMSPREC
WHERE ID = 1;
-- result: 2010-08-06-12.11.30.123456
--         2010-08-06-12.11.30.123000
--         2010-08-06-12.11.30.123456789000
--         2010-08-06-12.11.30.123456789
--         2010-08-06-12.11.30.123456
--         2010-08-06-12.11.30.123456789000
```

For the NEXT_DAY, ROUND_TIMESTAMP, TIMESTAMP_ISO, TRUNC_TIMESTAMP statements, if expression is a timestamp, the result of the function is a timestamp with the same precision as expression. Otherwise, the result of the function is a TIMESTAMP(6).

Archived

Statement changes

Other changes to clauses due to the improved precision are as follows:

ALTER TABLE FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

If data-type is specified, it must be timestamp with a precision of 6.

ALTER COLUMN ... SET DATA TYPE

When altering a TIMESTAMP column, the new precision must be at least as large as the existing precision of the column. If the precision of a timestamp column is increased, the existing data values are extended with zeros so that the number of fractional second digits matches the specified timestamp precision.

If the timestamp column that is being changed is part of an index, the index is placed in advisory REORG-pending (AREO*) status.

CREATE TABLE FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

If data-type is specified, it must be timestamp with a precision of 6.

- Built-in type

  You can optionally specify the precision, for example TIMESTAMP(12). The default value is TIMESTAMP(6).

- Byte counts of columns by data type

  Byte count is determined by formula in Figure 6-11

Catalog table changes

Because the timestamp can now have varying precision, the precision information needs to be reflected in the DB2 catalog.

DB2 uses the existing column SCALE (which was only used for the decimal data type in prior releases) to record the timestamp precision. Note that DB2 does not update this column when migrating existing objects from prior release to DB2 10, so you can have a SCALE column containing 0 and the LENGTH column containing 10 for some timestamp data, in which case, you should assume the precision is 6.

Be aware that SCALE is also 0 for timestamp precision 0, however, the LENGTH column in this case has a value of 7. The catalog tables that contain SCALE and LENGTH columns are: SYSCOLUMNS, SYSDATATYPES, SYSKEYTARGETS, SYSKEYTARGETS_HIST, SYSPARMS.

6.6.4 Application programming

Table 6-6 shows the timestamp related declarations generated by DCLGEN.

Table 6-6 Declarations generated by DCLGEN

SQL data type	C	COBOL	PL/I
TIMESTAMP	char var[27]	PIC X(26)	CHAR(26)
TIMESTAMP(0)	char var[20]	PIC X(19)	CHAR(19)
TIMESTAMP(p) p > 0	char var[21+p]	PIC X(20+p)	CHAR(20+p)

Table 6-7 shows the Assembly host variable equivalents that you can use when retrieving timestamp data.

Table 6-7 Equivalent SQL and Assembly data types

SQL data type	Assembly host variable equivalent	Notes
TIMESTAMP(0)	DS CLn	n must be at least 19.
TIMESTAMP(p) p > 0	DS CLn	n must be at least 19. To include fractional seconds, n must be 20+x where x is the number of fractional seconds to include; if x is less than p, truncation occurs on the fractional seconds part.

Table 6-8 shows C host variable equivalents that you can use when retrieving timestamp data.

Table 6-8 Equivalent SQL and C data types

SQL data type	C host variable equivalent	Notes
TIMESTAMP(0)	NUL-terminated character form, or VARCHAR structured form	The length must be at least 20 for NUL-terminated character form and at least 19 for the VARCHAR structured form.
TIMESTAMP(p) p > 0	NUL-terminated character form, or VARCHAR structured form	The length must be at least 20 for NUL-terminated character form and at least 19 for the VARCHAR structured form. To include fractional seconds, the length must be 21+x (for NUL-terminated) or 20+x (for structured VARCHAR) where x is the number of fractional seconds to include; if x is less than p, truncation occurs on the fractional seconds part.

Table 6-9 shows COBOL host variable equivalents that you can use when retrieving timestamp data.

Table 6-9 Equivalent SQL and COBOL data types

SQL data type	COBOL host variable equivalent	Notes
TIMESTAMP(0)	Fixed-length character string of length n. For example, 01 VAR-NAME PIC X(n).	n must be at least 19.
TIMESTAMP(p) p > 0	Fixed-length character string of length n. For example, 01 VAR-NAME PIC X(n).	n must be at least 19. To include fractional seconds, n must be 20+x where x is the number of fractional seconds to include; if x is less than p, truncation occurs on the fractional seconds part.

Table 6-10 shows PL/I host variable equivalents that you can use when retrieving timestamp data.

Table 6-10 Equivalent SQL and PL/I data types

SQL data type	PL/I host variable equivalent	Notes
TIMESTAMP(0)	CHAR(n)	n must be at least 19.

SQL data type	PL/I host variable equivalent	Notes
TIMESTAMP(p) p > 0	CHAR(n)	n must be at least 19. To include fractional seconds, n must be 20+x where x is the number of fractional seconds to include; if x is less than p, truncation occurs on the fractional seconds part.

Exchanging timestamp data with other database systems using DRDA

Support for greater timestamp precision in DRDA requires DRDA SQL Application Manager (SQLAM) level 9. DB2 10 for z/OS operates at SQLAM level 9 and exchanges this type of data with any DRDA system that is also at SQLAM 9 or higher.

DB2 does not send input data with greater timestamp precision to a downlevel server (a server at DRDA SQLAM level 8 or below). A SQLCODE -352 is generated. This scenario includes DB2 acting as an application requester, receiving the input data directly from a local application or client driver. It also includes DB2 acting as an intermediate server for a 3-part name statement, where this input data is received from an upstream requester. The application or client driver can resubmit the request with supported data types.

DB2 sends timestamp output data with a precision other than 6 to a downlevel requester after downgrading the data. If the precision is greater than 6, the timestamp data is truncated to timestamp(6). If the precision is less than 6, the timestamp data is padded with zero to provide the missing data.

ODBC

ODBC applications use the C data structure TIMESTAMP_STRUCT to declare application variables for storing timestamp values. This structure has been modified by changing the fraction field from SQLINTEGER to SQLBIGINT. Figure 6-13 shows the new structure.

```
typedef struct TIMESTAMP_STRUCT
  {
    SQLUSMALLINT   year;
    SQLUSMALLINT   month;
    SQLUSMALLINT   day;
    SQLUSMALLINT   hour;
    SQLUSMALLINT   minute;
    SQLUSMALLINT   second;
    SQLBIGINT      fraction;
  } TIMESTAMP_STRUCT;
```

Figure 6-13 TIMESTAMP_STRUCT used for SQL_C_TYPE_TIMESTAMP

The DB2 OBDC functions are impacted as follows:

- SQLBindParameter()

  To bind a parameter marker with the SQL type SQL_TYPE_TIMESTAMP, specify a value between 0 and 12 for the ibScale argument. This value is the maximum number of digits that can be included in the fractional seconds part of a timestamp. You must specify ibScale even if the parameter is a data-at-execute parameter. The cbColDef argument is ignored when the SQL type is SQL_TYPE_TIMESTAMP. If you do not specify a valid timestamp precision, the SQLBindParameter API call fails with SQLSTATE HY104.

- SQLBindCol(), SQLGetData()

  If the C data type of the application variable for retrieving the timestamp data is SQL_C_TYPE_TIMESTAMP, the pcbValue output argument contains 24 (size of the TIMESTAMP_STRUCT) after the fetch. If the C data type is a character data type, that is SQL_C_CHAR or SQL_C_WCHAR, the resulting string is in the "`yyyy-mm-dd hh:mm:ss[.fffffffffff]`" format.

- SQLGetTypeInfo()

  SQLGetTypeInfo() returns information about the data types that are supported by the database management systems that are associated with DB2 ODBC. For the SQL_TYPE_TIMESTAMP SQL data type, SQLGetTypeInfo() returns the following information:

  ```
  COLUMN_SIZE = 32, FIXED_PREC_SCALE = SQL_FALSE, MINIMUM SCALE = 0,
  MAXIMUM_SCALE = 12
  ```

- SQLSetColAttributes()

  If the SQL type is SQL_TYPE_TIMESTAMP, set cbColDef and ibScale arguments to the precision and scale of the TIMESTAMP column respectively. If the value for ibScale is less than 0 or greater than 12, SQLSetColAttributes fail with SQLSTATE HY104.

6.7 Support for TIMESTAMP WITH TIME ZONE

A *time zone* is the time difference in hours and minutes between the local time and Coordinated Universal Time (UTC), which is the international standard of civil time. A time zone can be expressed as an offset from UTC, in the form ±*th:tm*, where ± is the sign, *th* is the hour, and *tm* is the minute component of the time zone offset (for example, -08:00). Alternatively, a time zone can be expressed by a name or abbreviation (for example, *Pacific Standard Time* or *PST*).

Many applications need to deal with times from different time zone. However, the TIMESTAMP data type as implemented prior to DB2 10 does not capture an associated time zone. For example, clients in many locations can use a call center application. If call centers in Los Angeles and New York use the same call center application, a call time recorded simply as 11:00 a.m. is incomplete data. The call might have been recorded at 11:00 a.m. in either Los Angeles time or in New York time.

Storing a time zone with a timestamp value disambiguates what the timestamp value really means. The lack of DB2 support for time zone information for user data has led some customers to develop applications that store time zone information in another column of a table that contains a timestamp value. The application uses the stored time zone information to convert the timestamp value to different time zones, which causes extra burden on application developers in terms of resource and it is also error prone. Furthermore, applications developed using time zone capability on other platforms cannot be ported to the DB2 for z/OS without changing the design of their tables and making changes to the application code.

DB2 10 for z/OS introduces the TIMESTAMP WITH TIME ZONE data type to address this requirement. With DB2 10 for z/OS new-function mode, you can store, load, and manipulate a TIMESTAMP WITH TIME ZONE data.

The TIMESTAMP WITH TIME ZONE is considered a new data type and is different from the timestamp data type that existed in prior releases. To differentiate the two timestamp data types, the existing timestamp data type is referred to officially as TIMESTAMP WITHOUT TIME ZONE. However, in this book, we refer to it as simply the TIMESTAMP (omitting the

WITHOUT TIME ZONE phrase). We refer to the new data type as TIMESTAMP WITH TIME ZONE. Figure 6-14 shows the relevant portion of the diagram of the data types supported by DB2.

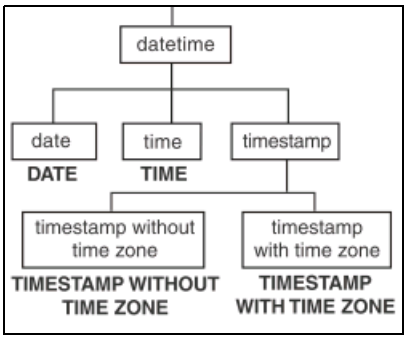


Figure 6-14 TIMESTAMP data types supported by DB2

6.7.1 Examples of TIMESTAMP WITH TIME ZONE

The concept of the timestamp precision applies to both timestamp data types (with time zone and without time zone). In 6.6, “Variable timestamp precision” on page 156, we discuss the timestamp precision for the TIMESTAMP WITHOUT TIME ZONE. Most of that section applies to the TIMESTAMP WITH TIME ZONE as well, but we mention here any differences and aspects related to the TIMESTAMP WITH TIME ZONE.

Example 6-35 illustrates various possible timestamp declarations.

Example 6-35 TIMESTAMP declaration examples

```
CREATE TABLE TRYTMSTZ(
 ID         INTEGER
,TMS_DFLT   TIMESTAMP                    -- precision  6 without time zone
,TMSTZ_DFLT TIMESTAMP WITH TIME ZONE     -- precision  6 with    time zone
,TMS_0      TIMESTAMP(0)                 -- precision  0 without time zone
,TMSTZ_6    TIMESTAMP(6) WITH TIME ZONE  -- precision  6 with    time zone
,TMS_9      TIMESTAMP(9)                 -- precision  9 without time zone
,TMSTZ_12   TIMESTAMP(12) WITH TIME ZONE -- precision 12 with    time zone
,TMS_12     TIMESTAMP(12)                -- precision 12 without time zone
);
INSERT INTO TRYTMSTZ VALUES(
 1
,'2010-08-06 12:30:00.123456'
,'2010-08-06 12:30:00.123456-07:00'
,'2010-08-06 12:30:00'
,'2010-08-06 12:30:00.123456-07:00'
,'2010-08-06 12:30:00.123456789'
,'2010-08-06 12:30:00.123456789012-07:00'
,'2010-08-06 12:30:00.123456789012'
);
```

Alternative spelling: The term TIMEZONE (one word) is accepted by DB2 DDL as well as TIME ZONE (two words). LOAD and UNLOAD utilities use only TIME ZONE.

A TIMESTAMP WITH TIME ZONE is a six-part or seven-part value (year, month, day, hour, minute, second, and optional fractional second) with the time zone specification, that

represents a date and time. The time zone is the difference in hours and minutes between local time and UTC. The range of the hour offset is -24 to +24, and the minute offset is 00 to 59. The time zone is specified in the format ±*th.tm* (where ± is the sign, *th* is the hour, and *tm* is the minute) with valid values ranging from -24.00 to +24.00.

Valid range for time zone: The SQL standard specification for a time zone range is from -12:59 to +14:00. The W3C XML standard for XSD schema definition specifies -14:00 to +14:00. Because the valid range for a CURRENT TIME ZONE special register is -24:00 to +24:00, the valid range for the time zone offset was chosen to also be -24:00 to 24:00 for compatibility.

The external representation of a TIMESTAMP WITH TIME ZONE value is the local timestamp followed by the time zone offset. For example, the city of Sacramento is in the Pacific Time Zone, which is 8 hours behind the Coordinated Universal Time (UTC-8) when daylight saving is not in effect, so Sacramento local time of 12:11:00 on the 06 August 2010 is represented as 2010-08-06 12.11.00-8:00. This TIMESTAMP WITH TIME ZONE value represents a UTC value 2010-08-06 20.11.00, which is derived by subtracting the time zone offset from the local timestamp.

The stored representation of a TIMESTAMP WITH TIME ZONE consists of two parts. The first part is the UTC timestamp, and its format is the same packed decimal format as for TIMESTAMP WITHOUT TIME ZONE. The second part that follows the UTC timestamp is a 2-byte representation of a time zone. Thus, the stored representation of a TIMESTAMP WITH TIME ZONE varies from 9 bytes (for timestamp with precision 0) to 15 bytes (for timestamp with precision 12). Figure 6-15 shows the formula for determining the stored length L_{int} of the TIMESTAMP WITH TIME ZONE based on the timestamp precision *p.*

$$L_{int} = 9 + \left\lceil \frac{p}{2} \right\rceil$$

Figure 6-15 Stored length of the TIMESTAMP WITH TIME ZONE based on its precision

The first byte of the 2-byte time zone consists of two packed decimal digits that represent time zone hour element and the first bit is used to represent the sign of the time zone offset (the bit is set if negative, and is zero if positive). The second byte of a time zone consists of two packed decimal digits that represent time zone minute element. For example, time zone offset -11:00 is represented as X'9100' and time zone offset 11:00 is represented as X'1100'.

Recall that the external length (as described in the SQLDA) for TIMESTAMP WITHOUT TIME ZONE varies from 19 bytes (for timestamp with precision 0) to 32 bytes (for timestamp with precision 12). For the TIMESTAMP WITH TIME ZONE, DB2 adds 128 bytes to represent the time zone. Thus, the external length for TIMESTAMP WITH TIME ZONE varies from 147 to 160 bytes.

When converting the time zone to external representation (a string) DB2 only needs 6 bytes (one byte for the sign, two bytes for the hour element, one byte for the separator and two bytes for the minute element), however the reason for reserving 128 bytes is for future enhancement because time zone can also be represented by a string (for example `'Pacific`

Standard Time'). So while the declared external length of TIMESTAMP WITH TIME ZONE is 147 to 160 bytes, the actual length at this time should be 25 (19 +6) to 38 (32 +6) bytes as demonstrated in Example 6-36.

Example 6-36 Actual and reserved lengths of a TIMESTAMP WITH TIME ZONE

```
SELECT LENGTH(CHAR(CURRENT_TIMESTAMP( 0) WITH TIME ZONE))     -- minimum declared
      ,LENGTH(CHAR(CURRENT_TIMESTAMP(12) WITH TIME ZONE))     -- maximum declared
      ,LENGTH(VARCHAR(CURRENT_TIMESTAMP( 0) WITH TIME ZONE))  -- minimum actual
      ,LENGTH(VARCHAR(CURRENT_TIMESTAMP(12) WITH TIME ZONE))  -- maximum actual
FROM   SYSIBM.SYSDUMMY1;
-- result: 147
--         160
--          25
--          38
```

Figure 6-12 shows the formula for determining the external length L_{ext} based on the timestamp precision *p*.

$$L_{ext} = 147 + p + \left\lceil \frac{p}{p+1} \right\rceil$$

Figure 6-16 Described (external) length of the timestamp based on its precision

6.7.2 String representation of TIMESTAMP WITH TIME ZONE values

The external representation of a TIMESTAMP WITH TIME ZONE is a string representation of the form *yyyy-mm-dd-hh.mm.ss±th:tm* (25 bytes, when precision is 0) or *yyyy-mm-dd-hh.mm.ss.nnnnnnnnnnnn±th:tm* (where the number of digits n for the fractional second can vary from 1 to 12, resulting in actual external length of 27 to 38 accordingly). This is the format that DB2 will return on output (for example when converting TIMESTAMP WITH TIME ZONE to a string); however, there are other valid string representations that DB2 accepts as input. Table 6-3 on page 154 describes all valid string representations of a timestamp. You can include an optional time zone to the formats described in that table, and this time zone can have one of the following formats:

- *±th:tm*, with values ranging from -24:00 to +24:00. A value of -0:00 is treated the same as +0:00.
- *±th*, with values ranging from -24 to +24. An implicit specification of 00 is assumed for the time zone minute element in this case.
- Uppercase *Z* to specify UTC (this is to support timestamp values originating from XML).
- The time zone can be optionally separated from the timestamp with one blank

Example 6-37 shows some valid and invalid string representations of a TIMESTAMP WITH TIME ZONE.

Example 6-37 Examples of string representation of a TIMESTAMP WITH TIME ZONE

```
'2007-05-14 11:55:00.1234+08:00'      -- valid, +th:tm format
'2007-05-14 11.55.00.1234-08:00'      -- valid, -th:tm format
'2007-05-14 11.55.00.1234 08:00'      -- invalid, missing sign
'2007-05-14 11.55.00.1234 +08:00'     -- valid, optional blank separator
'2007-05-14 11.55.00.1234  +08:00'    -- invalid, more than one blank
'2007-05-14 11.55.00.+08:00'          -- valid, no fractional seconds
'2007-05-14 11.55.00-08:00'           -- valid, no fractional seconds
```

```
'2007-05-14 11.55.00Z'               -- valid, UTC specification
'2007-05-14 11.55.00 Z'              -- valid, UTC specification
'2007-05-14 11.55.00.Z'              -- valid, UTC specification
'2007-05-14 11.55.00+Z'              -- invalid, only Z alone allowed
'2007-05-14 11:55:00.1234+8:00'      -- valid +th:tm
'2007-05-14 11:55:00.1234+8:0'       -- invalid, incomplete minute
'2007-05-14 11:55:00.1234+8:'        -- invalid, incomplete minute
'2007-05-14 11:55:00.1234+8'         -- valid, +th format
'2007-05-14 11:55:00-00'             -- valid, +th format
```

6.7.3 Determination of the implicit time zone

There might be situations where DB2 needs to implicitly determine a time zone. For example, this might be needed for operations that combine TIMESTAMP WITHOUT TIME ZONE value with a TIMESTAMP WITH TIME ZONE value, such as inserting a TIMESTAMP WITHOUT TIME ZONE value into a TIMESTAMP WITH TIME ZONE column. DB2 10 for z/OS introduces new DSNHDECP parameter IMPLICIT_TIMEZONE and a new special register SESSION TIME ZONE to aid in determining the implicit time zone. DB2 uses the IMPLICIT_TIMEZONE parameter to determine what time zone is to be associated with a value that does not have a time zone. The implicit time zone is determined as follows:

- If IMPLICIT_TIMEZONE is not specified or is specified as CURRENT, the implicit time zone is the value of the CURRENT TIME ZONE special register.
- If IMPLICIT_TIMEZONE is specified as SESSION, the implicit time zone is the value of the SESSION TIME ZONE special register.
- If IMPLICIT_TIMEZONE is specified as a character string in the format of '*±th:tm*', the implicit time zone is the time zone value represented by that character string.

TIMESTAMP WITHOUT TIME ZONE can be promoted to TIMESTAMP WITH TIME ZONE (implicit time zone will be used by DB2 in this case), and conversely TIMESTAMP WITH TIME ZONE can be promoted to TIMESTAMP WITHOUT TIME ZONE (time zone is simply disregarded in this case).

String representation of a timestamp: A string representation of a TIMESTAMP WITH TIME ZONE cannot be promoted or implicitly cast to TIMESTAMP WITHOUT TIME ZONE. If a string representation of a timestamp is implicitly cast to a TIMESTAMP WITHOUT TIME ZONE value, the string must not contain a time zone (SQLSTATE 22007, SQLCODE –20497).

Example 6-38 illustrates these points. In this example, we create a SQL scalar function TRYTMSPROMO with input parameter data type of timestamp(6) without time zone. We then reference this function using TIMESTAMP WITH TIME ZONE arguments and see that DB2 promotes the argument to TIMESTAMP WITHOUT TIME ZONE. Note that our inputs have different time zones (that is, they represent different time) yet the output is the same because the time zone was disregarded during the promotion cast. The third invocation of the function uses string representation of a TIMESTAMP WITH TIME ZONE, and it fails because such promotion (or cast) is not allowed.

Example 6-38 Promotion of timestamp data type

```
CREATE FUNCTION TRYTMSPROMO(P1 TIMESTAMP)
RETURNS VARCHAR(30)
LANGUAGE SQL
RETURN VARCHAR(P1);
```

```
SELECT TRYTMSPROMO(TIMESTAMP '2010-08-06 12:11:00.123456789-8:00')
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-06-12.11.00.123456
SELECT TRYTMSPROMO(TIMESTAMP '2010-08-06 12:11:00.123456789-9:00')
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-06-12.11.00.123456
SELECT TRYTMSPROMO(          '2010-08-06 12:11:00.123456789-9:00')
FROM   SYSIBM.SYSDUMMY1;
-- result: SQLCODE = -20497
```

Timestamp with time zone can be cast to the following data types:

- CHAR
- VARCHAR
- DATE
- TIME
- TIMESTAMP WITHOUT TIME ZONE
- TIMESTAMP WITH TIME ZONE

In addition, the following data types can be cast to TIMESTAMP WITH TIME ZONE:

- CHAR
- VARCHAR
- GRAPHIC (Unicode only)
- VARGRAPHIC (Unicode only)
- TIMESTAMP WITHOUT TIME ZONE

Example 6-39 demonstrates some casts from TIMESTAMP WITH TIME ZONE to other data types. We select a TIMESTAMP(6) WITH TIME ZONE column from the table TRYTMSTZ that was defined in Example 6-35 on page 167. The table has one row with the TMSTZ_6 column having the TIMESTAMP WITH TIME ZONE value 2010-08-06 12:30:00.123456-7:00. Note how casting to DATE, TIME, and TIMESTAMP returns the corresponding portion of the timestamp in local time and not UTC (that is, the time zone information is ignored).

Example 6-39 Example of TIMESTAMP WITH TIME ZONE casts

```
SELECT CAST(TMSTZ_6 AS CHAR(38)   )
      ,CAST(TMSTZ_6 AS VARCHAR(38))
      ,CAST(TMSTZ_6 AS DATE      )
      ,CAST(TMSTZ_6 AS TIME      )
      ,CAST(TMSTZ_6 AS TIMESTAMP  )
FROM  TRYTMSTZ;
-- result: 2010-08-06-12.30.00.123456-07:00
--         2010-08-06-12.30.00.123456-07:00
--         2010-08-06
--         12.30.00
--         2010-08-06-12.30.00.123456
```

6.7.4 TIMESTAMP WITH TIME ZONE assignment and comparison

The following assignment rule applies to both TIMESTAMP WITH TIME ZONE and TIMESTAMP WITHOUT TIME ZONE. A value that is assigned to a timestamp column, a timestamp variable, or a timestamp parameter must be a timestamp, a TIMESTAMP constant, or a valid string representation of a timestamp. A timestamp can be assigned only to a

timestamp column, a character-string or graphic-string column, a timestamp variable, or a character-string or graphic-string variable.

When a TIMESTAMP WITH TIME ZONE value is compared to a TIMESTAMP WITHOUT TIME ZONE VALUE, the TIMESTAMP WITHOUT TIME ZONE value is cast to TIMESTAMP WITH TIME ZONE before the comparison is made.

When a TIMESTAMP WITHOUT TIME ZONE value is compared to a string representation of a timestamp, the string is cast to TIMESTAMP WITHOUT TIME ZONE (error occurs during the cast if the string representation includes a time zone specification) before the comparison is made.

When a TIMESTAMP WITH TIME ZONE value is compared to a string representation of a timestamp, the string is cast to TIMESTAMP WITH TIME ZONE (regardless of whether the string representation included a time zone specification) before the comparison is made.

Two TIMESTAMP WITH TIME ZONE values are compared using the UTC representations of the values. The two TIMESTAMP WITH TIME ZONE values are considered equal if they represent the same instance in UTC, regardless of the time zone offsets that are stored in the values. Thus, 2010-08-06-12.11.00-08:00 (12:11 p.m. Pacific Standard Time) is the same as 2010-08-06-15.11.00-05:00 (3:11 p.m. Eastern Standard Time).

The same rule applies when comparing a TIMESTAMP WITH TIME ZONE value with a string representation of a TIMESTAMP WITH TIME ZONE value.

Example 6-40 lists some additional examples of comparison.

Example 6-40 Timestamp with time zone comparison examples

```
1) TIMESTAMP '1977-08-01 12:34:56-7:00' = TIMESTAMP '1977-08-01 12:34:56-7:00'
2) TIMESTAMP '1977-08-01 12:34:56-7:00' = TIMESTAMP '1977-08-01 12:34:56-8:00'
3) TIMESTAMP '1977-08-01 12:34:56-7:00' = TIMESTAMP '1977-08-01 11:34:56-8:00'
4) TIMESTAMP '1977-08-01 12:34:56-7:00' = TIMESTAMP '1977-08-01 12:34:56     '
5) TIMESTAMP '1977-08-01 12:34:56-8:00' = TIMESTAMP '1977-08-01 12:34:56     '
6) TIMESTAMP '1977-08-01 12:34:56     ' = '1977-08-01 12:34:56-8:00'
7) TIMESTAMP '1977-08-01 12:34:56     ' = '1977-08-01 12:34:56     '
8) TIMESTAMP '1977-08-01 12:34:56-7:00' = '1977-08-01 12:34:56     '
9) TIMESTAMP '1977-08-01 12:34:56-7:00' = '1977-08-01 12:34:56-7:00'
10)TIMESTAMP '1977-08-01 12:34:56-7:00' = '1977-08-01 12:34:56-8:00'
11)TIMESTAMP '1977-08-01 12:34:56-7:00' = '1977-08-01 11:34:56-8:00'

The results for the presented predicates are as follows:
1) TRUE because the UTC representations for both sides are the same.
2) FALSE because UTC representations of left hand side (time 19.34.56) is not the
   same as the UTC representation of the right hand side (time 20.34.56).
3) TRUE because the UTC representations for both sides are the same.
4) The right hand side is a TIMESTAMP WITHOUT TIME ZONE, so the implicit time zone
   will be used for comparison. Assuming that the implict time zone is -07:00
   the UTC representations for both sides will be the same, so TRUE.
5) The right hand side is a TIMESTAMP WITHOUT TIME ZONE, so the implicit time zone
   will be used for comparison. Assuming that the implict time zone is -07:00
   the UTC representation for the left hand side (time 20.34.56) is different from
   the UTC representation for the right hand side (time 19.34.56), so FALSE.
6) The left hand side is a TIMESTAMP WITHOUT TIME ZONE and the right hand side is
   a string representation of a TIMESTAMP WITH TIME ZONE. DB2 will attempt to cast
   the string to a TIMESTAMP WITHOUT TIME ZONE and will issue an error since this
   cast is not allowed.
```

```
7) The left hand side is a TIMESTAMP WITHOUT TIME ZONE and the right hand side is
   a string representation of a TIMESTAMP WITHOUT TIME ZONE. DB2 will cast the
   string to a TIMESTAMP WITHOUT TIME ZONE, and both sides will have the same
   timestamp value, so TRUE.
8) The left hand side is a TIMESTAMP WITH TIME ZONE and the right hand side is
   a string representation of a TIMESTAMP WITHOUT TIME ZONE. DB2 will cast the
   string to a TIMESTAMP WITH TIME ZONE using the implicit time zone. Assuming
   that the implict time zone is -07:00, both sides will have the same UTC
   representation, so TRUE.
9) The left hand side is a TIMESTAMP WITH TIME ZONE and the right hand side is
   a string representation of a TIMESTAMP WITH TIME ZONE. DB2 will cast the
   string to a TIMESTAMP WITH TIME ZONE. Both sides have the same UTC
   representation, so TRUE.
10)The left hand side is a TIMESTAMP WITH TIME ZONE and the right hand side is
   a string representation of a TIMESTAMP WITH TIME ZONE. DB2 will cast the
   string to a TIMESTAMP WITH TIME ZONE. The UTC representation for the left hand
   side (time 19.34.56) is different from the UTC representation for the right
   hand side (time 20.34.56), so FALSE.
11)The left hand side is a TIMESTAMP WITH TIME ZONE and the right hand side is
   a string representation of a TIMESTAMP WITH TIME ZONE. DB2 will cast the
   string to a TIMESTAMP WITH TIME ZONE. Both sides have the same UTC
   representation, so TRUE.
```

6.7.5 Rules for result data type with TIMESTAMP WITH TIME ZONE operands

When operation involves TIMESTAMP WITH A TIME ZONE operand or operands, the following rules are followed:

- If one operand is a TIMESTAMP WITH TIME ZONE of precision *x*, and the other operand is a TIMESTAMP WITH TIME ZONE of precision y, the result data type of an operation is a TIMESTAMP WITH TIME ZONE of precision MAX(*x*,*y*).
- If one operand is a TIMESTAMP WITH TIME ZONE of precision *x*, and the other operand is a character string, the result data type of an operation is a TIMESTAMP WITH TIME ZONE of precision *x*.
- if one operand is a TIMESTAMP WITH TIME ZONE of precision *x*, and the other operand is a TIMESTAMP WITHOUT TIME ZONE of precision *y*, the result data type of an operation is a TIMESTAMP WITH TIME ZONE of precision MAX(*x*,*y*).

Example 6-41 demonstrates the rules for determining the result data type for the operation that involves TIMESTAMP WITH TIME ZONE operands.

Example 6-41 Result data type when operation involves TIMESTAMP WITH TIME ZONE

```
SELECT VALUE(TMSTZ_6                    -- timestamp( 6) with time zone
            ,TMSTZ_12                   -- timestamp(12) with time zone
       )
FROM   TRYTMSTZ;
-- result: 2010-08-06-12.30.00.123456000000-07:00 -- precision 12 with time zone

SELECT VALUE('1977-01-18 12:34:56'      -- string (without time zone)
            ,TMSTZ_6                    -- timestamp(6) with time zone
       )
FROM   TRYTMSTZ;
-- result: 1977-01-18-12.34.56.000000-07:00       -- precision 6 with time zone
```

```
SELECT VALUE('1977-01-18 12:34:56Z'     -- string (with time zone)
            ,TMSTZ_6                    -- timestamp(6) with time zone
       )
FROM   TRYTMSTZ;
-- result: 1977-01-18-12.34.56.000000+00:00       -- precision 6 with time zone
SELECT VALUE(TMS_0                      -- timestamp(0) without timezone
            ,TMSTZ_6                    -- timestamp(6) with time zone
       )
FROM   TRYTMSTZ;
-- result: 2010-08-21-23.20.57.000000-07:00       -- precision 6 with time zone
```

6.7.6 CURRENT TIMESTAMP WITH TIME ZONE special register

When referencing CURRENT TIMESTAMP special register, if TIMESTAMP WITH A TIME ZONE is desired, the special register can be referenced as CURRENT TIMESTAMP (integer) WITH TIME ZONE, or CURRENT TIMESTAMP WITH TIME ZONE. SYSTIMESTAMP can be specified as an alternative to CURRENT TIMESTAMP(12) WITH TIME ZONE. The time zone for CURRENT TIMESTAMP WITH TIME ZONE special register is always determined from the CURRENT TIME ZONE special register, and CURRENT TIME ZONE is set based on the TIMEZONE parameter (in SYS1.PARMLIB(CLOCKXX)) of the processor for the server executing the SQL statement.

Example 6-42 shows some possible references to CURRENT TIMESTAMP special register (results are shown for the CURRENT TIME ZONE setting that represents the time zone offset of -07:00).

Example 6-42 CURRENT TIMESTAMP WITH TIME ZONE examples

```
SELECT CURRENT_TIMESTAMP WITH TIME ZONE       -- default precision 6
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-21-23.38.51.138546-07:00

SELECT CURRENT_TIMESTAMP(12) WITH TIME ZONE   -- precision 12
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-21-23.38.51.149405218750-07:00

SELECT CURRENT_TIMESTAMP(0) WITH TIME ZONE    -- precision 0
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-21-23.38.51-07:00

SELECT SYSTIMESTAMP                           -- precision 12
FROM   SYSIBM.SYSDUMMY1;
-- result: 2010-08-21-23.38.51.150258750001-07:00
```

Implicit time zone value note: If CURRENT TIMESTAMP (that is CURRENT TIMESTAMP WITHOUT TIME ZONE) special register is referenced in a TIMESTAMP WITH TIME ZONE context (for example, when compared with a TIMESTAMP WITH TIME ZONE column), the implicit time zone for the CURRENT TIMESTAMP special register is based on the implicit time zone system parameter (IMPLICIT_TIMEZONE). Thus, the implicit time zone value can be different from the value of CURRENT TIME ZONE special register. Keep this difference in mind when developing applications.

Archive

Example 6-43 demonstrates the possible difference between the implicit time zone and the CURRENT TIME ZONE value that is used for CURRENT TIMESTAMP WITH TIME ZONE. In this example, we assume that the value of CURRENT TIME ZONE special register is -70000. (which is a time duration value representing the time zone offset of “-07:00”). In addition, we further assume that the implicit time zone system parameter IMPLICIT_TIMEZONE is set to SESSION.

During the first insert, the SESSION TIME ZONE has the same value as the CURRENT TIME ZONE so identical timestamp values are inserted into columns C1 and C2. However, when SESSION TIME ZONE value changes, the values inserted into columns C1 and C2 are different for the same insert statement. Similarly, different values would be inserted if the implicit time zone system parameter IMPLICIT_TIMEZONE were set to some character string that is different from CURRENT TIME ZONE.

Example 6-43 CURRENT TIMESTAMP: Current and implicit time zone

```
CREATE TABLE TRYTMSIMPL(
  C1 TIMESTAMP WITH TIME ZONE
 ,C2 TIMESTAMP WITH TIME ZONE
);
INSERT INTO TRYTMSIMPL VALUES(
  CURRENT TIMESTAMP WITH TIME ZONE          -- uses CURRENT TIME ZONE
 ,CURRENT TIMESTAMP                         -- uses implicit time zone
);
SET SESSION TIME ZONE = '-08:00';           -- change implicit time zone
INSERT INTO TRYTMSIMPL VALUES(
  CURRENT TIMESTAMP WITH TIME ZONE          -- uses CURRENT TIME ZONE
 ,CURRENT TIMESTAMP                         -- uses implicit time zone
);
SELECT C1, C2 FROM TRYTMSIMPL;
-- result: C1                                C2
--         2010-08-22-00.24.33.353694-07:00  2010-08-22-00.24.33.353694-07:00
--         2010-08-22-00.24.33.430211-07:00  2010-08-22-00.24.33.430211-08:00
```

SESSION TIME ZONE special register

The SESSION TIME ZONE special register specifies a value that identifies the time zone of the application process.

The data type is VARCHAR(128).

The time zone value is in the format of *±th:tm*, where the *th* value represents the time zone hour offset, and the *tm* value represents the time zone minute offset. Valid values for the *th* value are between -12 and +14. Valid values for the *tm* value are between 0 and 59.

When referencing the SESSION TIME ZONE special register, you can use an alternative syntax of SESSION TIMEZONE. When setting the SESSION TIME ZONE special register, you can use an alternative syntax of: SESSIONTIMEZONE, SESSION TIMEZONE, TIMEZONE, or TIME ZONE. See Example 6-44.

Example 6-44 SESSION TIME ZONE example

```
SET SESSION TIME ZONE = '-08:00';
SET SESSION TIMEZONE  = '-08:00';
SET SESSIONTIMEZONE   = '-08:00';
SET TIMEZONE          = '-08:00';
SET TIME ZONE         = '-08:00';
```

```
SELECT SESSION TIME ZONE FROM SYSIBM.SYSDUMMY1;
SELECT SESSION TIMEZONE FROM SYSIBM.SYSDUMMY1;
```

The initial value of the special register in a user-defined function or stored procedure is inherited according to the rules for using special registers in a user-defined function or a stored procedure. In other contexts, the initial value of the special register represents the same time zone as the CURRENT TIME ZONE special register.

The value of the special register can be changed by executing the SET SESSION TIME ZONE statement. After a SET SESSION TIME ZONE statement is processed, the values of the SESSION TIME ZONE and CURRENT TIME ZONE special register might not reflect the same value.

Time zone specific expressions

You can use time zone specific expressions to adjust timestamp values and character-string or graphic-string representations of timestamp values to specific time zones. Figure 6-17 shows the syntax diagram for the time zone specific expressions.

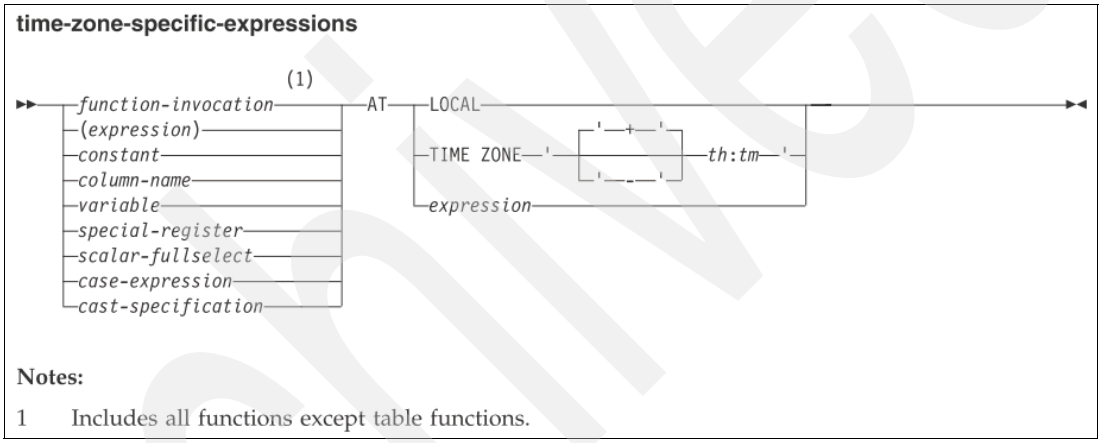


Figure 6-17 Syntax for time zone specific expressions

The first operand for time-zone-specific-expression must be an expression that returns the value of either a built-in timestamp or a built-in character or graphic string data type. If the first operand is a character string or graphic string, it must not be a CLOB or DBCLOB value, and its value must be a valid character-string or graphic-string representation of a timestamp.

If the first operand of time-zone-specific-expression returns a TIMESTAMP WITHOUT TIME ZONE value, the expression is implicitly cast to TIMESTAMP WITH TIME ZONE before it is adjusted to the indicated time zone.

The following options are available:

- AT LOCAL

  Specifies that the timestamp value is adjusted for the local time zone using the SESSION TIME ZONE special register.

- AT TIME ZONE '±*th:tm*'

  Specifies that the timestamp is adjusted for the time zone provided. The *th value* represents the time zone hour offset, and the *tm* value represents the time zone minute offset. The time zone value must be in the range of -12:59 to +14:00.

Archived

- AT *expression*

  Specifies that the timestamp is adjusted for the time zone that is represented by the *expression* value. The *expression* value is a character or graphic string and must not be a CLOB or DBCLOB value. The value must be left justified and be of the form '*±th:tm*', where the *th* value represents the time zone hour between -12 and +14, and the *tm* value represents the time zone minutes between 0 and 59, with values ranging from -12:59 to +14:00. The value must not be the null value.

  The *expression* value must be enclosed in parenthesis.

The expression returns a TIMESTAMP WITH TIME ZONE value in the indicated time zone.

Example 6-45 shows some sample time zone specific expressions.

Example 6-45 Time zone specific expression examples

```
SET TIMEZONE = '+2:00';
SELECT '2007-05-14-11:55:00.0 -8:00' AT LOCAL
FROM   SYSIBM.SYSDUMMY1;
-- result: 2007-05-14-21.55.00.000000+02:00
SET TIMEZONE = '+5:00';
SELECT '2007-05-14-11:55:00.0 -8:00' AT LOCAL
FROM   SYSIBM.SYSDUMMY1;
-- result: 2007-05-15-00.55.00.000000+05:00
SELECT '2007-05-14-11:55:00.0 -8:00' AT TIME ZONE '+00:00'
FROM   SYSIBM.SYSDUMMY1;
-- result: 2007-05-14-19.55.00.000000+00:00
SELECT '2007-05-14-11:55:00.0 -8:00'  AT TIME ZONE ('-'||'7'||':'||'00')
FROM   SYSIBM.SYSDUMMY1;
-- result: 2007-05-14-12.55.00.000000-07:00
```

TIMESTAMP WITH TIME ZONE arithmetic

If any of the operands are TIMESTAMP WITH TIME ZONE, any TIMESTAMP WITHOUT TIME ZONE values are implicitly cast to TIMESTAMP WITH TIME ZONE, and the datetime arithmetic operation is performed in UTC time (ignoring the time zone). Example 6-46 shows timestamps with time zone arithmetic.

Example 6-46 Timestamp with time zone arithmetic

```
SELECT TIMESTAMP '2007-05-14-12.55.00-7:00'    -- UTC time 19:55:00
     -TIMESTAMP '2007-05-14-11:55:00-8:00'    -- UTC time 19:55:00
FROM   SYSIBM.SYSDUMMY1;
-- result: 0
SELECT TIMESTAMP '2007-05-14-12.55.00-7:00'    -- UTC time 19:55:00
     -TIMESTAMP '2007-05-14-12:55:00-8:00'    -- UTC time 20:55:00
FROM   SYSIBM.SYSDUMMY1;
-- result: -10000
SELECT TIMESTAMP '2007-05-14-12.55.00-7:00' + 010000.
FROM   SYSIBM.SYSDUMMY1;
-- result: 2007-05-14-13.55.00-07:00
```

6.7.7 Scalar function changes

DB2 10 for z/OS introduces a new scalar function, TIMESTAMP_TZ, and also adds the ability to extract time zone values using scalar function EXTRACT.

The TIMESTAMP_TZ function

The TIMESTAMP_TZ function returns a TIMESTAMP WITH TIME ZONE value from the input arguments. Figure 6-18 shows the syntax diagram for this function.

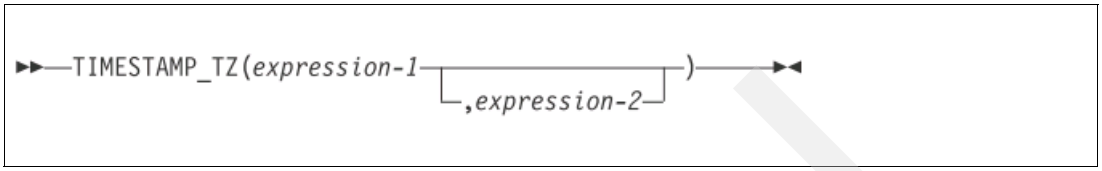


Figure 6-18 TIMESTAMP_TZ syntax diagram

The schema is SYSIBM. The following options are available:

- *expression-1*

  An expression that returns a value of one of the following built-in data types:

  - A TIMESTAMP WITHOUT TIME ZONE
  - A TIMESTAMP WITH TIME ZONE
  - A character string
  - A graphic string

  If *expression-1* is a character string or a graphic string, it must not be a CLOB or DBCLOB, its value must be a valid string representation of a TIMESTAMP WITHOUT TIME ZONE or a TIMESTAMP WITH TIME ZONE value, and it must have an actual length that is not greater than 255 bytes.

  If *expression-2* is specified, *expression-1* must be a TIMESTAMP WITHOUT TIME ZONE, or a string representation of a TIMESTAMP WITHOUT TIME ZONE.

- *expression-2*

  An expression that returns a character string or a graphic string. It must not be a CLOB or DBCLOB, and its value must be a valid string representation of a time zone in the format of '*±th:tm*' with values ranging from -12:59 to +14:00, where *th* represents time zone hour and *tm* represents time zone minute.

The result of the function is a TIMESTAMP WITH TIME ZONE with the same precision as *expression-1*. If *expression-1* is a string and does not have an explicit precision, the result precision is 6. If either argument can be null, the result can be null; if either argument is null, the result is the null value.

If both arguments are specified, the result is a TIMESTAMP WITH TIME ZONE value where *expression-1* specifies the timestamp and *expression-2* specifies the time zone. FROM_TZ can be specified as a synonym for TIMESTAMP_TZ when TIMESTAMP_TZ specifies both *expression-1* and *expression-2*.

If only one argument is specified and it is a TIMESTAMP WITHOUT TIME ZONE, the result is that timestamp cast to TIMESTAMP WITH TIME ZONE.

If only one argument is specified and it is a TIMESTAMP WITH TIME ZONE, the result is that TIMESTAMP WITH TIME ZONE.

If only one argument is specified and it is a string, the result is the TIMESTAMP WITH TIME ZONE represented by that string with precision 6.

Example 6-47 shows some invocations of TIMESTAMP_TZ. For this example, assume that the implicit time zone is '+11:00'.

Archived

Example 6-47 TIMESTAMP_TZ invocation examples

```
SELECT TIMESTAMP_TZ(TIMESTAMP '2007-05-14-12.55.00')
      ,TIMESTAMP_TZ(TIMESTAMP '2007-05-14-12.55.00.12345')
      ,TIMESTAMP_TZ(TIMESTAMP '2007-05-14-12.55.00+2:00')
      ,TIMESTAMP_TZ(TIMESTAMP '2007-05-14-12.55.00.12345+2:00')
      ,TIMESTAMP_TZ('2007-05-14-12.55.00.12345')
      ,TIMESTAMP_TZ('2007-05-14-12.55.00')
      ,TIMESTAMP_TZ('2007-05-14-12.55.00.12345+2:00')
      ,TIMESTAMP_TZ('2007-05-14-12.55.00+2:00')
FROM   SYSIBM.SYSDUMMY1;
-- result: 2007-05-14-12.55.00+11:00
--         2007-05-14-12.55.00.12345+11:00
--         2007-05-14-12.55.00+02:00
--         2007-05-14-12.55.00.12345+02:00
--         2007-05-14-12.55.00.123450+11:00
--         2007-05-14-12.55.00.000000+11:00
--         2007-05-14-12.55.00.123450+02:00
--         2007-05-14-12.55.00.000000+02:00

SELECT TIMESTAMP_TZ(TIMESTAMP '2007-05-14-12.55.00'      ,'-7:00')
      ,TIMESTAMP_TZ(TIMESTAMP '2007-05-14-12.55.00.12345','-7:00')
      ,TIMESTAMP_TZ('2007-05-14-12.55.00.12345'          ,'-7:00')
      ,TIMESTAMP_TZ('2007-05-14-12.55.00'                ,'-7:00')
FROM   SYSIBM.SYSDUMMY1;
-- result: 2007-05-14-12.55.00-07:00
--         2007-05-14-12.55.00.12345-07:00
--         2007-05-14-12.55.00.123450-07:00
--         2007-05-14-12.55.00.000000-07:00

-- the following invocations will result in error:
TIMESTAMP_TZ(TIMESTAMP '2007-05-14-12.55.00+2:00'      ,'-7:00')
TIMESTAMP_TZ(TIMESTAMP '2007-05-14-12.55.00.12345+2:00','-7:00')
TIMESTAMP_TZ('2007-05-14-12.55.00.12345+2:00'          ,'-7:00')
TIMESTAMP_TZ('2007-05-14-12.55.00+2:00'                ,'-7:00')
```

The EXTRACT function

EXTRACT scalar function adds the following extract time zone values:

- TIMEZONE_HOUR

  Specifies that the hour component of the time zone of the timestamp value is returned. TIMEZONE_HOUR can be specified only if the second argument is a timestamp-expression and the *timestamp-expression* contains a time zone.

- TIMEZONE_MINUTE

  Specifies that the minute component of the time zone of the timestamp value is returned. TIMEZONE_MINUTE can be specified only if the second argument is a *timestamp-expression* and the *timestamp-expression* contains a time zone.

Archived

Also, if the *timestamp-expression* argument includes a time zone, the result is determined from the UTC representation of the datetime value. Example 6-48 shows some invocations of EXTRACT.

Example 6-48 Invocations of EXTRACT

```
SELECT EXTRACT(TIMEZONE_HOUR FROM TIMESTAMP '2007-05-14-11.55.00-2:30')
      ,EXTRACT(TIMEZONE_MINUTE FROM TIMESTAMP '2007-05-14-11.55.00-2:30')
      ,EXTRACT(TIMEZONE_HOUR FROM TIMESTAMP '2007-05-14-11.55.00+2:30')
      ,EXTRACT(TIMEZONE_MINUTE FROM TIMESTAMP '2007-05-14-11.55.00+2:30')
      ,EXTRACT(HOUR  FROM TIMESTAMP '2007-05-14-11.55.00+2:00')
      ,EXTRACT(HOUR  FROM TIMESTAMP '2007-05-14-11.55.00-8:00')
FROM   SYSIBM.SYSDUMMY1#
-- result:  -2
--         -30
--           2
--          30
--           9
--          19
```

6.7.8 Statements changes

Here, we describe the SQL statement changes that are related to the new TIMESTAMP WITH TIME ZONE data type.

- ALTER TABLE

  A TIMESTAMP WITHOUT TIME ZONE column can only be altered to TIMESTAMP WITHOUT TIME ZONE with a larger precision, and a TIMESTAMP WITH TIME ZONE column can only be altered to TIMESTAMP WITH TIME ZONE with a larger precision.

- CREATE TABLE

  If *data-type* is specified in the FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP clause, it must be a timestamp(6).

  When declaring a timestamp data type, you can optionally specify WITHOUT TIME ZONE or WITH TIME ZONE. The default is WITHOUT TIME ZONE.

  The byte count for a TIMESTAMP WITH TIME ZONE column is determined by formula in Figure 6-15.

  PARTITION BY RANGE does not support TIMESTAMP WITH TIME ZONE data type.

  FIELDPROC cannot be specified for a column with a TIMESTAMP WITH TIME ZONE data type.

- CREATE INDEX

  The key-expression cannot contain expression that would need an implicit time zone value (for example a cast from a TIMESTAMP WITHOUT TIME ZONE to a TIMESTAMP WITH TIME ZONE).

- SET SESSION TIME ZONE

  The SET SESSION TIME ZONE statement assigns a value to the SESSION TIME ZONE special register.

  This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Archived

Figure 6-19 shows the syntax diagram.

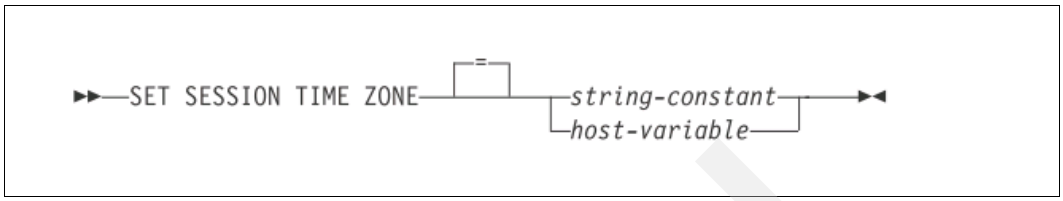


Figure 6-19 Syntax diagram for SET SESSION TIME ZONE

The following options are available:

– *string-constant*

  Identifies a time zone with a value of the form '*±th:tm*', where *th* represents the time zone hour between -12 and +14, and *tm* represents the time zone minutes between 0 and 59, with values ranging from -12:59 to +14:00.

– *host-variable*

  Specifies a variable that contains a time zone. The variable must be a CHAR or VARCHAR variable that is not followed by an indicator variable. The variable must not be the null value. The value must be left justified and be of the form '*±th:tm*', where *th* represents the time zone hour between -12 and +14, and *tm* represents the time zone minutes between 0 and 59, with values ranging from -12:59 to +14:00.

  Setting the SESSION TIME ZONE special register does not affect the CURRENT TIME ZONE special register. See “CURRENT TIMESTAMP WITH TIME ZONE special register” on page 174 for more details.

  There are several SET SESSION TIME ZONE syntax alternatives. They are discussed with some examples in “SESSION TIME ZONE special register” section.

Catalog table changes

The column COLTYPE in the catalog tables SYSCOLUMNS and SYSCOLUMNS_HISTS will have a value TIMESTZ for the TIMESTAMP WITH TIME ZONE data type.

Utilities

Utilities add support for TIMESTAMP WITH TIME ZONE for LOAD and UNLOAD by providing the optional keyword IMPLICIT_TZ ‘*timezone-string*’.

For LOAD, it specifies the implicit time zone to use when time zone is missing from the timestamp string being loaded and the data type of corresponding column in the target table is TIMESTAMP WITH TIME ZONE. For UNLOAD, it specifies the implicit time zone to use when unloading TIMESTAMP WITHOUT TIME ZONE column to a TIMESTAMP WITH TIME ZONE column. TIME ZONE is a two words parameter in LOAD and UNLOAD.

When IMPLICIT_TZ keyword is not specified, the default implicit time zone is determined from the implicit time zone system parameter. The ‘*timezone-string*’ specifies the implicit time zone value. The time zone is the difference (in hours and minutes) between local time and UTC. The range of the hour component is -12 to 14, and the minute component is 00 to 59. The time zone is specified in the form *±th:tm*, with values ranging from -12:59 to +14:00.

For more information, refer to *DB2 10 for z/OS Utility Guide and Reference*, SC19-2984.

Note that for RUNSTATS, HIGH2KEY and LOW2KEY only contain UTC portion of TIMESTAMP WITH TIME ZONE value (time zone will be dropped).

6.7.9 Application programming

Timestamp with time zone is considered a different data type from the TIMESTAMP WITHOUT TIME ZONE. Therefore, a new data type identifier is introduced for the TIMESTAMP WITH TIME ZONE. The data type identifier is 2448 for not nullable and 2449 for nullable TIMESTAMP WITH TIME ZONE.

Table 6-11 shows the TIMESTAMP WITH TIME ZONE related declarations generated by DCLGEN.

Table 6-11 Declarations generated by DCLGEN

SQL data type	C	COBOL	PL/I
TIMESTAMP(0) WITH TIME ZONE	struct { short int c1_len; char c1_data[147]; } c1;	10 C1. 49 C1-LEN PIC S9(4) USAGE COMP. 49 C1-TEXT PIC X(147).	CHAR(147) VAR
TIMESTAMP(p) WITH TIME ZONE p > 0	struct { short int c1_len; char c1_data[148+p]; } c1;	10 C1. 49 C1-LEN PIC S9(4) USAGE COMP. 49 C1-TEXT PIC X(148+p).	CHAR(148+p) VAR

Table 6-12 shows Assembly host variable equivalents that you can use when retrieving TIMESTAMP WITH TIME ZONE data.

Table 6-12 Equivalent SQL and Assembly data types

SQL data type	Assembly host variable equivalent	Notes
TIMESTAMP(0) WITH TIME ZONE	DS HL2,CLn	n must be at least 25.
TIMESTAMP(p) WITH TIME ZONE p > 0	DS HL2,CLn	n must be at least 26+p.

Table 6-13 shows C host variable equivalents that you can use when retrieving TIMESTAMP WITH TIME ZONE data.

Table 6-13 Equivalent SQL and C data types

SQL data type	C host variable equivalent	Notes
TIMESTAMP(0) WITH TIME ZONE	NUL-terminated character form, or VARCHAR structured form	The length must be at least 26 for NUL-terminated character form and at least 25 for the VARCHAR structured form.
TIMESTAMP(p) WITH TIME ZONE p > 0	NUL-terminated character form, or VARCHAR structured form	The length must be at least 27+p for NUL-terminated character form and at least 26+p for the VARCHAR structured form.

Table 6-14 shows COBOL host variable equivalents that you can use when retrieving TIMESTAMP WITH TIME ZONE data.

Table 6-14 Equivalent SQL and COBOL data types

SQL data type	COBOL host variable equivalent	Notes
TIMESTAMP(0) WITH TIME ZONE	Varying-length character string. For example, 01 VAR-NAME. 49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC X(n).	The inner variables must have a level of 49. n must be at least 25.
TIMESTAMP(p) WITH TIME ZONE p > 0	Varying-length character string. For example, 01 VAR-NAME. 49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC X(n).	The inner variables must have a level of 49. n must be at least 26+p.

Table 6-15 shows PL/I host variable equivalents that you can use when retrieving TIMESTAMP WITH TIME ZONE data.

Table 6-15 Equivalent SQL and PL/I data types

SQL data type	PL/I host variable equivalent	Notes
TIMESTAMP(0) WITH TIME ZONE	CHAR(n) VAR	n must be at least 25.
TIMESTAMP(p) WITH TIME ZONE p > 0	CHAR(n) VAR	n must be at least 26+p.

Table 6-16 shows Java data type equivalents that you can use when retrieving TIMESTAMP WITH TIME ZONE data.

Table 6-16 Equivalent SQL and Java data types

SQL data type	Java data type	Notes
TIMESTAMP(p) WITH TIME ZONE	java.sql.TimestampTZ	If the timestamp precision of the target is less than the timestamp precision of the assigned value, the excess fractional seconds are truncated.

Exchanging timestamp data with other database systems using DRDA

Support for TIMESTAMP WITH TIME ZONE in DRDA requires DRDA SQLAM level 9. DB2 10 for z/OS operates at SQLAM level 9 and exchanges this type of data with any DRDA system that is also at SQLAM 9 or higher.

DB2 does not send input data with TIMESTAMP WITH TIME ZONE to a downlevel server (a server at DRDA SQLAM level 8 or below). A SQLCODE -352 is generated. This scenario includes DB2 acting as an application requester, receiving the input data directly from a local application or client driver. It also includes DB2 acting as an intermediate server for a

three-part name statement, where this input data is received from an upstream requester. The application or client driver can resubmit the request with supported data types.

DB2 sends TIMESTAMP WITH TIME ZONE output data with a precision other than 6 to a downlevel requester after downgrading the data to TIMESTAMP. The time zone is truncated so that the data becomes TIMESTAMP. If the precision is greater than 6, the TIMESTAMP data is truncated to TIMESTAMP(6). If the precision is less than 6, the TIMESTAMP data is padded with zero to provide the missing data.

This behavior supports middleware products that perform SELECT * against up-level servers. It is expected that these dynamic applications will issue DESCRIBE to understand the type and size of the output data and will set up appropriate application data types to receive the output data. By choosing the correct type and size, the application avoids padding problems.

6.8 Support for OLAP aggregation specification

In this section, we describe the support for new OLAP specifications. First, we provide a high-level overview of the new functionality and give the syntax diagrams for the new syntax. Then, we discuss some examples to demonstrate some important concepts of the OLAP specifications.

DB2 10 for z/OS extends the support for advanced SQL geared towards data warehousing applications. In this release, OLAP specification is extended with support for a new class of OLAP specification called *moving aggregates*. These specifications support important OLAP capabilities, such as cumulative sums and moving averages by using a window. A window specifies a partitioning, an ordering of rows within partitions and an aggregation group. The aggregation group specifies which rows of a partition, relative to the current row, should participate in the calculation. The aggregation OLAP specifications can only be specified in the select list or the ORDER BY clause of a select statement.

These OLAP specifications, as well as the RANK, DENSE_RANK, and ROW_NUMBER that were introduced in DB2 9 for z/OS, are somewhat of a mixture between the traditional scalar functions and the aggregate functions supported by DB2. Scalar functions are functions that compute a single value for current row based on zero or more input arguments (for example LENGTH, BIGINT, and so forth). Aggregate functions are applied to a group of rows that are collapsed into a single row; they compute a single value for this group (for example COUNT, SUM, and other computed values). By contrast, the OLAP specification functions compute a single value for current row based on some or all the rows in a defined group. For this reason, these specifications are sometimes referred to as *scalar aggregate functions* because they share similarities with both the scalar and aggregate functions.

We present the following diagrams to illustrate this concept.

Figure 6-20 shows the processing of a scalar function. Each current row is input to the function and one value is produced per row.

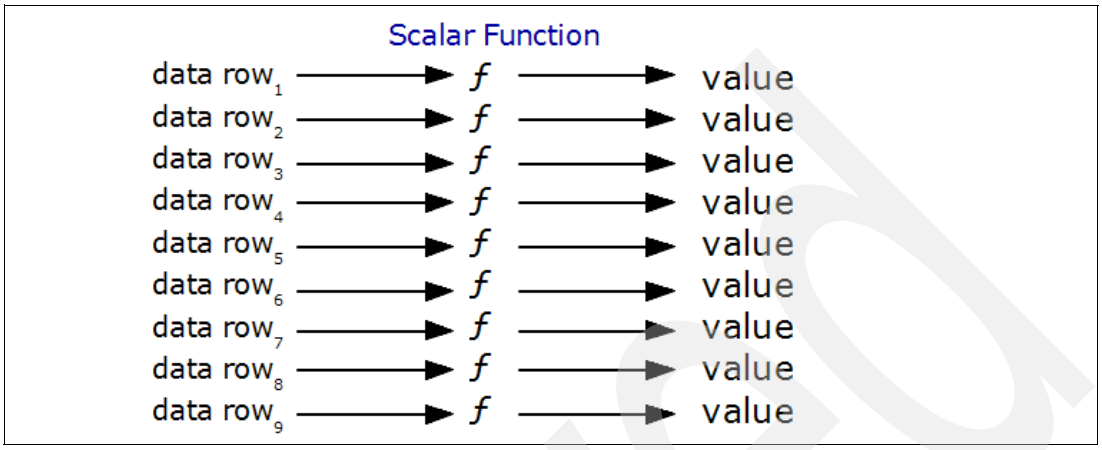


Figure 6-20 Scalar function processing

Figure 6-21 shows the processing of an aggregate function. Rows are collapsed into groups and the input to the function is the set of the rows in a group. One value per group is produced.

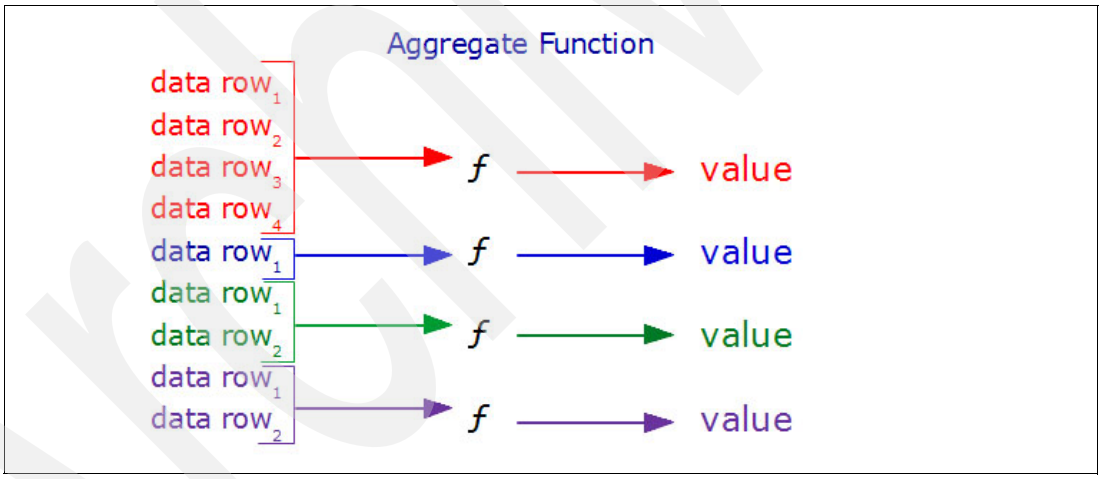


Figure 6-21 Aggregate function processing

Figure 6-22 shows the processing of an OLAP specification. Rows are combined into sets called partitions, and within each partition, the input to the function are the set of rows from a window. One value per row is produced. In other words, while the function output is generated for each row (similar to scalar function), the value of the output depends on a set of rows in the window (similar to aggregate function).

Archived

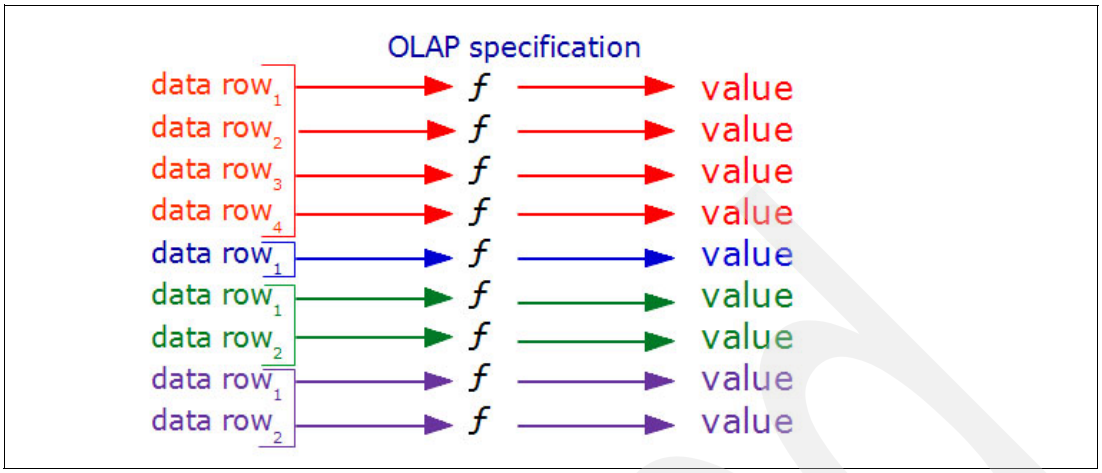


Figure 6-22 OLAP specification processing

Before we dive into the explanation of the OLAP specifications, we define a table to be used in several of our examples in this section. Example 6-49 shows the declaration and the data for such a table. The table contains a dummy row ID, developer name, date of the monthly report, and the number of APARs opened for that month. Note that '?' represents a null value.

Example 6-49 TRYOLAP table definition and its data

```
CREATE TABLE TRYOLAP(
 ID            INTEGER
,NAME          VARCHAR(30)
,DATE          DATE
,APARS_OPENED  INTEGER
);
SELECT * FROM TRYOLAP;
-- result:
+-----------------------------------------------------+
|    ID    |    NAME    | REPORT_DATE | APARS_OPENED |
+-----------------------------------------------------+
|        1 | Andrei     | 2010-01-01  |          1   |
|        2 | Nicole     | 2010-01-01  |          5   |
|        3 | Eugene     | 2010-01-01  |         10   |
|        4 | Andrei     | 2010-02-01  |          5   |
|        5 | Nicole     | 2010-02-01  |         10   |
|        6 | Eugene     | 2010-02-01  |          5   |
|        7 | Andrei     | 2010-03-01  |          6   |
|        8 | Eugene     | 2010-03-01  |          3   |
|        9 | Andrei     | 2010-04-01  |          8   |
|       10 | Nicole     |           ? |          ?   |
|       11 | Eugene     | 2010-04-01  |          5   |
|       12 | Andrei     | 2010-05-01  |          6   |
|       13 | Eugene     | 2010-05-01  |         10   |
|       14 | Andrei     | 2010-06-01  |         10   |
|       15 | Eugene     | 2010-06-01  |         15   |
|       16 | Nicole     | 2010-06-01  |         15   |
+-----------------------------------------------------+
```

One of the main characteristics of the OLAP specification processing is the notion of a window. A window determines which rows are participating in the evaluation of the function,

the order in which the rows should be processed, and the aggregation group, which is a set of rows in a partition for which the aggregation function is to be performed.

The window specification include the following components:

- Partitioning

  Partitioning is specified using PARTITION BY clause. Partitioning divides the final result set of the subselect (that is the final result table after applying FROM/WHERE/GROUP BY/HAVING) into partitions based on partitioning expression. If partitioning is not specified, the entire result set constitutes a single partition.

  Partitioning is similar to grouping using GROUP BY; however, it does not collapse each group into a single row like GROUP BY does. Instead, it associates each row of the result table to a particular partition.

  We demonstrate this in Example 6-50 where we first run a query that executes the SUM() aggregate function on APARS_OPENED column with grouping by NAME using GROUP BY, and observe that for each of the three groups the rows were collapsed into a single row. Then, we run a query that executes the SUM() OLAP-specification on APARS_OPENED column with partitioning by NAME. We observe that the rows were not collapsed but were partitioned by the NAME. In our examples, we mark the end of each partition to help you visualize this concept.

Example 6-50 GROUP BY versus PARTITION BY

```
SELECT NAME, SUM(APARS_OPENED) SUM
FROM   TRYOLAP
GROUP BY NAME;
-- result: NAME       SUM
--         Andrei     36
--         Eugene     48
--         Nicole     30

SELECT NAME, SUM(APARS_OPENED) OVER(PARTITION BY NAME) SUM
FROM   TRYOLAP;
-- result: NAME       SUM
--         Andrei     36
--         Andrei     36
--         Andrei     36
--         Andrei     36
--         Andrei     36
--         Andrei     36
--------------------------- end of partition
--         Eugene     48
--         Eugene     48
--         Eugene     48
--         Eugene     48
--         Eugene     48
--         Eugene     48
--------------------------- end of partition
--         Nicole     30
--         Nicole     30
--         Nicole     30
--         Nicole     30
```

Figure 6-23 shows the syntax for the PARTITION BY clause.

Archive

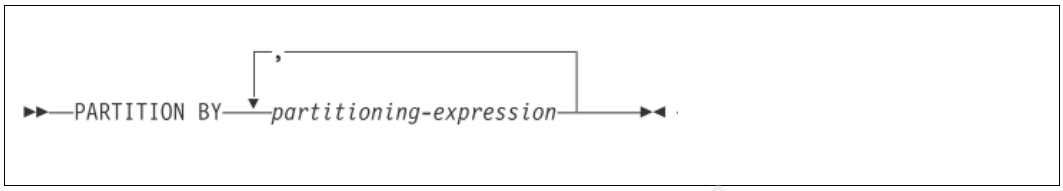


Figure 6-23 Window-partition-clause syntax

- Ordering

  Ordering is specified using ORDER BY clause. Ordering specifies the sort order that is to be used within each partition. Unlike ordering using traditional ORDER BY where all rows of the result set are sorted together, the window ordering sorts each partition separately. Window ordering helps to determine the value of the OLAP specification. Note that you can explicitly control how nulls are sorted by using NULLS FIRST or NULLS LAST in the ORDER BY clause.

  Example 6-51 uses ordering on REPORT_DATE for each NAME partition. We observe that each partition is sorted separately. We also use ASC NULLS FIRST clause to demonstrate that option.

Example 6-51 ORDER BY ordering in partitions

```
SELECT NAME, REPORT_DATE, APARS_OPENED #,
       SUM(APARS_OPENED) OVER(PARTITION BY NAME
                              ORDER BY REPORT_DATE ASC NULLS FIRST) SUM
FROM   TRYOLAP
-- result: NAME     REPORT_DATE   #     SUM
--         Andrei   2010-01-01     1      1
--         Andrei   2010-02-01     5      6
--         Andrei   2010-03-01     6     12
--         Andrei   2010-04-01     8     20
--         Andrei   2010-05-01     6     26
--         Andrei   2010-06-01    10     36
---------------------------------------- end of partition
--         Eugene   2010-01-01    10     10
--         Eugene   2010-02-01     5     15
--         Eugene   2010-03-01     3     18
--         Eugene   2010-04-01     5     23
--         Eugene   2010-05-01    10     33
--         Eugene   2010-06-01    15     48
---------------------------------------- end of partition
--         Nicole             ?     ?      ?
--         Nicole   2010-01-01     5      5
--         Nicole   2010-02-01    10     15
--         Nicole   2010-06-01    15     30
```

Archive

Figure 6-24 shows syntax for the ORDER BY clause.

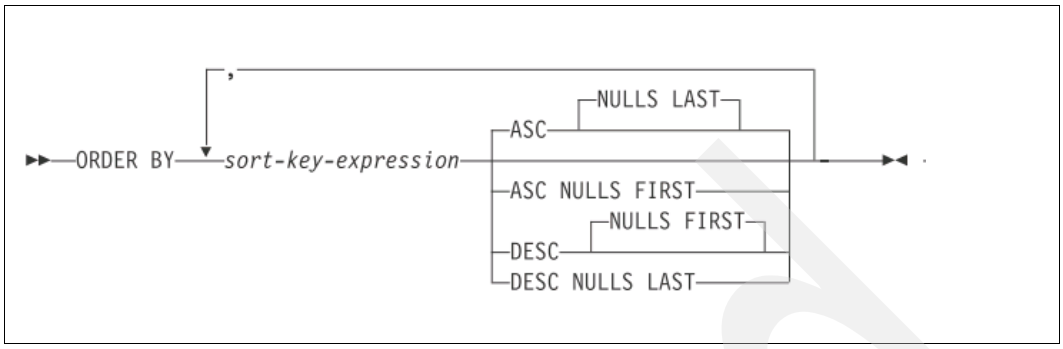


Figure 6-24 Window-ordering-clause syntax

- Aggregation group

  Aggregation group is specified using ROWS/RANGE clause. Aggregation group identifies a set of rows on which to perform an aggregate function. There are two mandatory components in the specification of an aggregation group:

  – ROWS/RANGE clause which indicates the aggregation group type. Keyword ROWS indicates the group is formed by counting rows. In this case the group is referred to as a physical aggregation group. RANGE indicates the group is formed by a numeric offset from the sort key. In this case the group is referred to as a logical aggregation group.

  – An indication of the starting row and ending row of the aggregation group.

Physical aggregation group

An aggregation group specified with ROWS keyword is called *physical* because the rows participating in the aggregation group are determined by counting actual (physical) rows in a partition. For example, if you specified that the aggregation is to take place for the current row and the two preceding rows. Then exactly current row and the two rows immediately preceding the current row will be in the aggregation group, no matter what the sort key values for the rows actually are and whether or not there are any gaps in the ordered sequence of sort keys. Example 6-52 specifies the physical aggregation group that uses current and two preceding rows for the aggregation.

Example 6-52 Aggregation using ROWS

```
SELECT ID, NAME, REPORT_DATE, APARS_OPENED #,
       SUM(APARS_OPENED) OVER(PARTITION BY NAME
                              ORDER BY     REPORT_DATE
                              ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) SUM
FROM   TRYOLAP
WHERE  NAME IN('Andrei', 'Nicole');
-- result: ID  NAME     REPORT_DATE   #     SUM
--          1  Andrei   2010-01-01     1      1
--          4  Andrei   2010-02-01     5      6
--          7  Andrei   2010-03-01     6     12
--          9  Andrei   2010-04-01     8     19
--         12  Andrei   2010-05-01     6     20
--         14  Andrei   2010-06-01    10     24
---------------------------------------------- end of partition
--          2  Nicole   2010-01-01     5      5
--          5  Nicole   2010-02-01    10     15
```

```
--         16  Nicole   2010-06-01    15     30
--         10  Nicole              ?     ?     25
```

The calculation occurs as we describe here.

The result set is partitioned by NAME, so there are two partitions—one for ‘Andrei’ and one for ‘Nicole’. Rows in each partition are sorted by REPORT_DATE (nulls are sorted last by default). The result is the two partitions with rows ordered in the same sequence as shown in Example 6-52.

Aggregation is done for current and two physically preceding rows:

- For the first row of the first partition (ID=1), we get SUM = 1 because there are no preceding rows yet.
- For the second row (ID=4), we aggregate current and one preceding row, so SUM=5+1=6.
- Similarly, the third row is SUM=6+5+1=12. The fourth row is SUM=8+6+5=19. And so on.

For the first row of the second partition (ID=2) we follow the same algorithm and obtain sums: 5, 15, 30, and 25. For the last row of the second partition (ID=10) the APARS_OPENED value is null, which is ignored by the SUM() aggregate function, and hence, the result is SUM=10+15+NULL=25. Note that there are gaps in the ordered sort sequence. One gap is between the sort keys ‘2010-02-01’ and ‘2010-06-01’ (if we are to assume that monthly reports should happen every month), and the other gap is between the sort key ‘2010-06-01’ and the null value. However, it is irrelevant for physical aggregation group because it just counts the rows.

It is important to note, however, that using physical aggregation grouping with ordering on sparse input data (that is when there are gaps in the ordered sequence of the sort keys) might produce undesired results. To demonstrate, suppose we want to calculate the moving average of the number of APARs opened by Nicole for the current and two preceding months. If we use physical aggregation grouping we get the result shown in Example 6-53.

Example 6-53 Sparse input data effect on physical aggregation grouping

```
SELECT ID, NAME, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AVG
FROM   TRYOLAP
WHERE  NAME = 'Nicole';
-- result: ID  NAME     REPORT_DATE   #     AVG
--          2  Nicole   2010-01-01     5      5
--          5  Nicole   2010-02-01    10      7
--         16  Nicole   2010-06-01    15     10
--         10  Nicole   ?              ?     12
```

If we examine the result for row with ID=16, it looks reasonable, since the average of the input values for the current and the two preceding values is AVG = (15+10+5)/3 = 10. However, we intended to average over the current and *two* preceding months. And the average for the current month of “2010-06-01” (June) and two preceding months (which are not present in the input) equals to AVG=(15)/1 = 15. In other words, DB2 went beyond the intended 2-month boundary and included rows for four and five months prior to the current row’s month value. Similarly, the average calculated for the REPORT_DATE value of null (row with ID=10) has no meaning from the perspective of counting preceding months.

Another important concept we should mention is the situation when there are duplicate sort keys (that is, when there are duplicate rows in the partition). SQL semantics does not define

any particular order for returning duplicate rows. In other words, if we sort a set of 2-tuples: {(1, ‘A’), (1, ‘B’), (1, ‘C’)} by the first attribute, the order of the tuples could be: (1, ‘A’), (1, ‘B’), (1, ‘C’), or (1, ‘C’), (1, ‘B’), (1, ‘A’), or any other permutation of the tuples (since the value of the first attribute is the same). In fact, the order of the duplicate rows in the sorted result is implementation-dependent. Therefore, the result of the aggregation when using physical aggregation grouping could be non-deterministic, if duplicate rows are partially included and partially excluded from the window.

Let us illustrate this in the example. Suppose we run the query shown in Example 6-54.

Example 6-54 Duplicate rows effect on physical aggregation grouping: Query

```
SELECT ID, NAME, REPORT_DATE,
       MIN(REPORT_DATE) OVER(ORDER BY NAME
                             ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) MIN
FROM   TRYOLAP
WHERE  APARS_OPENED = 10;
```

We order by the NAME and have one partition that includes all rows of the result set. The result set contains only the rows with ten opened APARs. We want to see the earliest report date between the current and the preceding (alphabetically) person. Example 6-55 shows one possible result of the query.

Example 6-55 Duplicate rows effect on physical aggregation grouping: Result 1

```
-- result: ID  NAME     REPORT_DATE   MIN
--         14  Andrei   2010-06-01    2010-06-01
--         13  Eugene   2010-05-01    2010-05-01
--          3  Eugene   2010-01-01    2010-01-01
--          5  Nicole   2010-02-01    2010-01-01
```

We observe that the duplicate rows (rows with duplicate sort key ‘Eugene’) happened to end up in the sequence where the row with ID=13 comes before the row with ID=3. Next we present another possible result of the same query in Example 6-56.

Example 6-56 Duplicate rows effect on physical aggregation grouping: Result 2

```
-- result: ID  NAME     REPORT_DATE   MIN
--         14  Andrei   2010-06-01    2010-06-01
--          3  Eugene   2010-01-01    2010-01-01
--         13  Eugene   2010-05-01    2010-01-01
--          5  Nicole   2010-02-01    2010-02-01
```

This time, the duplicate rows happened to be in a different sequence (row with ID=3 first, then row with ID=13). Note how this affects the result of the MIN() function.

To summarize, the physical aggregation grouping has the following characteristics:

- You can specify more than one sort key in the ORDER BY clause (since counting rows can cross the sort group boundary)
- Sort key can be of any data type that is allowed to be sorted
- Result could be non-deterministic (since it is possible for duplicate rows to be only partially included in the group)
- Input data for the sort key should be dense (no gaps), otherwise the result might or might not be meaningful (depending on the intent of the query).

Logical aggregation group

An aggregation group specified with RANGE keyword is called logical because the rows participating in the aggregation group are determined by using the sort key value and calculating the offset from that sort key value. In other words, unlike physical aggregation group where you specify the number of rows preceding or following current row, for the logical aggregation group you specify what value should be subtracted from or added to the sort key value of the current row.

Example 6-57 uses the same query as in Example 6-52 on page 189, except instead of the physical aggregation group that uses current and two preceding rows for the aggregation, we specify the logical aggregation group that uses current row and all preceding rows that are two months prior to the month of the current row (we do this by using a DECIMAL(8,0) constant 00000200., which is a date duration value representing two months).

Example 6-57 Aggregation using RANGE

```
SELECT ID, NAME, REPORT_DATE, APARS_OPENED #,
       SUM(APARS_OPENED) OVER(PARTITION BY NAME
                              ORDER BY     REPORT_DATE
                              RANGE 00000200. PRECEDING) SUM
FROM   TRYOLAP
WHERE  NAME IN('Andrei', 'Nicole');
-- result: ID  NAME     REPORT_DATE   #     SUM
--          1  Andrei   2010-01-01     1       1
--          4  Andrei   2010-02-01     5       6
--          7  Andrei   2010-03-01     6      12
--          9  Andrei   2010-04-01     8      19
--         12  Andrei   2010-05-01     6      20
--         14  Andrei   2010-06-01    10      24
------------------------------------------------ end of partition
--          2  Nicole   2010-01-01     5       5
--          5  Nicole   2010-02-01    10      15
--         16  Nicole   2010-06-01    15      15
--         10  Nicole            ?     ?       ?
```

The calculation is essentially the same as in the initial example with the ROWS keyword. There are, as before, two partitions, and the rows in each partition are sorted by REPORT_DATE. Aggregation is done for the current row and any preceding rows which have the sort key value within two months back from the current row's sort key value.

Thus, for the first partition, the result matches the result of the corresponding query that used physical grouping (Example 6-52), which is expected, because counting physical rows and counting logical months happens to be the same for that set of rows. However, results for the second partition differ between the two queries. The result for the row with ID=16 is 15 now, and it was 30 when ROWS clause was used, because of the gap in the sort key sequence that we discussed earlier. For logical grouping, a sort key value is used for determining the group boundary. Because we specified that we want the group to be up to two months preceding the month of the current row and because there are no rows two months apart from the "2010-06-30" date, only that current row is in the group, hence the result value of 15.

Note that even this answer might not be what you expect. For example, if you want to treat missing rows as equivalent of zero APARs being opened for the missing month. In this case the expected average would be AVG = (15+0+0)/3 = 5. Alternatively, perhaps you want to treat any missing rows as invalid case and return null for any incomplete group. You can control this behavior in your query (for example, by using CASE expression to test for the

number of rows in the group). Just be aware of this behavior for incomplete aggregation groups and for groups with duplicate rows.

Similarly, there are no other rows that are two months apart from the null value, which is why the result for the last row (ID=10) is null (the value of APARS_OPENED for that row). *In fact, null value can never be reached by adding or subtracting a finite offset, so any rows with the sort key value of null will be in its own separate group.* The only exception being is when options UNBOUNDED PRECEDING or UNBOUNDED FOLLOWING are used to designate the group boundary, which are treated as infinite offsets and so they can cross the gap between null and non-null.

Thus, logical grouping is much better suited for dealing with sparse data. For example, we can re-run our query from Example 6-53 on page 190 where we observed undesirable outcome. This time we use the logical grouping. Example 6-58 shows the modified query and its result.

Example 6-58 Sparse input data effect on logical aggregation grouping

```
SELECT ID, NAME, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE 00000200. PRECEDING) AVG
FROM   TRYOLAP
WHERE  NAME = 'Nicole';
-- result: ID  NAME     REPORT_DATE   #     AVG
--          2  Nicole   2010-01-01    5      5
--          5  Nicole   2010-02-01   10      7
--         16  Nicole   2010-06-01   15     15
--         10  Nicole   ?             ?      ?
```

Another distinction from the physical grouping is the way the duplicate rows are handled. In logical grouping, *all* duplicate rows (rows with the same sort key value) participate in the group. Therefore, the result is deterministic (provided the function argument is deterministic) even when the ordering of the rows is non-deterministic.

For example, if we rerun our query from Example 6-55 on page 191 using the logical grouping, we get the same result for both of the alternative ordering of the duplicate rows (the two rows with IDs 3 and 13), as demonstrated in Example 6-59, which shows the modified query and the two possible ordering for the two duplicate rows. You can observe the same result for both ordering.

Note that we had to modify the ORDER BY clause slightly, since, for the logical grouping, the sort key cannot be a character string. As a quick and easy workaround, we sort by the integer representation of the first character of the name, which works for our simple example because all the three names in the table start with unique character. Therefore, we get exactly the same sorting result as though we sorted by the full name.

Example 6-59 Duplicate rows effect on logical aggregation grouping

```
SELECT ID, NAME, REPORT_DATE,
       MIN(REPORT_DATE) OVER(ORDER BY ASCII(NAME)
                             RANGE 1 PRECEDING   ) MIN
FROM   TRYOLAP
WHERE  APARS_OPENED = 10;
-- result from possible ordering #1:
-- result: ID  NAME     REPORT_DATE   MIN
--         14  Andrei   2010-06-01    2010-06-01
--         13  Eugene   2010-05-01    2010-01-01
--          3  Eugene   2010-01-01    2010-01-01
```

```
--          5  Nicole   2010-02-01    2010-02-01
-- result from possible ordering #2:
-- result: ID  NAME     REPORT_DATE   MIN
--         14  Andrei   2010-06-01    2010-06-01
--          3  Eugene   2010-01-01    2010-01-01
--         13  Eugene   2010-05-01    2010-01-01
--          5  Nicole   2010-02-01    2010-02-01
```

To summarize, the logical aggregation grouping has the following characteristics:

- You can specify only one sort key in the ORDER BY clause (one exception is the use of UNBOUNDED PRECEDING or UNBOUNDED FOLLOWING, for which more than one sort key is allowed, since there is no subtraction or addition for the key values)
- Sort key must be of the data type that supports subtraction and addition (again, exception is the use of UNBOUNDED keywords)
- Result is deterministic[3] (provided that input is deterministic)
- Input data for the sort key does not have to be dense

Aggregation group boundaries

An aggregation group cannot cross the partition, however it can contain rows preceding the current row, it can “look ahead” and include rows following the current row. It is also possible to have an unbounded aggregation group, either looking back to the beginning of the partition or looking ahead to the end of the partition, or both. Aggregation group might not even include the current row at all.

Aggregation group boundaries are controlled by specifying group-start, group-between, or group-end clauses.

[3] Deterministic functions always return the same result any time they are called with a specific set of input values and given the same state of the database. Non-deterministic functions can return different results each time they are called with a specific set of input values even if the database state that they access is the same.

Archive

Figure 6-25 shows the syntax diagram for specifying the aggregation group.

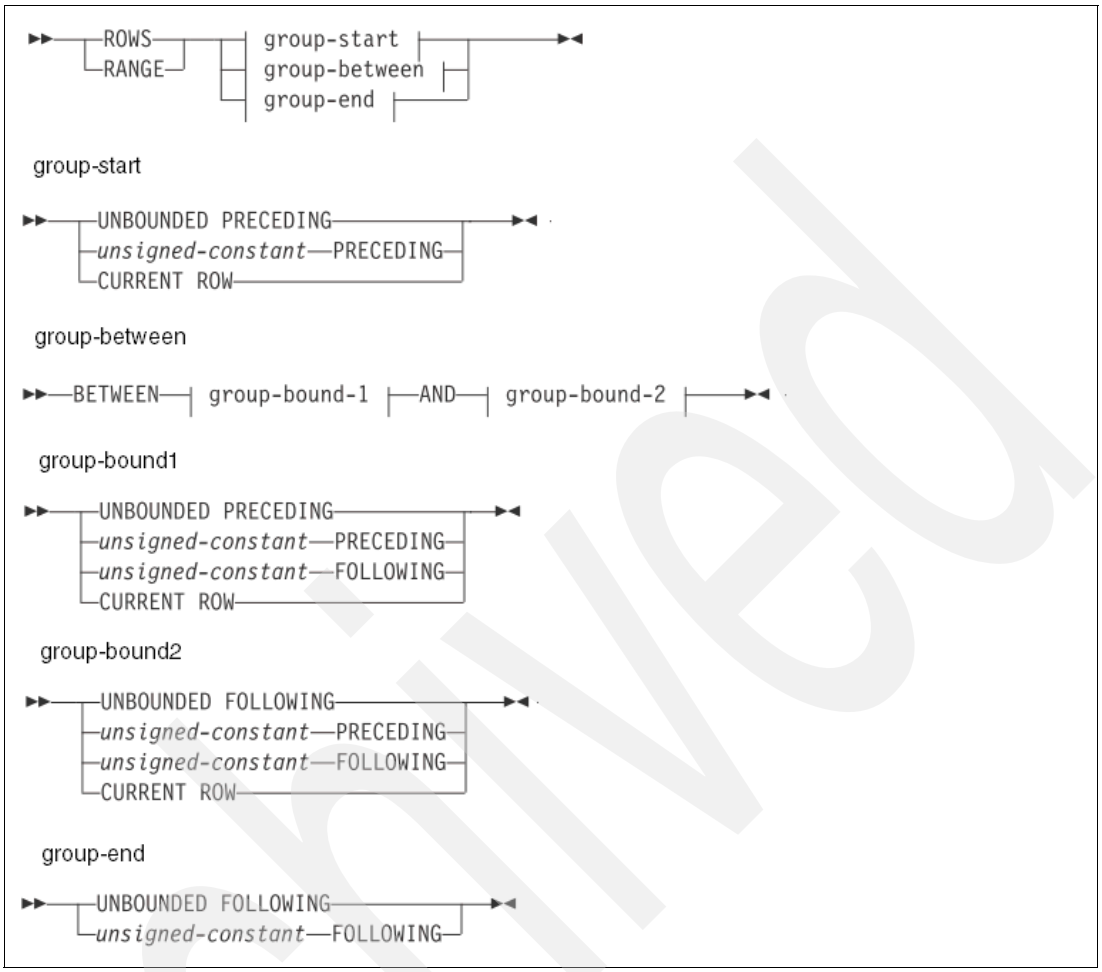


Figure 6-25 Window aggregation-group-clause

Only sensible combination for the starting and ending bounds is supported. Table 6-17 shows supported combinations. An entry of Y indicates that the given combination is supported, and an entry of N indicates that the combination is not allowed.

Table 6-17 Supported aggregation group boundary combinations

To: / From:	UNBOUNDED PRECEDING	n PRECEDING	CURRENT ROW	n FOLLOWING	UNBOUNDED FOLLOWING
UNBOUNDED PRECEDING	N	Y	Y	Y	Y
n PRECEDING	N	Y	Y	Y	Y
CURRENT ROW	N	N	Y	Y	Y
n FOLLOWING	N	N	N	Y	Y
UNBOUNDED FOLLOWING	N	N	N	N	N

Note that combinations such as BETWEEN 1 PRECEDING AND 3 PRECEDING or BETWEEN 3 FOLLOWING AND 1 FOLLOWING are allowed. However, these examples

Archived

result in empty aggregation sets, since the bounds are not stated in the proper order. This is analogous to the predicate BETWEEN 3 AND 1, which is never true, because 3 > 1.

We discuss the supported aggregation group boundary combinations in more detail with some examples.

BETWEEN UNBOUNDED PRECEDING AND n PRECEDING

The start of the group is unbounded, meaning that the start of the group is the first row of the partition. The end of the group precedes the current row by *n*, which means that the current row is not in the group. This grouping is referred to as an *uncentered* aggregation.

The *n* is an unsigned constant and specifies either the range or the number of rows that precede the current row. If ROWS is specified, *n* must be zero or a positive integer that indicates a number of rows. If RANGE is specified, the data type of *n* must be comparable to the data type of the sort key of the window-order-clause. Only one sort key is allowed, and the data type of the sort key must allow subtraction and addition. Example 6-60 shows a query that calculates the average number of APARs opened by Eugene for all the preceding months up to and excluding the current month.

Example 6-60 OLAP-specification aggregation group boundary example 1

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE BETWEEN
                                  UNBOUNDED PRECEDING AND
                                  00000100. PRECEDING    ) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10      ?
--          6  2010-02-01     5     10
--          8  2010-03-01     3      7
--         11  2010-04-01     5      6
--         13  2010-05-01    10      5
--         15  2010-06-01    15      6
```

BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

The start of the group is unbounded, meaning that the start of the group will be the first row of the partition. The end of the group is the current row. This specification is equivalent to specifying UNBOUNDED PRECEDING. If ROWS is specified, the current row is the aggregation group end boundary. If RANGE is specified, the aggregation group end boundary includes all of the rows with the same sort key value as the sort key value of the current row (that is rows that follow the current row will be included if they are duplicates of the current row). This is a default specification (with RANGE) if window-order-clause is specified.Example 6-61 shows a query that calculates the average number of APARs opened by Eugene for all the preceding months up to and including the current month,

Example 6-61 OLAP-specification aggregation group boundary example 2

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE UNBOUNDED PRECEDING) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10      10
```

```
--          6  2010-02-01     5       7
--          8  2010-03-01     3       6
--         11  2010-04-01     5       5
--         13  2010-05-01    10       6
--         15  2010-06-01    15       8
```

BETWEEN UNBOUNDED PRECEDING AND n FOLLOWING

The start of the group is unbounded, meaning that the start of the group will be the first row of the partition. The end of the group follows the current row by *n* (refer to BETWEEN UNBOUNDED PRECEDING AND n PRECEDING on page 196 for description of *n*).

Example 6-62 shows a query that calculates the average number of APARs opened by Eugene for all the preceding months up to one month following the current month.

Example 6-62 OLAP-specification aggregation group boundary example 3

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE BETWEEN
                                    UNBOUNDED PRECEDING AND
                                    00000100. FOLLOWING    ) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10       7
--          6  2010-02-01     5       6
--          8  2010-03-01     3       5
--         11  2010-04-01     5       6
--         13  2010-05-01    10       8
--         15  2010-06-01    15       8
```

BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

The start of the group is unbounded. The end of the group is unbounded as well, meaning that the end of the group will be the last row of the partition. In other words, entire partition is part of the group. This is referred to as *reporting* aggregate. This is a default specification if window-order-clause is not specified. Example 6-63 shows a query that calculates the average number of APARs opened by Eugene over all available months.

Example 6-63 OLAP-specification aggregation group boundary example 4

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER() AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10       8
--          6  2010-02-01     5       8
--          8  2010-03-01     3       8
--         11  2010-04-01     5       8
--         13  2010-05-01    10       8
--         15  2010-06-01    15       8
```

Archive

BETWEEN n PRECEDING AND m PRECEDING

The start of the group precedes the current row by *n*. The end of the group precedes the current row by *m,* which this means that the current row is not in the group. This grouping is referred to as an *uncentered* aggregation.

The *n* and *m* are unsigned constants and specify either the range or the number of rows that precede the current row. If ROWS is specified, *n* and *m* must be zero or a positive integer that indicates a number of rows. If RANGE is specified, the data type of *n* and *m* must be comparable to the data type of the sort key of the window-order-clause. Only one sort key is allowed, and the data type of the sort key must allow subtraction. If $n < m$, no row would ever be in a group, so result would be a null value.

Example 6-64 shows a query that calculates the average number of APARs opened by Eugene from two to one months preceding the current month.

Example 6-64 OLAP-specification aggregation group boundary example 5

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE BETWEEN
                                    00000200. PRECEDING AND
                                    00000100. PRECEDING    ) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10      ?
--          6  2010-02-01     5     10
--          8  2010-03-01     3      7
--         11  2010-04-01     5      4
--         13  2010-05-01    10      4
--         15  2010-06-01    15      7
```

BETWEEN n PRECEDING AND CURRENT ROW

The start of the group precedes the current row by *n*. The end of the group is the current row. This specification is equivalent to specifying n PRECEDING.

Example 6-65 shows a query that calculates the average number of APARs opened by Eugene from three preceding months up to and including the current month.

Example 6-65 OLAP-specification aggregation group boundary example 6

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE BETWEEN
                                    00000300. PRECEDING AND
                                    CURRENT ROW            ) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10      10
--          6  2010-02-01     5       7
--          8  2010-03-01     3       6
--         11  2010-04-01     5       5
--         13  2010-05-01    10       5
--         15  2010-06-01    15       8
```

BETWEEN n PRECEDING AND m FOLLOWING

The start of the group precedes the current row by *n*. The end of the group follows the current row by *m*.

Example 6-66 shows a query that calculates the average number of APARs opened by Eugene from one months preceding to one month following the current month.

Example 6-66 OLAP-specification aggregation group boundary example 7

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE BETWEEN
                                    00000100. PRECEDING AND
                                    00000100. FOLLOWING    ) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10       7
--          6  2010-02-01     5       6
--          8  2010-03-01     3       4
--         11  2010-04-01     5       6
--         13  2010-05-01    10      10
--         15  2010-06-01    15      12
```

BETWEEN n PRECEDING AND UNBOUNDED FOLLOWING

The start of the group precedes the current row by *n*. The end of the group is unbounded. Example 6-67 shows a query that calculates the average number of APARs opened by Eugene from one preceding months up to the rest of the available months.

Example 6-67 OLAP-specification aggregation group boundary example 8

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE BETWEEN
                                    00000100. PRECEDING AND
                                    UNBOUNDED FOLLOWING    ) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10       8
--          6  2010-02-01     5       8
--          8  2010-03-01     3       7
--         11  2010-04-01     5       8
--         13  2010-05-01    10      10
--         15  2010-06-01    15      12
```

BETWEEN CURRENT ROW AND CURRENT ROW

The start of the group is the current row. The end of the group is the current row as well, meaning that only the current row is in the group (remember that for RANGE, any duplicate rows are in the group as well).

BETWEEN CURRENT ROW AND n FOLLOWING

The start of the group is the current row. The end of the group follows the current row by *n*. This is equivalent to specifying n FOLLOWING.

Archived

Example 6-68 shows a query that calculates the average number of APARs opened by Eugene for current month and three following months.

Example 6-68 OLAP-specification aggregation group boundary example 9

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE BETWEEN
                                    CURRENT ROW AND
                                    00000300. FOLLOWING) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10       5
--          6  2010-02-01     5       5
--          8  2010-03-01     3       8
--         11  2010-04-01     5      10
--         13  2010-05-01    10      12
--         15  2010-06-01    15      15
```

BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING

The start of the group is the current row. The end of the group is unbounded. This is equivalent to specifying UNBOUNDED FOLLOWING. Example 6-69 shows a query that calculates the average number of APARs opened by Eugene from current month up to the rest of the available months.

Example 6-69 OLAP-specification aggregation group boundary example 10

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE UNBOUNDED FOLLOWING) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10       8
--          6  2010-02-01     5       7
--          8  2010-03-01     3       8
--         11  2010-04-01     5      10
--         13  2010-05-01    10      12
--         15  2010-06-01    15      15
```

BETWEEN n FOLLOWING AND m FOLLOWING

The start of the group follows the current row by *n*. The end of the group follows the current row by *m*, which means that the current row is not in the group. This grouping is referred to as an *uncentered* aggregation.

Example 6-70 shows a query that calculates the average number of APARs opened by Eugene from one to two months following the current month.

Example 6-70 OLAP-specification aggregation group boundary example 11

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE BETWEEN
                                    00000100. FOLLOWING AND
                                    00000200. FOLLOWING    ) AVG
```

```
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10       4
--          6  2010-02-01     5       4
--          8  2010-03-01     3       7
--         11  2010-04-01     5      12
--         13  2010-05-01    10      15
--         15  2010-06-01    15       ?
```

BETWEEN n FOLLOWING AND UNBOUNDED FOLLOWING

The start of the group follows the current row by *n*. The end of the group is unbounded. meaning that the end of the group is the last row in the partition, which means that the current row is not in the group. This grouping is referred to as an *uncentered* aggregation.

Example 6-71 shows a query that calculates the average number of APARs opened by Eugene from one following month up to the rest of the available months.

Example 6-71 OLAP-specification aggregation group boundary example 12

```
SELECT ID, REPORT_DATE, APARS_OPENED #,
       AVG(APARS_OPENED) OVER(ORDER BY REPORT_DATE
                              RANGE BETWEEN
                                  00000100. FOLLOWING AND
                                  UNBOUNDED FOLLOWING    ) AVG
FROM   TRYOLAP
WHERE  NAME = 'Eugene';
-- result: ID  REPORT_DATE    #     AVG
--          3  2010-01-01    10       7
--          6  2010-02-01     5       8
--          8  2010-03-01     3      10
--         11  2010-04-01     5      12
--         13  2010-05-01    10      15
--         15  2010-06-01    15       ?
```

Figure 6-26 shows the syntax diagram for the overall aggregation specification.

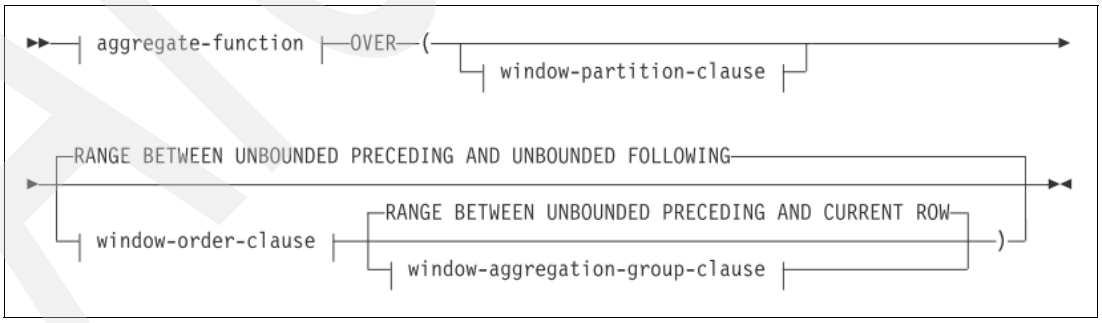


Figure 6-26 Aggregation specification syntax

The supported aggregate functions are: AVG, CORRELATION, COUNT, COUNT_BIG, COVARIANCE, MAX, MIN, STDDEV, SUM, VARIANCE. You cannot specify DISTINCT or ALL for any of those functions when used in the context of OLAP aggregation specification.

Archived

Archived

7

Application enablement

In this chapter, we describe DB2 10 for z/OS functions that are not strictly confined to SQL that provide infrastructure support for new applications or that simplify portability of existing applications to DB2 for z/OS from other database systems.

We discuss the following topics:

- Support for temporal tables and versioning
- Access plan stability and instance-based statement hints
- Addition of extended indicator variables
- New Universal Language Interface program (DSNULI)
- Access to currently committed data
- EXPLAIN MODE special register to explain dynamic SQL
- Save LASTUSED information for packages

Archived

© Copyright IBM Corp. 2010. All rights reserved.

7.1 Support for temporal tables and versioning

One of the major functions added in DB2 10 for z/OS is the support for temporal tables and versioning. In this section, we provide a brief background of the importance of having some temporal-related capabilities in the DBMS and then dive into the details of the DB2 implementation of such support.

The term *temporal* pertains to the concept of time. The concept of associating data with some point in time is paramount to practically every aspect of today's business. Just to give a few examples of the variety of useful applications of maintaining some historic data:

- A company might want to keep track of when certain customer policy or contract was in effect, when does the existing policy or contract expire or is eligible for an update. This is applicable to practically any type of business (health care, insurance, banking, and other type of businesses).
- A regulatory or compliance law might require to maintain historic data over long periods of time and have this data easily accessible. For example, in this case, you need to maintain the historical data as well as updates to data that is associated with the past activities. An update operation needs to preserve and not affect the past history.
- Some data recovery tasks might be possible or be simplified if the data changes are recorded automatically by the system. For example, a row-level recovery might be possible.

Without some temporal capability provided by the DBMS, application developers and database administrators must manage different versions of application data. Such tasks can involve complicated, difficult-to-maintain, scenarios (for example, many triggers or complex logic of keeping copies of the tables in synch) and might pose performance concerns. With ever growing (sometimes exponentially) data creation and collection, there is a strong demand for temporal capability inside the DBMS.

DB2 10 for z/OS introduces support for temporal tables and data versioning. Starting with DB2 10 new-function mode, you can define temporal attributes for a table and enable versioning of the table's data. This capability can reduce the complexity of applications that do (or plan to do) this type of processing and might improve application performance (by moving the logic from application level into the database system level).

DB2 defines a notion of a *period*, which is a time interval represented by a start time and an end time. Note that we use the term *time* here in a general sense, denoting some form of a time representation, be that a date or a timestamp. Also note that there are other possible time interval representations, such as a start time along with a duration; however, DB2 only supports time intervals that are expressed in terms of the starting and ending times. A start time of a period is considered to be included into that period, and the end time of the period is considered to be excluded from that period.

Important time interval note: It is important to reiterate that a period in DB2 for z/OS is a (closed, opened) time interval, also referred to as *half-open time interval* or *(inclusive, exclusive) time interval*. Be cautious when developing applications (or porting existing applications), as your organization might be using a different convention (for example, another popular convention is (closed, closed) time interval where the end time is included into the interval).

A period is implemented using two columns:

- One column contains the start value of a period.
- The other column contains the end value of a period.

The allowed data type for those columns depends on the type of the period and is either a DATE, a TIMESTAMP(6) WITHOUT TIME ZONE, or a TIMESTAMP(12) WITHOUT TIME ZONE. Time zone support for periods is not provided at this time; all values in those columns are local datetime values and are not in UTC.

There are two types of time periods that DB2 supports:

system period	A pair of columns with system-maintained values that indicate the period of time when a row is valid, which can also be referred to as SYSTEM_TIME period.
application period	A pair of columns with application-maintained values that indicate the period of time when a row is valid, which can also be referred to as BUSINESS_TIME period.

Each of the two period types is geared towards supporting a specific temporal capability. The system period is intended for supporting a concept of data versioning and is used for providing the system maintained history of the data. The application period is intended for supporting a user-controlled way of establishing the notion of validly of the data. It allows user applications to specify and control the periods when certain data is considered valid to the user.

7.1.1 System period

As its name implies, system period is managed by the system itself rather than by an application or a database administrator. System period is used by DB2 for automatic maintenance of historical data. That is, DB2 automatically keeps track of when a row is inserted, updated, or deleted from a table. The old rows (deleted or updated) get archived into another table. This automatic maintenance is referred to as *system-period data versioning*. A table that has the system-period data versioning enabled is called *system-period temporal table*. A system-period temporal table contains current (active) rows of the table, and the table that contains the archived old rows is called a *history table*.

Defining system period and enabling system-period data versioning

The system period has a name of SYSTEM_TIME and is defined on a row-begin column and a row-end column. Both of these columns must be defined as TIMESTAMP(12) WITHOUT TIME ZONE NOT NULL with a GENERATED attribute that designates these columns as row-begin and row-end.

Example 7-1 shows how a system period can be defined for a table. In this example, we define two timestamp columns and specify that they are to represent a row-begin and a row-end. Then, we define the system period SYSTEM_TIME using those two columns.

Example 7-1 SYSTEM_TIME definition for a table

```
CREATE TABLE TRYTEMPORAL_S(
 POLICY        INTEGER
,ATTRIBUTE     VARCHAR(30)
,STATUS        VARCHAR(20)
```

```
,P_FROM        TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN
,P_TO          TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END

,PERIOD SYSTEM_TIME(P_FROM, P_TO)
);
```

Note that specifying the data type for the row-begin and the row-end is optional. For example, you could define P_FROM as:

```
F_FROM NOT NULL GENERATED AS ROW BEGIN
```

After the PERIOD is defined, it cannot be dropped from a table. This behavior is similar to how a column cannot be dropped from a table.

For a column declared AS ROW BEGIN, DB2 generates a value for each row that is inserted and for every row in which any column is updated. The generated value is a timestamp that corresponds to the start time associated with the most recent transaction. If multiple rows are inserted with a single SQL statement, the values for the transaction start timestamp column are the same. A table can have only one column defined as a ROW BEGIN timestamp column.

For a column declared AS ROW END, DB2 generates a maximum value for each row that is inserted and for every row in which any column is updated. If versioning is active when a row is deleted through an update or delete operation, the value that is generated will be a timestamp that corresponds to the most recent transaction start time that is associated with the transaction. If an update to a row causes the ROW END value to be less than or equal to the ROW BEGIN value, an error is returned. A table can have only one column defined as a ROW END timestamp column.

You can query the SYSIBM.SYSCOLUMNS catalog table to see the information about a column that is part of a period. The DEFAULT column of SYSIBM.SYSCOLUMNS contains a value of ‘Q’ for any column defined with the AS ROW BEGIN attribute, and a value of ‘R’ for any column defined with the AS ROW END attribute. The PERIOD column of SYSIBM.SYSCOLUMNS contains a value of ‘S’ for a column that is the start of the SYSTEM_TIME period and a value of ‘T’ for the column that is the end of the SYSTEM_TIME period.

Defining a system period alone might not be useful, because DB2 is only recording timestamp information for times of the row’s insert or last update. To get value from the system period, you need to enable system-period data versioning for the table. To enable data versioning for a table, the table, in addition to the definition of the system period, must have a column that is defined with the AS TRANSACTION START ID attribute, and a history table that is associated with that table must exist. Only then can you enable the system-period data versioning for the table.

We demonstrate this in Example 7-2. We first amend our table definition to include another timestamp column with the AS TRANSACTION START ID attribute. Next we create the history table. And finally, we enable the system-period data versioning for our table with the ALTER ADD VERSIONING USE HISTORY TABLE alter statement.

Example 7-2 Creating a system-period temporal table and enabling data versioning

```
CREATE TABLE TRYTEMPORAL_S(
 POLICY        INTEGER
,ATTRIBUTE     VARCHAR(30)
,STATUS        VARCHAR(20)
,P_FROM        TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN
```

```
,P_TO          TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END
,TRANS_ID      TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID

,PERIOD SYSTEM_TIME(P_FROM, P_TO)
);
-- create history table:
CREATE TABLE HIST_TRYTEMPORAL_S(
 POLICY        INTEGER
,ATTRIBUTE     VARCHAR(30)
,STATUS        VARCHAR(20)
,P_FROM        TIMESTAMP(12) NOT NULL
,P_TO          TIMESTAMP(12) NOT NULL
,TRANS_ID      TIMESTAMP(12) NOT NULL
);
-- enable system-period data versioning:
ALTER TABLE TRYTEMPORAL_S
 ADD VERSIONING USE HISTORY TABLE HIST_TRYTEMPORAL_S;
-- table TRYTEMPORAL_S is now a system-period temporal table.
```

You can query the SYSIBM.SYSTABLES catalog table to verify that the system-period data versioning is enabled for the table. The columns VERSIONING_SCHEMA and VERSIONING_TABLE for the row associated with the system-period temporal table contain the schema name and the table name of the corresponding history table. In addition, the VERSIONING_SCHEMA and VERSIONING_TABLE columns for the row that is associated with the history table contain the schema name and the table name of the corresponding system-period temporal table. In addition, the TYPE column for the row associated with a history table has a value of ‘H’. See Example 7-3.

Example 7-3 Example of SYSIBM.SYSTABLE data versioning information

```
SELECT CREATOR || '.' || NAME AS TABLE, TYPE,
       VERSIONING_SCHEMA || '.' || VERSIONING_TABLE AS VERSIONING_TABLE
FROM   SYSIBM.SYSTABLES
WHERE  NAME LIKE '%TRYTEMPORAL_S';
-- result:   TABLE                  TYPE        VERSIONING_TABLE
--         DB2R4.TRYTEMPORAL_S       T       DB2R4.HIST_TRYTEMPORAL_S
--         DB2R4.HIST_TRYTEMPORAL_S  H       DB2R4.TRYTEMPORAL_S
```

Most likely, you have existing tables that you want to enable for system-period data versioning, instead of creating new table. In this case, you need to perform a series of alters to the existing table as shown in Example 7-4.

Example 7-4 Enabling system-period data versioning for an existing table

```
-- assume the following table exists:
-- TRYTEMPORAL_E(POLICY INTEGER, ATTRIBUTE VARCHAR(30), STATUS VARCHAR(20))
-- first, ALTER the existing table:
ALTER TABLE TRYTEMPORAL_E
  ADD COLUMN SYS_BEGIN TIMESTAMP(12) NOT NULL
                       GENERATED ALWAYS AS ROW BEGIN
  ADD COLUMN SYS_END   TIMESTAMP(12) NOT NULL
                       GENERATED ALWAYS AS ROW END
  ADD COLUMN TRANS_ID  TIMESTAMP(12) NOT NULL
                       GENERATED ALWAYS AS TRANSACTION START ID
  ADD PERIOD SYSTEM_TIME(SYS_BEGIN, SYS_END);
-- then create the history table:
```

```
CREATE TABLE HIST_TRYTEMPORAL_E(
 POLICY     INTEGER
,ATTRIBUTE  VARCHAR(30)
,STATUS     VARCHAR(20)
,SYS_BEGIN  TIMESTAMP(12) NOT NULL
,SYS_END    TIMESTAMP(12) NOT NULL
,TRANS_ID   TIMESTAMP(12) NOT NULL)
);
-- finally, alter to enable the data versioning:
ALTER TABLE TRYTEMPORAL_E
  ADD VERSIONING USE HISTORY TABLE HIST_TRYTEMPORAL_E;
```

In Example 7-4, we assumed that there were no existing timestamp columns that you want to become the basis for SYSTEM_TIME period. In this case, you use ADD COLUMN clause of the ALTER statement to add the new columns.

Note that we used one ALTER statement to make multiple alternations to the table, but using separate alters works also. After the alter takes effect, all existing rows in the table will have the same initial value of 0001-01-01-00.00.00.000000000000 in the SYS_BEGIN column (which is the row-begin column) and the same initial value of 9999-12-31-24.00.00.000000000000 in the SYS_END column (which is the row-end column).

Alternatively, you might have existing timestamp columns that you want to become the basis for SYSTEM_TIME period. In this case, you would use ALTER COLUMN clause of the ALTER statement to change the existing column to the row-begin or row-end column. Note that in this case you might need to perform the following alterations.

1. First, change the data type from timestamp(6) to timestamp(12), because 6 is the default precision and the only precision that is available prior to DB2 10 for z/OS.
2. Second, change the column to be GENERATED ALWAYS as either ROW BEGIN or ROW END. After such alter takes effect, the values in the SYS_BEGIN and SYS_END in all existing rows would not be affected (would be preserved). Because these two columns become un-updatable by the user, you might want to ensure they contain expected values prior to altering these columns to row-begin or row-end columns.

After the alteration of the existing table (as in Example 7-4) or after the creation of the new table (as in Example 7-2), the table has the row-begin and row-end columns and the SYSTEM_TIME period defined based on those columns.

The next step is to create the *history table*. The history table must have the same structure as the associated system-period temporal table (the table we just altered) minus the PERIOD definition. After you create the history table, you can enable data versioning to alter the system-period temporal table, specifying the ADD VERSIONING USE HISTORY TABLE clause. When this alter takes effect, DB2 automatically handles updates and deletes by moving the changed (or deleted) rows into the associated history table.

If you skip the last step (or if it fails), notice that the system-period temporal table behaves as expected, as though the data versioning was enabled. That is, DB2 updates the values in the row-begin and row-end columns for any updates or deletes accordingly (because this action is completed automatically by the nature of the GENERATED columns), and the deleted rows are removed from the table, as expected. However, what is missing are the corresponding changes to the history table. Because the ALTER ADD VERSIONING USE HISTORY TABLE ties the system-period temporal table and the history table together, without that link, DB2 has no knowledge about the history table and the data versioning is not enabled.

In summary, to create a temporal table with system-period data versioning enabled, you need to have two tables:

- The first table is the system-period temporal table, which contains the currently active rows. (This table is your existing table or newly created table that you expect to be queried by the applications and users.)
- The second table is the history table, which contains the old (deleted or updated) rows. (This table is a newly created table that, typically, is not queried directly by the applications or users for normal processing.)

After the two tables are created, you issue an ALTER TABLE statement with the ADD VERSIONING clause to establish the system-period data versioning relationship between the two tables. The ADD VERSIONING clause is specified on an ALTER TABLE statement for the table to be defined as the system-period temporal table (that is, the table with system-period data versioning), and not the history table. The history table is identified in the USE HISTORY TABLE clause of the ADD VERSIONING clause of the ALTER TABLE statement for the system-period temporal table.

You can disable the system-period data versioning for a table by using a DROP VERSIONING clause of the ALTER table statement.

Requirements

To enable system-period data versioning requires the conditions that we list here.

The system-period temporal table must have:

- A column defined as TIMESTAMP(12) WITHOUT TIME ZONE, NOT NULL, with the attribute for a begin column of a SYSTEM_TIME period, namely, GENERATED ALWAYS AS ROW BEGIN. Note that this column is not updatable.
- A column defined as TIMESTAMP(12) WITHOUT TIME ZONE, NOT NULL, with the attribute for an end column of a SYSTEM_TIME period, namely, GENERATED ALWAYS AS ROW END. Note that this column is not updatable.
- A column defined as TIMESTAMP(12) WITHOUT TIME ZONE, NOT NULL, with the attribute for a transaction-start-ID, namely, GENERATED ALWAYS AS TRANSACTION_START_ID. Note that this column is not updatable.
- A period, SYSTEM_TIME, specified on the first two timestamp columns described earlier in which the first column is a row-begin column and the second column is a row-end column.

The system-period temporal table must *not* have:

- A security label column
- Any row permissions
- Any column masks
- A clone table defined on it
- A materialized query table definition
- An incomplete table definition
- Any other tables defined in the same table space

The history table must have:

- The same number of columns as the system-period temporal table, minus the PERIOD.
- The columns must have the same corresponding names, data type, null attribute, CCSID, FOR BIT or SBCS. or mixed data attribute, hidden attribute and fieldproc as the system-period temporal table.

Note, that GENERATED attributes do not have to match. The system-period temporal table might have generated columns (such as identity column or row-change-timestamp), and the data versioning can still be supported. You must declare the corresponding columns in the history table without the use of the GENERATED keyword and declare only the matching data type and nullability.

Example 7-5 demonstrates this concept. We create a history table for a system-period temporal table that contains identity column and the row-change-timestamp column.

Example 7-5 System-period data versioning with generated columns

```
CREATE TABLE TRYTEMPORAL_G(
 ID         INTEGER       NOT NULL GENERATED ALWAYS AS IDENTITY
,POLICY     INTEGER
,ATTRIBUTE  VARCHAR(30)
,STATUS     VARCHAR(20)
,ROW_CHANGE TIMESTAMP(6)  NOT NULL GENERATED FOR EACH ROW ON UPDATE
                                   AS ROW CHANGE TIMESTAMP
,SYS_BEGIN  TIMESTAMP(12) NOT NULL GENERATED AS ROW BEGIN
,SYS_END    TIMESTAMP(12) NOT NULL GENERATED AS ROW END
,TRANS_ID   TIMESTAMP(12)          GENERATED AS TRANSACTION START ID

,PERIOD SYSTEM_TIME(SYS_BEGIN, SYS_END)
);

CREATE TABLE HIST_TRYTEMPORAL_G(
 ID         INTEGER       NOT NULL
,POLICY     INTEGER
,ATTRIBUTE  VARCHAR(30)
,STATUS     VARCHAR(20)
,ROW_CHANGE TIMESTAMP(6)  NOT NULL
,SYS_BEGIN  TIMESTAMP(12) NOT NULL
,SYS_END    TIMESTAMP(12) NOT NULL
,TRANS_ID   TIMESTAMP(12)
);
```

The history table must *not* have:

- A security label, identity, row change timestamp, row-begin, row-end, or transaction-start-ID column
- A generated column other than a ROWID. Note that ROWID values in the history table will have new values, and values from the associated system-period temporal table will not be preserved.
- A period definition
- Any row permissions
- Any column masks
- A clone table defined on it
- Only one table defined in the table space.
- A materialized query table definition
- An incomplete table definition

Archived

Additional restrictions for system-period data versioning include:

- For point-in-time recovery to keep the system-period temporal table and history data in sync, the history table space and system-period temporal table space must be recovered as a set and cannot be done individually unless the keyword VERIFYSET NO is used.
- No utility operation is allowed that will delete data from the system-period temporal table. This includes LOAD REPLACE, REORG DISCARD, and CHECK DELETE YES.
- A schema attribute (data type, add column, and other such attributes) cannot be altered for a system-period temporal table or history table.
- A history table, table space containing a history table, or database containing a history table cannot be explicitly dropped. A history table is implicitly dropped when the associated system-period temporal table is dropped.
- A clone table cannot be defined for a system-period temporal table or history table.
- A table cannot be created in a table space that contains a system-period temporal table or history table. In other words, both tables must be created in their own separate table spaces, and there could be no other tables in those table spaces.
- A system-period temporal table, or a column of a system-period temporal table, cannot be renamed.
- A history table, or a column of a history table, cannot be renamed.

Manipulating data with system-period data versioning

After you enable the system-period data versioning for a table, DB2 automatically manages the data changes in the table. Let us first observe some insert/update/delete behavior and then discuss the ways you can query the historic data. For the following examples, we use the TRYTEMPORAL_S table that we defined in Example 7-2 on page 206. Assuming that we start with an empty table, we insert, update, and delete rows at different points in time.

We first insert data. Example 7-6 shows the insert statements performed at various times and the resulting values populated into the table. For brevity, we truncate the fractional second part of the timestamps on output. Note that we cannot explicitly insert into the three timestamp columns, which we defined as GENERATED.

Example 7-6 DML with system-period data versioning: INSERT

```
-- assume we perform this INSERT at the following time: 2010-01-01 08:30:00
INSERT INTO TRYTEMPORAL_S(POLICY, ATTRIBUTE, STATUS) VALUES(
 1, 'Purple', 'Under reiew');
-- assume we perform this INSERT at the following time: 2010-08-01 10:45:30
INSERT INTO TRYTEMPORAL_S(POLICY, ATTRIBUTE, STATUS) VALUES(
 2, 'Blue', 'Under reiew');
-- assume we perform this INSERT at the following time: 2010-05-01 15:05:51
INSERT INTO TRYTEMPORAL_S(POLICY, ATTRIBUTE, STATUS) VALUES(
 3, 'Yellow', 'Under reiew');

SELECT POLICY, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM   TRYTEMPORAL_S;
-- result: POLICY          FROM                    TO
--              1   2010-01-01-08.30.00   9999-12-31-24.00.00
--              2   2010-08-01-10.45.30   9999-12-31-24.00.00
--              3   2010-05-01-15.05.51   9999-12-31-24.00.00

SELECT POLICY, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM   HIST_TRYTEMPORAL_S;
-- result: no rows returned
```

From the insert example, we observe that DB2 automatically recorded the time of the insert transaction in the P_FROM column, which is defined as the row-begin column. And DB2 also recorded a maximum possible timestamp value in the P_TO column, which is defined as the row-end column. The maximum timestamp value in the row-end column indicates that the row is current (valid at the present time). In fact, all rows in the system-period temporal table should have the row-end column set to some future time because this table should contain only the current rows (other rows are moved to the history table).

Thus, the SYSTEM_TIME row-end column in a system-period temporal table typically contains a timestamp value of 9999-12-31-24.00.00.000000000000. The only exception is if you originally had this column as a normal timestamp column and later altered it to be the row-end column. In that case, the pre-existing values is preserved after the alter to enable the system-period data versioning.

Let us take a look at what happens when data in system-period temporal table is updated. Example 7-7 presents a scenario where we update one row in our table TRYTEMPORAL_S and we observe what changes does DB2 make for the row-begin and row-end columns.

Example 7-7 DML with system-period data versioning: Update

```
-- assume we perform this UPDATE at the following time: 2010-09-01 16:00:00
UPDATE TRYTEMPORAL_S
 SET STATUS = 'Approved'
WHERE  POLICY = 2;

SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM   TRYTEMPORAL_S;
-- result: POLICY   STATUS               FROM                  TO
--              1   Under review    2010-01-01-08.30.00   9999-12-31-24.00.00
--              2   Approved        2010-09-01-16.00.00   9999-12-31-24.00.00
--              3   Under review    2010-05-01-15.05.51   9999-12-31-24.00.00

SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM   HIST_TRYTEMPORAL_S;
-- result: POLICY   STATUS               FROM                  TO
--              2   Under review    2010-08-01-10.45.30   2010-09-01-16.00.00
```

We observe that after updating the row with policy number 2 to *Approved* status, the new P_FROM value for the updated row is the time the update transaction took place, and the history table now contains the row with the old status value and the time period of when that old value was in effect (valid).

Next, we take a look at the delete behavior, which is presented in Example 7-8.

Example 7-8 DML with system-period data versioning: Delete

```
-- assume we perform this DELETE at the following time: 2010-09-02 12:34:56
DELETE FROM TRYTEMPORAL_S
WHERE  POLICY = 1;

SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM   TRYTEMPORAL_S;
-- result: POLICY   STATUS               FROM                  TO
--              2   Approved       2010-09-01-16.00.00   9999-12-31-24.00.00
--              3   Under reiew    2010-05-01-15.05.51   9999-12-31-24.00.00
SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM   HIST_TRYTEMPORAL_S;
```

```
-- result: POLICY   STATUS               FROM                    TO
--               2   Under reiew    2010-08-01-10.45.30   2010-09-01-16.00.00
--               1   Under reiew    2010-01-01-08.30.00   2010-09-02 12:34:56
```

We observe that after deleting the row with policy number 1, the row is no longer present in the system-period temporal table, and a new row has been inserted into the history table with the time period of when that row was active (valid).

There are several ways for you to access the data from the tables with system-period data versioning. For example, if you are concerned only with the current data, you can query the system-period temporal table. Additionally, if you need only historical data, you can query the history table. If you are interested in both current and historical data, query both tables, and combine the results.

To improve the usability and to simplify accessing the historical data, DB2 extends the FROM clause of the table-reference. You can specify that historical data is requested from the system-period temporal table by using the following period-specification clauses after the table name in the FROM clause:

- FOR SYSTEM_TIME AS OF *timestamp-expression*

  Specifies that the table-reference includes each row for which the begin value for the specified period is less than or equal to *timestamp-expression* and the end value for the period is greater than *timestamp-expression*.

- FOR SYSTEM_TIME FROM *timestamp-expression1* TO *timestamp-expression2*

  Specifies that the table-reference includes rows that exist for the period that is specified from *timestamp-expression1* up to *timestamp-expression2*. A row is included in the table-reference if the start value for the period in the row is less than *timestamp-expression2* and if the end value for the period in the row is greater than *timestamp-expression1*.

  Note that the *timestamp-expression2* is not inclusive, which means that any row that starts with *timestamp-expression2* will not qualify in the result.

- FOR SYSTEM_TIME BETWEEN *timestamp-expression1* AND *timestamp-expression2*

  Specifies that the table-reference includes rows in which the specified period overlaps at any point in time between *timestamp-expression1* and *timestamp-expression2*. A row is included in the table-reference if the start value for the period in the row is less than or equal to *timestamp-expression2* and if the end value for the period in the row is greater than *timestamp-expression1*.

  The table reference contains zero rows if *timestamp-expression1* is greater than *timestamp-expression2*. If *timestamp-expression1* = *timestamp-expression2*, the expression is equivalent to AS OF *timestamp-expression1*. If *timestamp-expression1* or *timestamp-expression2* is the null value, the table reference is an empty table.

  The *timestamp-expression2* is inclusive, which means that any row that starts with *timestamp-expression2* will qualify in the result.

Note that you specify the new period-specification clauses for a system-period temporal table, and not the history table. The history table is not referenced directly by the query. DB2 rewrites the query to include data from the history table with a UNION ALL operator.

If the table is a system-period temporal table and a period-specification for the SYSTEM_TIME period is not specified, the table reference includes all current rows of the table and does not include any historical rows of the table.

The period-specification on a FROM clause of a table reference should simplify coding of applications because only the system-period temporal table needs to be referenced by a query, regardless of whether you need to access current or historical data or both.

We now includes examples of querying a system-period temporal table using the period-specification on the FROM clause.

Example 7-9 shows the usage of the FOR SYSTEM_TIME AS OF clause.

Example 7-9 Example of FOR SYSTEM_TIME AS OF

```
-- assume table TRYTEMPORAL_S has the following data in POLICY, P_FROM, PT_TO:
-- POLICY              P_FROM                            P_TO
--      1  2001-01-01-00.00.00.000000000000  2008-01-01-00.00.00.000000000000
--      2  2002-01-01-00.00.00.000000000000  2006-01-01-00.00.00.000000000000
--      3  2003-01-01-00.00.00.000000000000  2004-01-01-00.00.00.000000000000
--      4  2005-01-01-00.00.00.000000000000  2007-01-01-00.00.00.000000000000

SELECT POLICY FROM TRYTEMPORAL_S
              FOR SYSTEM_TIME AS OF TIMESTAMP '2006-08-08 00.00.00';
-- result: POLICY
--              1
--              4

SELECT POLICY FROM TRYTEMPORAL_S
              FOR SYSTEM_TIME AS OF TIMESTAMP '2005-08-08 00.00.00';
-- result: POLICY
--              1
--              2
--              4
```

Example 7-10 shows the usage of the FOR SYSTEM_TIME FROM... TO clause. We assume the data in the TRYTEMPORAL_S table is the same as in the previous example. Note that POLICY 4 has the same start timestamp (2005-01-01...) as the one we specify in the TO for the first query; however, the row is not qualified because the FROM... TO is inclusive-exclusive. In addition, in the second query, POLICY 3 has the same end timestamp (2004-01-01...) as the one specified in the FROM; however, the row is not qualified because the SYSTEM_TIME period is inclusive-exclusive.

Example 7-10 Example of FOR SYSTEM_TIME FROM... TO

```
SELECT POLICY FROM TRYTEMPORAL_S
              FOR SYSTEM_TIME FROM  TIMESTAMP '2003-01-01 00.00.00'
                              TO    TIMESTAMP '2005-01-01 00.00.00';
-- result: POLICY
--              1
--              2
--              3

SELECT POLICY FROM TRYTEMPORAL_S
              FOR SYSTEM_TIME FROM  TIMESTAMP '2004-01-01 00.00.00'
                              TO    TIMESTAMP '2007-01-01 00.00.00';
-- result: POLICY
--              1
--              2
--              4
```

Example 7-11 shows the usage of the FOR SYSTEM_TIME BETWEEN... AND clause. We again, assume the same data in the TRYTEMPORAL_S table as in previous examples. We use the same values for the BETWEEN range as we used in the previous example for the FROM... TO range to demonstrate the difference between the two clauses. Because the BETWEEN option is inclusive-inclusive, the first query qualifies the row with POLICY 4. The second query does not qualify row with POLICY 3 for the same reason as in the previous example, and that is, the SYSTEM_TIME period is inclusive-exclusive.

Example 7-11 Example of FOR SYSTEM_TIME BETWEEN... AND

```
SELECT POLICY FROM TRYTEMPORAL_S
              FOR SYSTEM_TIME BETWEEN TIMESTAMP '2003-01-01 00.00.00'
                              AND     TIMESTAMP '2005-01-01 00.00.00'
-- result: POLICY
--              1
--              2
--              3
--              4

SELECT POLICY FROM TRYTEMPORAL_S
              FOR SYSTEM_TIME BETWEEN TIMESTAMP '2004-01-01 00.00.00'
                              AND     TIMESTAMP '2007-01-01 00.00.00';
-- result: POLICY
--              1
--              2
--              4
```

7.1.2 Application period

As its name implies, application period is managed by the application or a user. The users, not the DB2 system, control what values are stored for the application period. The application period can be considered as a period showing when a row is valid to the user. It has the property of modeling data in the past, present, and future, as the data values are controlled by the application.

With application period, DB2 also provides functionality to create a UNIQUE index that will be unique over a period of time. DB2 also extends UPDATE and DELETE syntax to provide capability to modify rows based upon the application period. And, of course, rows can be fetched based upon the application period as well (by using the same period-specification extension of the FROM clause that we describe in 7.1.1, “System period” on page 205).

Defining application period

The application period has a name of BUSINESS_TIME and is defined on a begin column and an end column. Both of these columns must be defined either as TIMESTAMP(6) WITHOUT TIME ZONE NOT NULL or as DATE NOT NULL. The data type for the begin column and end column must be the same.

Example 7-12 shows how an application period can be defined for a table. In this example, we define two date columns to serve as the begin and the end columns of the period. Then, we define the application period BUSINESS_TIME using those two columns. Note that after the PERIOD is defined, it cannot be dropped from a table. This behavior is similar to how a column cannot be dropped from a table.

Archived

Example 7-12 BUSINESS_TIME period definition for a table

```
CREATE TABLE TRYTEMPORAL_B(
 POLICY        INTEGER     NOT NULL
,ATTRIBUTE     VARCHAR(30)
,STATUS        VARCHAR(20)
,P_FROM        DATE        NOT NULL
,P_TO          DATE        NOT NULL

,PERIOD BUSINESS_TIME(P_FROM, P_TO)
);
```

You can query the SYSIBM.SYSCOLUMNS catalog table to see the information about a column that is part of a period. The PERIOD column of SYSIBM.SYSCOLUMNS contains a value of 'B' for a column that is the start of the BUSINESS_TIME period and a value of 'C' for the column that is the end of the BUSINESS_TIME period.

Tables with a BUSINESS_TIME period defined, can be referred to as the application-period temporal tables.

You can also define the temporal table with BUSINESS_TIME period to have a unique attribute that is unique for a period of time by specifying the BUSINESS_TIME WITHOUT OVERLAPS clause in CREATE TABLE, ALTER TABLE, or CREATE UNIQUE INDEX statements. This clause instructs DB2 to add the end column and begin column of the BUSINESS_TIME period to the index in ascending order and to enforce that there are no overlaps in time with respect to the specified index keys.

Example 7-13 demonstrates some ways to define the no-overlap enforcement. We show an ALTER TABLE and the CREATE UNIQUE INDEX statements. Note how the columns representing the beginning and the end of the BUSINESS_TIME period are not part of the key list.

Example 7-13 Examples of BUSINESS_TIME WITHOUT OVERLAPS clause

```
ALTER TABLE TRYTEMPORAL_B
  ADD UNIQUE(POLICY, BUSINESS_TIME WITHOUT OVERLAPS);

CREATE UNIQUE INDEX IDX_TRYTEMPORAL_B ON
  TRYTEMPORAL_B(POLICY, BUSINESS_TIME WITHOUT OVERLAPS);
```

After the BUSINESS_TIME period is established, you can query the business time data and update and delete data based upon a period of time. You need to make a similar extension to the FROM clause to allow the querying of the data based on the BUSINESS_TIME period:

- FOR BUSINESS_TIME AS OF *value*

  Specifies that the table-reference includes each row for which the begin value for the specified period is less than or equal to *value* and the end value for the period is greater than *value*.

- FOR BUSINESS_TIME FROM *value1* TO *value2*

  Specifies that the table-reference includes rows that exist for the period that is specified from*value1* up to *value2*. A row is included in the table-reference if the start value for the period in the row is less than *value2* and the end value for the period in the row is greater than *value1*.

  Note that the *value2* is not inclusive, which means that any row that starts with *value2* will not qualify in the result.

- FOR BUSINESS_TIME BETWEEN *value1* AND *value2*

  Specifies that the table-reference includes rows in which the specified period overlaps at any point in time between *value1* and *value2*. A row is included in the table-reference if the start value for the period in the row is less than or equal to *value2* and the end value for the period in the row is greater than *value1*. The table reference contains zero rows if *value1* is greater than *value2*. If *value1* = *value2*, the expression is equivalent to AS OF *value1*. If *value1* or *value2* is the null value, the table reference is an empty table.

  Note that the *value2* is inclusive, which means that any row that starts with *value2* will qualify in the result.

Thus, the semantics of FOR BUSINESS_TIME clause usage is the same as the one for the FOR SYSTEM_TIME clause. Example 7-14 provides one example of the usage of this clause.

Example 7-14 Example of FOR BUSINESS_TIME clause

```
-- assume table TRYTEMPORAL_B has the following data in POLICY, P_FROM, P_TO:
-- POLICY    P_FROM      P_TO
--      1  2001-01-01  2008-01-01
--      2  2002-01-01  2006-01-01
--      3  2003-01-01  2004-01-01
--      4  2005-01-01  2007-01-01

SELECT POLICY FROM TRYTEMPORAL_B
              FOR BUSINESS_TIME FROM DATE '2003-01-01'
                                TO   DATE '2005-01-01';
-- result: POLICY
--              1
--              2
--              3
```

In addition to the FROM clause extension, the UPDATE and DELETE statements are enhanced to allow updating and deleting based upon a period of time. A new clause FOR PORTION OF BUSINESS_TIME can be specified in UPDATE or DELETE statements as follows:

```
FOR PORTION OF BUSINESS_TIME FROM value1 TO value2
```

This clause specifies that the update or delete operation applies to rows for the period that is specified from *value1* to *value2*. No rows are deleted if *value1* is greater than or equal to *value2*, or if *value1* or *value2* is the null value. The specified period FROM *value1* TO *value2* is inclusive-exclusive.

The rows in a table for which the UPDATE or DELETE is issued can fall into one of the following categories:

- The row might be *not contained* in the specified period, which occurs when both columns of BUSINESS_TIME period are less than or equal to *value1* or are greater than or equal to *value2*.
- The row might be *fully contained* in the specified period, which occurs when the value for the begin column for BUSINESS_TIME period in the row is greater than or equal to *value1* and the value for the corresponding end column in the row is less than *value2.*
- The row might be *partially contained* in the specified period, which occurs when row overlaps the beginning of the specified period or the end of the specified period, but not both.

We say that the row overlaps the beginning of the specified period, if the value of the begin column is less than *value1* and the value of the end column is greater than *value1*. Similarly, we say that the row overlaps the end of the specified period, if the value of the end column is greater than or equal to *value2* and the value of the begin column is less than *value2*.

- The row can *fully overlap* the specified period, which occurs when the row overlaps the beginning of the specified period and overlaps the end of the specified period.

When updating or deleting a row, DB2 takes different actions, depending on the category to which the row belongs:

- If the row is *not contained* in the specified period, no update/delete occurs for that row.
- If the row is *fully contained* in the specified period, the row is updated, deleted.
- If the row is *partially contained* in the specified period, the row is updated/deleted and a new row is inserted using the original values from the row, except that:
  - If the row overlaps the beginning of the specified period, the end column is set to *value1*.
  - if the row overlaps the end of the specified period, the begin column is set to *value2*.
- if the row *fully overlaps* the specified period, the row is update/deleted and two new rows are inserted using the original values from the row, except that the begin column of one row is set to *value2* and the end column of another row is set to *value1*.

The following examples demonstrate all these cases. We assume the same data in the TRYTEMPORAL_B table as in the previous example. Example 7-15 is for the *not contained* case. We observe that no row is deleted.

Example 7-15 Example of FOR PORTION OF BUSINESS_TIME semantics: Case 1

```
-- assume table TRYTEMPORAL_B has the following data in POLICY, P_FROM, P_TO:
-- POLICY    P_FROM      P_TO
--      1  2001-01-01  2008-01-01
--      2  2002-01-01  2006-01-01
--      3  2003-01-01  2004-01-01
--      4  2005-01-01  2007-01-01
DELETE FROM TRYTEMPORAL_B
       FOR PORTION OF BUSINESS_TIME FROM DATE'1999-01-01'
                                    TO   DATE'2000-01-01';
SELECT POLICY, P_FROM, P_TO
FROM TRYTEMPORAL_B;
-- result: POLICY    P_FROM      P_TO
--              1  2001-01-01  2008-01-01
--              2  2002-01-01  2006-01-01
--              3  2003-01-01  2004-01-01
--              4  2005-01-01  2007-01-01
```

Example 7-16 shows the fully contained case. We observe that the row is deleted and no other rows are inserted.

Example 7-16 Example of FOR PORTION OF BUSINESS_TIME semantics: Case 2

```
-- assume table TRYTEMPORAL_B has the following data in POLICY, P_FROM, P_TO:
-- POLICY    P_FROM      P_TO
--      1  2001-01-01  2008-01-01
--      2  2002-01-01  2006-01-01
--      3  2003-01-01  2004-01-01
```

```
--      4  2005-01-01  2007-01-01
DELETE FROM TRYTEMPORAL_B
       FOR PORTION OF BUSINESS_TIME FROM DATE'2003-01-01'
                                    TO   DATE'2004-04-04'
WHERE POLICY > 2;
SELECT POLICY, P_FROM, P_TO
FROM TRYTEMPORAL_B;
-- result: POLICY    P_FROM      P_TO
--              1  2001-01-01  2008-01-01
--              2  2002-01-01  2006-01-01
--              4  2005-01-01  2007-01-01
```

Example 7-17 shows the *partially contained* case. We observe that in addition to the row being deleted, a new row has been inserted with the same values except for the begin column value which is now equals to the *value2* provided in the FOR PORTION OF BUSINESS_TIME clause.

Example 7-17 Example of FOR PORTION OF BUSINESS_TIME semantics: Case 3

```
-- assume table TRYTEMPORAL_B has the following data in POLICY, P_FROM, P_TO:
-- POLICY    P_FROM      P_TO
--      1  2001-01-01  2008-01-01
--      2  2002-01-01  2006-01-01
--      3  2003-01-01  2004-01-01
--      4  2005-01-01  2007-01-01
DELETE FROM TRYTEMPORAL_B
       FOR PORTION OF BUSINESS_TIME FROM DATE'2002-01-01'
                                    TO   DATE'2002-02-02'
WHERE POLICY > 1;
SELECT POLICY, P_FROM, P_TO
FROM TRYTEMPORAL_B;
-- result: POLICY    P_FROM      P_TO
--              1  2001-01-01  2008-01-01
--              3  2003-01-01  2004-01-01
--              4  2005-01-01  2007-01-01
--              2  2002-02-02  2006-01-01
```

Finally, Example 7-18 shows the case where a row *fully overlaps* the specified period. We observe that in addition to the row being deleted, two new rows are inserted with the same values except for the begin column value in one new row and the end column value in the other new row.

Example 7-18 Example of FOR PORTION OF BUSINESS_TIME semantics: Case 4

```
-- assume table TRYTEMPORAL_B has the following data in POLICY, P_FROM, P_TO:
-- POLICY    P_FROM      P_TO
--      1  2001-01-01  2008-01-01
--      2  2002-01-01  2006-01-01
--      3  2003-01-01  2004-01-01
--      4  2005-01-01  2007-01-01
DELETE FROM TRYTEMPORAL_B
       FOR PORTION OF BUSINESS_TIME FROM DATE'2003-03-03'
                                    TO   DATE'2004-04-04'
WHERE POLICY = 1;
SELECT POLICY, P_FROM, P_TO
FROM TRYTEMPORAL_B;
```

Archived

```
-- result: POLICY    P_FROM      P_TO
--               2  2002-01-01  2006-01-01
--               3  2003-01-01  2004-01-01
--               4  2005-01-01  2007-01-01
--               1  2001-01-01  2003-03-03
--               1  2004-04-04  2008-01-01
```

Note that SQLERRD3 reflects only the number of rows that are updated and does not include the count of rows that are inserted.

Note that triggers that are defined to be activated on an INSERT are now activated whenever a split of an original row causes a new row to be inserted into the table.

Currently, there is no support for ALTER INDEX to specify ADD BUSINESS_TIME WITHOUT OVERLAPS. And there is no support for SELECT FROM UPDATE and SELECT FROM DELETE when the UPDATE or DELETE statement has the FOR PORTION OF clause specified.

When you use both the SYSTEM_TIME period and the BUSINESS_TIME period in the same table, the table is referred to as bi-temporal table.

7.2 Access plan stability and instance-based statement hints

DB2 10 for z/OS delivers a framework that is geared towards the support of the existing and future features that pertain to query access path. In DB2 10, the features that fall into this category are as follows:

- Access plan stability
- Instance-based statement hints

In this section, we provide a high-level overview of this framework and then describe how it is used by the access plan stability and instance-based statement hints features.

The features that pertain to the query access path are as follows:

- Access path repository
- EXPLAIN PACKAGE statement

  EXPLAIN PACKAGE statement extracts EXPLAIN information for all static SQL statements in a package and populates this information in the PLAN_TABLE that is owned by the statement invoker. This statement extracts only information for packages that are created on DB2 9 or higher.

  This statement is useful if you did not bind with EXPLAIN(YES) or if PLAN_TABLE entries are lost.
- New DB2 commands
  - BIND QUERY
  - FREE QUERY
- New catalog table

  SYSIBM.SYSPACKCOPY holds metadata for the old package copies (that is, PREVIOUS and ORIGINAL). It contains all the columns of SYSIBM.SYSPACKAGE. Column COPYID has a value of 1 for PREVIOUS copy and 2 for ORIGINAL copy.

Archive

- New BIND options
  - EXPLAIN(ONLY)
  - SQLERROR(CHECK)

7.2.1 Access path repository

The access path repository holds important query metadata (such as query text), query access paths and other information (such as optimization options). The repository supports versioning by maintaining copies of access paths and associated options for each query. Note that we use the term *copy* instead of *version* to avoid any confusion with package versions that are also supported by DB2.

Figure 7-1 shows the contents of the access path repository.

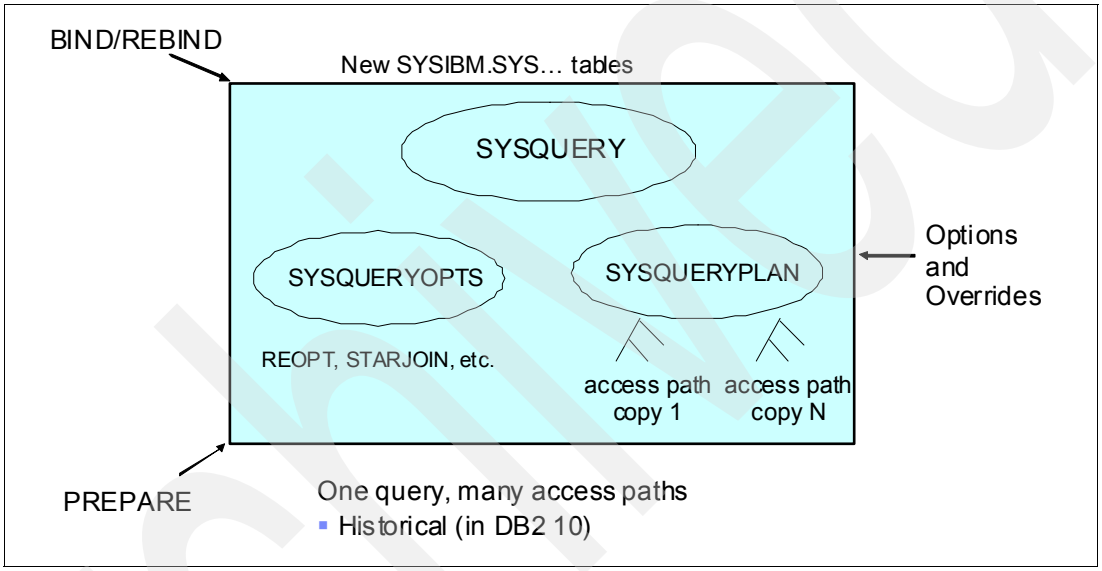


Figure 7-1 Access path repository

The repository comprises of the following catalog tables in the DSNDB06.SYSTSQRY catalog table space:

- SYSIBM.SYSQUERY is the central table of the access path repository. It holds one row for each static or dynamic SQL query that is being stabilized.

  For static and dynamic queries, much of the information stored in this table is common to all copies.

- SYSIBM.SYSQUERYPLAN holds access path descriptions (that is, EXPLAIN records) for each query represented in SYSQUERY. SYSQUERYPLAN contains a set of columns that are derived from the existing PLAN_TABLE schema. In the presence of versioning, a query in SYSQUERY can have more than one access path for it.
- SYSIBM.SYSQUERYOPTS holds miscellaneous information about each copy of a query in SYSQUERY.

The repository is intended to be populated with access paths of SQL queries when access path related features are used (such as access plan stability or system-level access path hints). Currently it is populated for system-level access path hints only.

7.2.2 BIND QUERY and FREE QUERY DB2 commands

You can manage the access path repository with the BIND QUERY and FREE QUERY commands.

The BIND QUERY (DSN) subcommand

The BIND QUERY subcommand reads the statement text, default schema, and a set of bind options from every row of DSN_USERQUERY_TABLE, and system-level access path hint information from correlated PLAN_TABLE rows. BIND QUERY then inserts the pertinent data into the SYSIBM.SYSQUERY, SYSIBM.SYSQUERYPLAN, and SYSIBM.SYSQUERYOPTS catalog tables.

Figure 7-2 shows the BIND QUERY syntax.

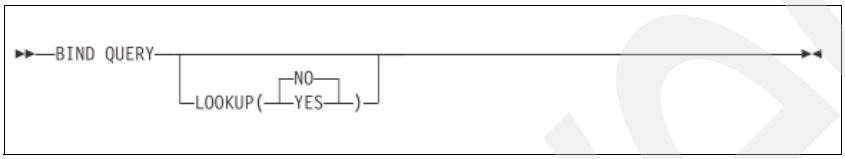


Figure 7-2 BI ND QUERY syntax

When the LOOKUP option is not specified or LOOKUP(NO) is specified, DB2 reads the information from the DSN_USERQUERY_TABLE and inserts the data into catalog tables SYSIBM.SYSQUERY, SYSIBM.SYSQUERYPLAN, and SYSIBM.SYSQUERYOPTS.

When LOOKUP(YES) is specified, no inserts occur and only the lookup is performed. The result is reported through messages. More specifically, DB2 reads the information from the DSN_USERQUERY_TABLE and looks for the matching rows in the catalog tables SYSIBM.SYSQUERY and SYSIBM.SYSQUERYPLAN.

For each row in the DSN_USERQUERY_TABLE that has a valid matching row in the SYSIBM.SYSQUERY table or the SYSIBM.SYSQUERYPLAN table, DB2 issues message DSNT280I. If no valid matching rows are identified in the SYSIBM.SYSQUERY table or the SYSIBM.SYSQUERYPLAN table, DB2 issues message DSNT281I.

If no rows from the DSN_USERQUERY_TABLE table have matching rows in the SYSIBM.SYSQUERY table or the SYSIBM.SYSQUERYPLAN table during BIND QUERY processing, DB2 issues message DSNT291I. For rows from the DSN_USERQUERY_TABLE table that have matching rows in the SYSIBM.SYSQUERYPLAN table during BIND QUERY processing, DB2 issues message DSNT290I.

The FREE QUERY subcommand

The FREE QUERY subcommand removes one or more queries from the access path repository. Use this command to purge one or more queries from the access path repository. If any of the specified queries are resident in the dynamic SQL cache, they are purged from the cache.

If you free multiple queries with this subcommand, each successful free is committed before the next query is freed. If an error occurs on a certain query, FREE QUERY terminates for that query and continues with the next query to be processed. Figure 7-3 shows the syntax for FREE QUERY.

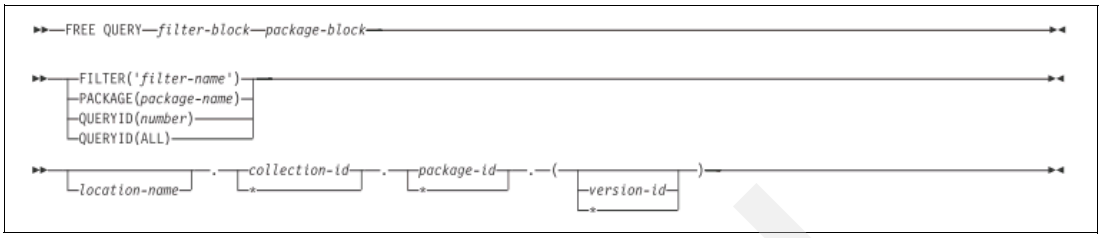


Figure 7-3 FREE QUERY syntax

The following options are available:

- FILTER('*filter-name*')

  Specifies the queries that are to be removed from the access path repository. *filter-name* is a value in the USERFILTER column of the SYSIBM.SYSQUERYcatalog table. During FREE QUERY processing, all the rows in the SYSIBM.SYSQUERY table that have the USERFILTER value filter-name are removed. Deletions from the SYSIBM.SYSQUERY table are cascaded to the SYSIBM.SYSQUERYPLAN table or the SYSIBM.SYSQUERYOPTS table.

- PACKAGE(package-name)

  The name of the package from which the queries are to be freed.

- QUERYID(*number*)

  Frees an entry in the SYSIBM.SYSQUERY table that has the same QUERYID value as *number*, and the corresponding entries in the SYSIBM.SYSQUERYPLAN table or the SYSIBM.SYSQUERYOPTS table.

- QUERYID(ALL)

  Frees all the entries from the SYSIBM.SYSQUERY table and the corresponding entries from the SYSIBM.SYSQUERYPLAN table or the SYSIBM.SYSQUERYOPTS table.

Example 7-19 shows an example to free all access paths for all queries.

Example 7-19 FREE QUERY Example 1

```
FREE QUERY QUERYID(ALL)
```

Example 7-20 shows an example to free queries for which the USERFILTER column of the SYSIBM.SYSQUERY catalog table contains SALESAPP.

Example 7-20 FREE QUERY Example 2

```
FREE QUERY FILTER('SALESAPP')
```

Example 7-21 shows an example to free all queries in package SALESPACK.

Example 7-21 FREE QUERY Example 3

```
FREE QUERY PACKAGE(SALESCOLL.SALESPACK)
```

7.2.3 Access plan stability

Query optimization in DB2 depends on many inputs, and even minor changes in the database environment can cause access paths to change. Reoptimization is not always the preferred solution and sometimes it can cause performance regression.

Archived

Static SQL queries are usually insulated from erratic changes in the access paths because DB2 allows precompiling or binding these queries such that an executable form is created and stored in the database. The executable form is subsequently reused for multiple submissions of that query. However, there are situations when static SQL queries are reoptimized and access paths regress.

This situation can occur after an explicit user action (using BIND/REPLACE or REBIND), but it can also be triggered automatically due to a change in the database (such as dropped indexes). REBIND after a release migration can also choose a different access path due to changes in the way the access path is chosen.

Dynamic SQL queries are more prone to wild swings in access paths because DB2 does not retain access paths of dynamic queries on disk. Dynamic SQL access paths are typically just cached in memory for performance reasons, and after an access path is purged from the cache, a new one needs to be generated (PREPARE). There is no certainty that the same access path is generated again, or that the new access path would better.

When access paths change due to reoptimization (REBIND or PREPARE) and a regression is detected, it is often difficult to revert to a prior access path. For the known critical transactions, users have had techniques like access path hints to revert to old access paths; however the procedure is not simple and identifying the source of such regressions can take time.

Access plan stability is meant to provide a fast and relative inexpensive way of reverting to a previous “good” access path improving availability and allowing time for problem determination.

7.2.4 DB2 9 access plan stability support

Here, we provide the background information for access plan stability support in DB2 9 first, because that support was introduced after the general availability using new function APAR.

You might want to REBIND plans or packages under DB2 9 to exploit DB2 9 virtual storage constraint relief for static SQL. Alternatively, you might want to enable SPROCs for fast column processing. Plans or packages bound prior to DB2 9 have SPROCs disabled in DB2 9, and they need to be enabled again. You can rebuild them dynamically at execution time, but the CPU performance impact is 0-10%. Alternatively, you can REBIND the plans and packages to enable SPROCs.

DB2 9 APAR PK52523 (enabling APAR available in DB2 9 CM) with PK52522 (as pre-conditioning APAR to be applied to V8 (NFM) and DB2 9 CM) provide the initial support for access path stability at the package level (not plan) as follows:

- It applies only to static SQL and supports regular packages, trigger packages, non-native SQL procedures, and external procedures.
- It allows you to retain package information (access paths, authorizations, query text, metadata, dependencies, and other information) during REBIND.
- It saves old copies of packages in the catalog tables and directory, which causes some growth in SPT01 limited to a maximum of 64 GB.
- Although over time, a package can have multiple copies, only one of these copies is the ACTIVE or CURRENT copy.

For considerations on performance using this function in DB2 9, refer to *DB2 9 for z/OS Performance Topics*, SG24-7473.

DB2 9 introduces the DSNZPARM PLANMGMT, and PLANMGMT and SWITCH options for REBIND PACKAGE and REBIND TRIGGER PACKAGE commands. The DSNZPARM PLANMGMT is not on any installation panel.

PLANMGMT in macro DSN6SPRM

PLANMGMT affects the behavior of the new options for REBIND PACKAGE and REBIND TRIGGER PACKAGE commands. PLANMGMT can have a value of OFF, BASIC, or EXTENDED. The default value of OFF results in little change in the pre-DB2 9 REBIND behavior. REBIND begins by purging all on-disk and in-memory pieces of the current package copy from the catalog tables and directory. It then compiles each static query in the package and creates new records that represent the result of the compilation. At the end of the rebind, many of the records that pertain to the package (including important sections) are lost. However, this setting might or might not purge any of the original copies.

A value of BASIC or EXTENDED for PLANMGMT results in slightly altered behavior. REBIND automatically saves copies of all pertinent records from catalog tables and directory that pertain to the existing package. In the event of a performance regression, these records can be restored and you can fall back to the saved copies.

The BASIC and EXTENDED values primarily differ in the number of copies of a package that are retained:

- When you specify a value of BASIC, DB2 retains up to two copies of a package. These copies are the currently active copy and one previous copy. With each invocation of REBIND ... PLANMGMT(BASIC), any previous copy is discarded, and the current copy becomes the previous copy. The incoming copy then becomes the current copy.
- When you specify a value of EXTENDED, DB2 retains up to three copies of a package. These copies are the currently active copy, a previous copy, and an original copy. The original copy is the one that existed when the REBIND ... PLANMGMT(EXTENDED) command was first run. This copy is created only once. It is never overwritten.

Bound packages are contained in the DB2 catalog table SPT01. Keeping one or two extra copies of packages increases accordingly the space need for SPT01, which is often already of large size. Compression for SPT01 is enabled through APAR PK80375.

REBIND (TRIGGER) PACKAGE ... PLANMGMT option

Figure 7-4 shows the new PLANMGMT option for REBIND (TRIGGER) PACKAGE command.

Archive

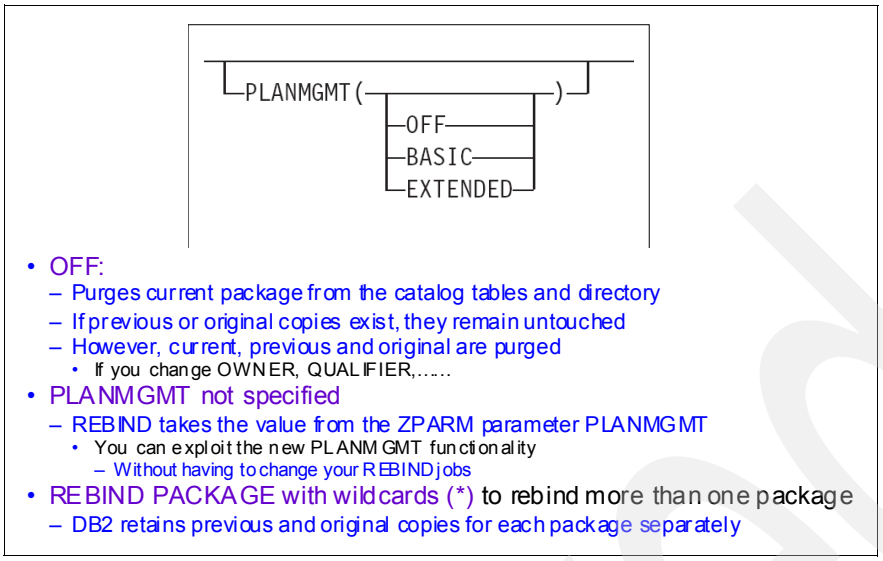


Figure 7-4 REBIND (TRIGGER) PACKAGE...PLANMGMT option

When you rebind a package with a value of BASIC or EXTENDED for PLANMGMT, most rebind options can be changed from the settings that existed before. However, the following options must stay the same:

- OWNER
- QUALIFIER
- DBPROTOCOL
- ENABLE
- DISABLE
- PATH
- PATHDEFAULT
- IMMEDWRITE

If you change any of these options when PLANMGMT is set to a value of BASIC or EXTENDED, DB2 issues an error.

Figure 7-5 shows the effect of PLANMGMT(BASIC) option.

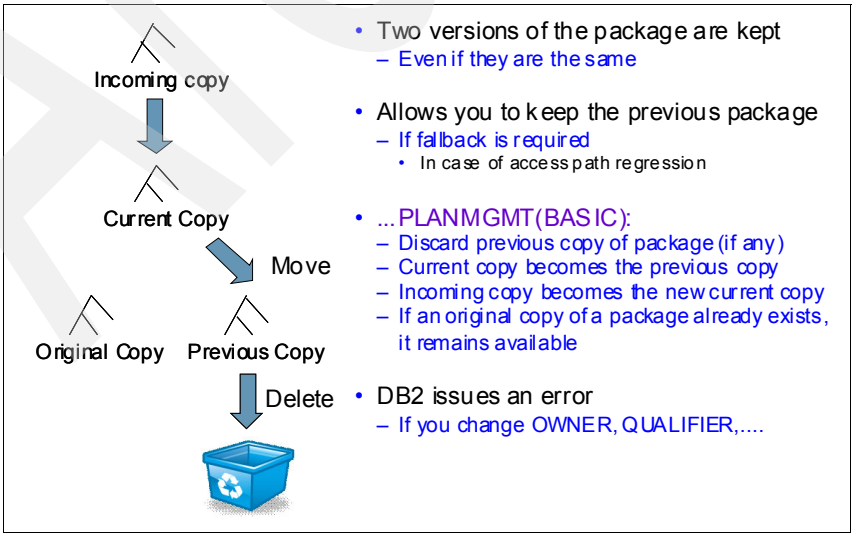


Figure 7-5 REBIND (TRIGGER) PACKAGE...PLANMGMT(BASIC)

When migrating from DB2 V8 to DB2 9 or 10 in a disk-constrained environment, use PLANMGMT(BASIC) for the first REBIND after migration to DB2 9. For subsequent REBINDs, use PLANMGMT(OFF). Using this statement is important. This mechanism preserves V8 packages but does not preserve any older DB2 9 packages. Using PLANMGMT(BASIC) multiple times destroys older V8 packages

Figure 7-6 shows the effect of PLANMGMT(EXTENDED) option.

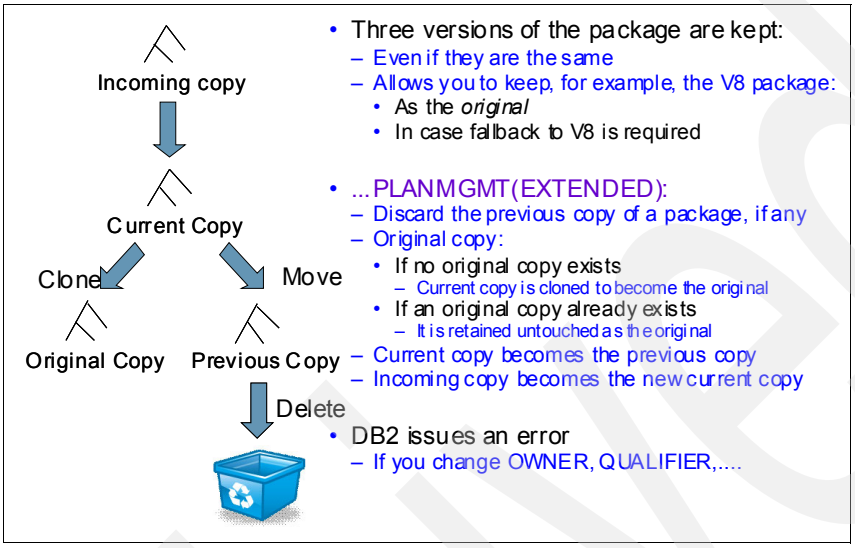


Figure 7-6 REBIND (TRIGGER) PACKAGE...PLANMGMT(EXTENDED)

When migrating from DB2 V8 to DB2 9 or DB2 10 with environments without DASD constraints, use PLANMGMT(EXTENDED) for the first REBIND on migration to DB2 9. Also use PLANMGMT(EXTENDED) for any subsequent REBINDs on DB2 9. This mechanism preserves V8 packages as ORIGINAL and older DB2 9 packages as PREVIOUS.

REBIND (TRIGGER) PACKAGE ... SWITCH option

After having done the REBIND, how do you switch to the PREVIOUS or ORIGINAL package? For this, DB2 9 provides the SWITCH option in REBIND PACKAGE and REBIND TRIGGER PACKAGE commands.

Archive

Figure 7-7 shows the effect of SWITCH option when PREVIOUS is specified.

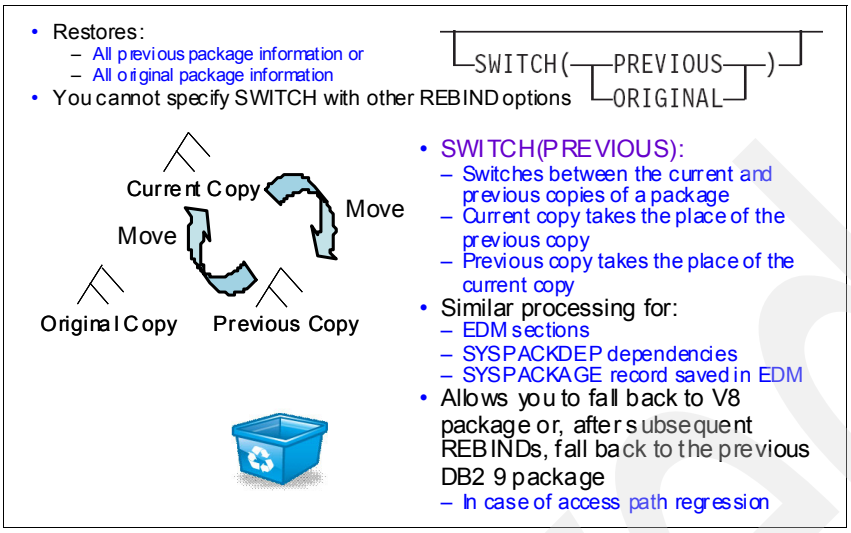


Figure 7-7 REBIND (TRIGGER) PACKAGE...SWITCH (1 of 2)

Figure 7-8 shows the effect of SWITCH option when ORIGINAL is specified.

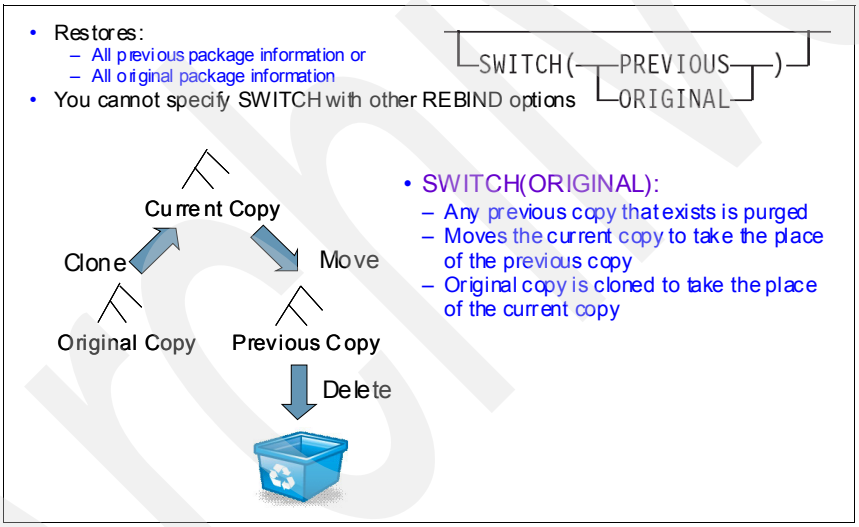


Figure 7-8 REBIND (TRIGGER) PACKAGE...SWITCH (2 of 2)

FREE PACKAGE PLANMGMTSCOPE option

DB2 9 introduced the new PACKAGE PLANMGMTSCOPE option to free inactive copies of packages manually to help reclaim disk space.

Archive

Figure 7-9 shows the effect of this option.

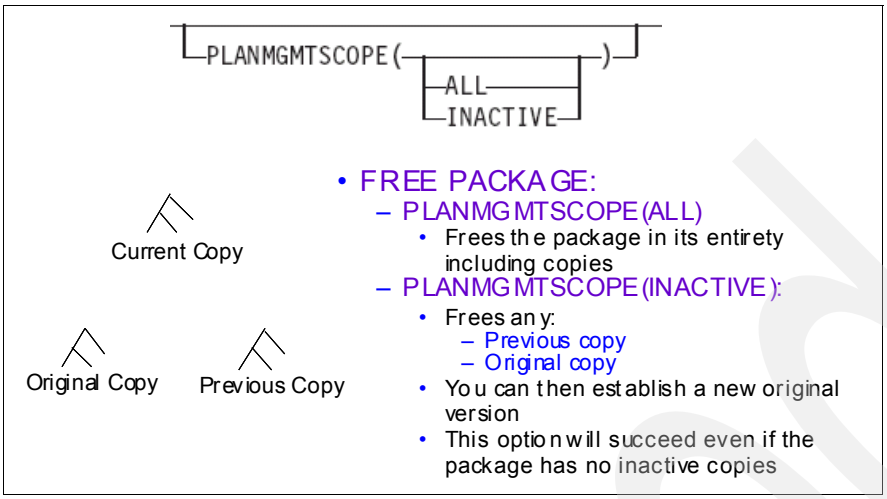


Figure 7-9 FREE PACKAGE command with PLANMGMTSCOPE option

7.2.5 DB2 10 access plan stability support

DB2 10 extends the access plan stability support by addressing the following issues:

- Prior to DB2 10, the SYSPACKAGE catalog table contains only the information about the current version. No information is available for previous or original copies. Users must use SWITCH(PREVIOUS) or SWITCH(ORIGINAL) to reveal the SYSPACKAGE data for that version.

  DB2 10 introduces the SYSPACKCOPY catalog table that we describe in 7.2, “Access plan stability and instance-based statement hints” on page 220.

- Prior to DB2 10, when redundant package copies are saved (that is, when the access path does not change), SPT01 space is wasted.

  DB2 10 for z/OS introduces APRETAINDUP REBIND option that determines whether or not DB2 retains an old package when access paths of the old copy are identical to the incoming (that is, newer) copy. This option applies only when PLANMGMT(BASIC) or PLANMGMT(EXTENDED) is in effect. APRETAINDUP(YES) is the default, which causes DB2 to always retain older package copies. This use is compatible with DB2 9. However, if APRETAINDUP(NO) is used, DB2 only retains the newer copy and this provides savings in disk space.

- DB2 10 now supports the following REBIND PACKAGE options for the native SQL stored procedure packages:
  - PLANMGMT
  - SWITCH

SPT01 compression note: SPT01 compression is not available in DB2 10 (because SPT01 uses LOBs).

DSNZPARM parameters

PLANMGMT is a DSNZPARM introduced by DB2 9 and can be set to OFF (default), BASIC, or EXTENDED. In DB2 10 the default is EXTENDED. DB2 10 introduces a new DSNZPARM PLANMGMTSCOPE and currently the only value permitted is STATIC.

Support note: At this time, DB2 10 enhancements do not address access path stability of dynamic SQL.

7.2.6 Instance-based statement hints

Instance-based statement hints (also referred to as *system-level access path hints*) provide a new mechanism for matching hints to a given query. In prior releases, QUERYNO linked queries to their associated hints, which might be error prone because QUERYNO might potentially change, which requires a change to applications for dynamic SQL.

With DB2 10, the mechanism uses query text to match with corresponding hints (similar to how statement matching is performed for the dynamic statement caching). Using this mechanism, the hints are enforced based on the statement text for the entire DB2 subsystem (hence the name “instance-based” or “system-level”).

You can use the new hints is as follows:

- Populate a user table DSN_USERQUERY_TABLE with the query text
- Populate PLAN_TABLE with corresponding hints
- Run new command BIND QUERY, which will integrate the hints into the DSN_USERQUERY_TABLE repository
- Use FREE QUERY to remove the hint

The infrastructure also allows you to have statement-level BIND options, which is extremely useful if you need statement-level granularity (as opposed to package level BIND option or a subsystem level DSNZPARMs). The way to use statement-level BIND options is similar to using the instance-based hints, by using the DSN_USERQUERY_TABLE to associate a given SQL statement with particular BIND options.

APAR PM24937 provides various fixes for optimization hints.

7.3 Addition of extended indicator variables

There are frequent situations where an application needs to insert or update data only for a subset of columns for a given table. For example, updating customer information might require updates to several different combinations of columns, depending on the situation. Perhaps, the address information needs to change, or a phone number, or some contact information, or some combination of those. Even the address change might require changing one or several of the columns (for example, street name, city, zip code, apartment number, and other address information).

The application developers are faced with a dilemma of how to provide an application that could handle all possible insert or update requests when it is not known which columns are being inserted to or updated until application execution time. Although there are several approaches to handle this issue, each of the approaches can have significant drawbacks.

For example, one way to handle this issue is to construct at execution time, for each user request, a dynamic SQL statement that contains the needed combination of columns. However, this method can impact overall system performance negatively if the dynamic statement cache is filled with these “custom” queries.

Another method is to code SQL statements, ahead of time, for all expected combinations of columns (one statement for each combination), which can become tedious quickly. You would need to have a statement for each possible supported combination for the INSERT, UPDATE, and MERGE statements. This method can also be difficult to maintain, because changes in requirements or other enhancements can result is new combinations of columns.

Yet another method is to code one SQL statement that handles all expected combinations of columns. With this method, an application would have to have logic to determine the values for all columns. However, it might not be always possible for the application to determine the correct values to use for columns whose values are not provided.

DB2 10 for z/OS addresses this difficulty of supporting changes to only a subset of columns in a table by introducing the *extended indicator variable* support.

Starting in DB2 10 new-function mode, you can use special values in the indicator variable to indicate to DB2 that the value for the associated host variable is not supplied and to specify how DB2 should handle the missing value. You can specify the extended indicator variable value of default (-5) to indicate that the target column of the host variable is to be set to its defined DEFAULT value, or you can specify the extended indicator variable value of unassigned (-7) to indicate that the host variable's value is UNASSIGNED and its target column is to be treated as though it had not appeared in the statement.

Given such indicator variable values, there is no need for the application to re-send a column's current value, or to know a column's DEFAULT value.

With the introduction of this support, the indicator variables can now be classified as follows:

- Indicator variable

  A variable used with a host variable to represent the SQL null value in an application program. An indicator variable with a negative value indicates that the host variable represents the null value. An indicator variable with a value of 0 or a positive value designates a host variable that contains a specified value instead of the null value.

- Extended indicator variable

  A variable used with a host variable to represent the SQL null value, the default value, or the unassigned value in an application program. When the extended indicator variable is used as an input variable, a value of -5 indicates that the associated host variable represents the default value for the associated column, a value of -7 indicates that the host variable is unassigned, and a value of -1, -2, -3, -4, or -6 indicates that the host variable represents the null value.

  An extended indicator variable with a value of 0 or a positive value designates a host variable that contains a specified value instead of the null value. When the extended indicator variable is used as an output variable, the values have the same meanings as for an indicator variable.

To clarify, the indicator variable support applies to parameter markers as well, because parameter markers specify host variables for a dynamically prepared SQL statement.

The support for the extended indicator variables can be enabled at a package level, by using the EXTENDEDINDICATOR option on the BIND PACKAGE (or REBIND PACKAGE) command. You can also enable extended indicator variables on a statement level for dynamic SQL by using the WITH EXTENDED INDICATORS attribute on the PREPARE statement.

When extended indicator variables are enabled for a given statement (either directly using the PREPARE attribute when preparing the statement or by binding the package containing the

statement with EXTENDEDINDICATOR(YES)), you can set the indicator variable to a special value (default or unassigned) instead of providing a value in the associated host variable.

Note, however, that because no host variable value is provided to DB2 in this case, the use of the special extended indicator variable values is rather limited. This limitation is because there are not that many cases where not having the data can be tolerated (for example, what would it mean if an unassigned variable appeared on the predicate).

Current restriction is that the special extended indicator variable values can be specified only for host variables that appear in:

- The set assignment list of an UPDATE operation in UPDATE or MERGE statements
- The values list of an INSERT operation in INSERT or MERGE statements
- The select list of an INSERT statement in the FROM clause of the SELECT statement
- The source-table parameter of a MERGE statement

Furthermore, the host variable cannot be part of an expression other than an explicit cast (if the target column is not nullable, the explicit cast can be only to the same data type as the target column. Otherwise, an error is issued (SQLCODE -365, SQLSTATE 22539)). With multi-row insert, the extended indicator variable values of default or unassigned can be used inside the indicator array.

Because the extended indicator variable support is intended for allowing applications to skip sending data for any of the host variables specified on the statement, the support is applicable to input host variables only (that is, on output, DB2 does not set indicator variables to any of the special values).

Table 7-1 lists the valid values that could be specified for an indicator variable.

Table 7-1 Values for extended indicator variables

Value	Description
0 (zero) or a positive integer	Specifies a non-null value. A 0 or a positive value specifies that the first host identifier provides the value of this host variable reference.
-1, -2, -3, -4, or -6	Specifies a null value.
-5	If extended indicator variables are not enabled, a value of -5 specifies the null value. If extended indicator variables are enabled, a value of -5 specifies that the DEFAULT value is to be used for the target column for this host variable.
-7	If extended indicator variables are not enabled, a value of -7 specifies the null value. If extended indicator variables are enabled, a value of -7 specifies that the UNASSIGNED value is to be used for the target column for this host variable, which means that the target column is to be treated as though it was not specified in the statement.

For the INSERT statement, setting the extended indicator variable to -5 or -7 leads to the same result, because the semantics of the INSERT is to insert a default value for any column that is missing an input value (that is, unassigned).

For UPDATE or MERGE UPDATE, setting the extended indicator variable to -5 leads to the column being updated to the default value. In addition, setting the extended indicator variable to -7 leads to the update of the column not being applied.

Example 7-22 shows the extended indicator variable usage. We first create a simple table that contains columns with DEFAULT values defined.

Example 7-22 Extended indicator variable example: Table declaration

```
CREATE TABLE TRYINDVAR(
 NAME          VARCHAR(50)
,COUNTRY       VARCHAR(30) DEFAULT 'Earth'
,CITY          VARCHAR(30) DEFAULT 'Unknown'
,ZIP           CHAR(5)     DEFAULT '00000'
);
```

Next, we use a C application to insert some data into the table using the special extended indicator variable values. Example 7-23 shows the source code for the application.

Example 7-23 Extended indicator variable example: C application for INSERT

```
main()
{
 EXEC SQL INCLUDE SQLCA;
 EXEC SQL BEGIN DECLARE SECTION;
   struct {
    short hvi_name;
    short hvi_city;
    short hvi_country;
    short hvi_zip;
   } hv_indicators;

   char hv_name??(50??);
   char hv_country??(30??);
   char hv_city??(30??);
   char hv_zip??(5??);
 EXEC SQL END DECLARE SECTION;

 memset(&hv_indicators, 0, sizeof(hv_indicators));
 strcpy(hv_name,    "Josef");
 strcpy(hv_country, "Switzerland");
 hv_indicators.hvi_city = -5;           /* use DEFAULT    */
 hv_indicators.hvi_zip  = -7;           /* use UNASSIGNED */
 EXEC SQL
 INSERT INTO TRYINDVAR VALUES(
   :hv_name:hvi_name
  ,:hv_country:hvi_country
  ,:hv_city:hvi_city
  ,:hv_zip:hvi_zip
 );
}
```

We precompile, compile, and link-edit the program, then we bind the DBRM to a package and use the EXTENDEDINDICATOR(YES) option to enable the extended indicator variables (e.g. BIND PACKAGE(COLL0001) MEMBER(PACK0001) EXTENDEDINDICATOR(YES)). Then we bind the package to a plan and run the program. After the program finishes, we query the table to see what was inserted and observe the following inserted row:

```
('Josef', 'Switzerland', 'Unknown', '00000')
```

Archived

Next, we use C application to perform an update on our table using the special extended indicator variable values. The source code for the application is given in Example 7-24.

Example 7-24 Extended indicator variable example: C application for UPDATE

```
main()
{
 EXEC SQL INCLUDE SQLCA;
 EXEC SQL BEGIN DECLARE SECTION;
   struct {
    short hvi_name;
    short hvi_city;
    short hvi_country;
    short hvi_zip;
   } hv_indicators;

   char hv_name??(50??);
   char hv_country??(30??);
   char hv_city??(30??);
   char hv_zip??(5??);
 EXEC SQL END DECLARE SECTION;

 memset(&hv_indicators, 0, sizeof(hv_indicators));
 strcpy(hv_name,    "Michael");
 strcpy(hv_country, "Australia");
 hv_indicators.hvi_name    = -7;         /* skip update    */
 hv_indicators.hvi_country = -5;         /* use DEFAULT    */
 hv_indicators.hvi_city    = -7;         /* skip update    */
 hv_indicators.hvi_zip     = -7;         /* skip update    */
 EXEC SQL
 UPDATE TRYINDVAR SET
  NAME    = :hv_name:hvi_name
 ,COUNTRY = :hv_country:hvi_country
 ,CITY    = :hv_city:hvi_city
 ,ZIP     = :hv_zip:hvi_zip
 ;
}
```

After building the program and binding the DBRM as in INSERT example, we query the table and observe the following row:

```
('Josef', 'Earth', 'Unknown', '00000').
```

Note that an update of all columns except for the COUNTRY was skipped and that the value in the COUNTRY column was set to the DEFAULT value ('Earth') rather than the value in the associated host variable ('Australia'). This allows you to have one update statement for any possible scenario, and you would control what gets updated by setting the extended indicator variables without having to worry about the context of the host variables for the columns that are going to be skipped or set to the default value.

As a quick demonstration, we disable the extended indicator variables for this package by rebinding with EXTENDEDINDICATOR(NO) option (e.g. REBIND PACKAGE(COLL0001) MEMBER(PACK0001) EXTENDEDINDICATOR(NO)). Then re-run the UPDATE application and observed that all column values were set to null. This is because when extended indicator variables are disabled, any negative value in the indicator variable indicates a null value.

Archived

7.4 New Universal Language Interface program (DSNULI)

When building applications with releases prior to DB2 10, the DB2 applications must be link-edited with one of the following Language Interface modules:

- DSNRLI for RRSAF and Stored Procedures
- DSNALI for Call Attachment Facility
- DSNELI for TSO Attachment Facility
- DSNCLI for CICS Attachment
- DFSLI00 for IMS Attachment

This link-editing poses some usability issues because application developers frequently need to run the same program logic from different environments, and doing so is not possible with link-editing a single Language Interface modules. For example, a TSO application program that you run in a DSN session must be link-edited with the TSO language interface program (DSNELI), so the same program cannot be run from CICS or even invoked outside of DSN command processor (for example, // EXEC PGM=pgmname).

DB2 10 for z/OS introduces a universal language interface module DSNULI that can be used instead of DSNRLI, DSNALI, DSNELI, or DSNCLI. DSNULI can be linked with RRS, CAF, TSO, and CICS applications (including stored procedures, which use RRS), in place of DSNRLI, DSNALI, DSNELI, or DSNCLI. All of the existing language interfaces are still available, so DSNULI is simply an alternative that can be linked with an application when you want to maintain a single library of application modules that can run in multiple environments.

Archived

Figure 7-10 shows the general structure of DSNULI and a program that uses it.

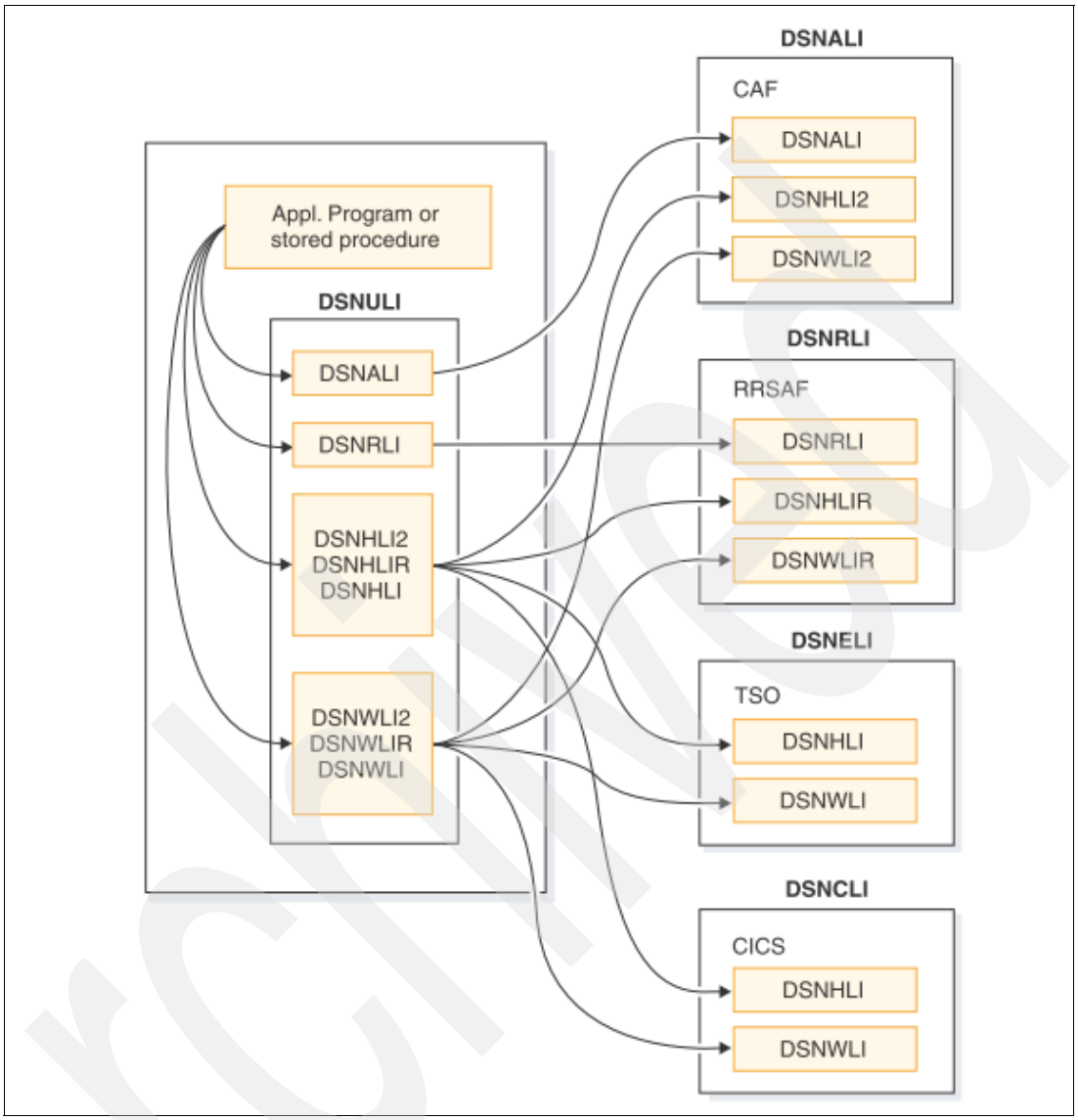


Figure 7-10 Application program or stored procedure linked with DSNULI

If you have 31-bit application programs that you want to use in multiple attachment environments, you can link those applications with DSNULI instead of DSNELI, DSNALI, DSNRLI, or DSNCLI to produce a single module that is executable in any of the following runtime environments:

- TSO
- CAF
- RRSAF
- CICS
- MVS batch

DSNULI does not support dynamic loading or IMS applications, and it does not support 24-bit applications. The way DSNULI works is that it determines the runtime environment from which the application has been invoked and then dynamically loads and branches to the appropriate language interface module (DSNELI, DSNALI, DSNRLI, or DSNCLI).

Naturally, this kind of execution time detection hinders performance, so there is a trade off between ease of application deployment and the application speed. There are nine entry points in DSNULI, which are associated with different attachment scenarios. DSNULI dynamically loads and branches to the appropriate language interface module based on the entry point used by the application (for example, when certain attachment facility entry point was used explicitly), or based on the current environment (for example, when application used default entry point).

To link-edit your application with DSNULI you can use a linkage editor control statement INCLUDE SYSLIB(DSNULI). Example 7-25 shows a sample JCL compile and link-edit step for a COBOL program.

Example 7-25 Link-editing DSNULI to your application

```
//AUDSQL    EXEC DSNHICOB,MEM=AUDSQL,
//         COND=(4,LT),
//  PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)',TEST)
//PC.DBRMLIB   DD DSN=DB2R4.DBRMLIB.DATA(AUDSQL),
//             DISP=SHR
//PC.SYSLIB    DD DSN=DB2R4.SOURCE.DATA,
//             DISP=SHR
//PC.SYSIN     DD DISP=SHR,DSN=DB2R4.SOURCE(AUDSQL)
//COB.SYSLIB   DD DISP=SHR,DSN=DB2R4.SOURCE
//COB.SYSPRINT DD SYSOUT=*
//LKED.SYSLMOD DD DISP=SHR,DSN=DB2R4.LOAD(AUDSQL)
//LKED.SYSLIB  DD DISP=SHR,DSN=DB2R4.LOAD
//             DD
//             DD DISP=SHR,DSN=CEE.SCEELKED
//LKED.SYSIN   DD *
 INCLUDE SYSLIB(DSNULI)
 NAME AUDSQL(R)
//*
```

We list here some additional considerations:

- Performance

  If maximum performance is the primary requirement, link-edit with DSNELI, DSNALI, DSNRLI, or DSNCLI rather than DSNULI.

  If maintaining a single copy of a load module is the primary requirement, link-edit with DSNULI.

- Implicit connections

  If CAF implicit connect functionality is required, link-edit your application with DSNALI instead of with DSNULI. DSNULI defaults to RRSAF implicit connections if an attachment environment has not been established upon entry to DSNHLI. Attachment environments are established by calling DSNRLI or DSNALI initially, or by running an SQL application under the TSO command processor or under CICS.

- Delete of stub

  DSNULI will not explicitly delete the loaded DSNELI, DSNALI, DSNRLI or DSNCLI. If an application cannot tolerate having these modules deleted only at task termination, use DSNELI, DSNALI, DSNRLI or DSNCLI instead of DSNULI.

- AMODE

  DSNULI is shipped with the linkage attributes AMODE(31), RMODE(ANY) and must be entered in AMODE(31).

Archived

7.5 Access to currently committed data

DB2 allows access only to data that is committed and consistent, with the exception of the ISOLATION (UR)[1] option for BIND. Normally, a transaction or unit-of-work is suspended when it tries to access a page or row if another unit-of-work has updated that page or row but has not yet committed the change. The first unit-of-work must wait until the blocking lock is either released or the transaction times out.

DB2 9 introduces the SKIP LOCKED DATA clause on the SELECT statement, which allows DB2 to skip any pages or rows that are *blocked* by locks held by other unit-of-works. Only committed and consistent data is still returned; however, not all the rows that qualify might be returned.

DB2 10 further enhances this type of support by allowing you to access the version of data that was last committed (that is, the version of data that existed before the “blocking” unit-of-work has changed the row but has not yet committed the change).

Figure 7-11 shows the bind option introduced in the BIND and REBIND PLAN or PACKAGE statements.

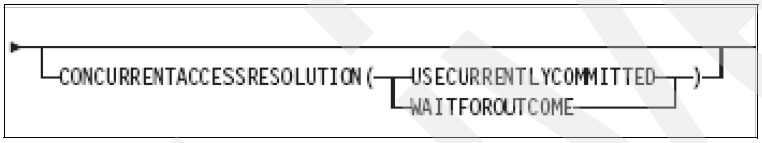


Figure 7-11 BIND option CONCURRENTACCESSRESOLUTION

A clause is introduced on the PREPARE statement, as shown in Figure 7-12.

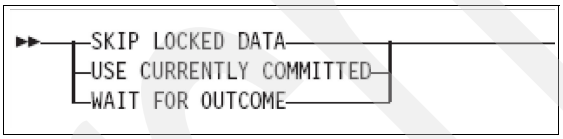


Figure 7-12 USE CURRENTLY COMMITTED clause

Finally, a bind option is included in the option list on the CREATE or ALTER PROCEDURE or FUNCTION statements, as shown in Figure 7-13.

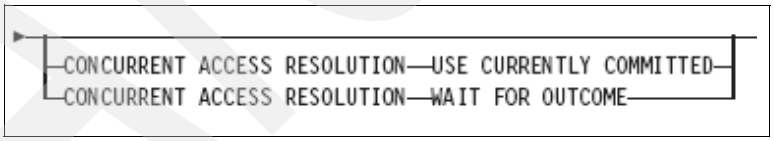


Figure 7-13 CREATE PROCEDURE option list option

You can request to access currently committed data both at the statement level or plan or package level. However, the statement level overrides the package or plan level.

Let us walk through how access to currently committed data works. A reader tries to read a row, but standard lock avoidance techniques fail because the row is possibly uncommitted. The reader then requests a conditional lock on the row. If the lock is not available and held by an inserter, DB2 skips the row. If the lock is not available and held by a deleter, DB2 returns the row.

[1] ISOLATION (UR) or uncommitted read option allows an application to read while acquiring fewer locks at the risk of reading uncommitted data.

This process works because deleted rows are only pseudo-deleted in DB2 10. There is still no loss in space reuse, because the space becomes available after the deleter commits. However, the entire row image is also logged physically as with earlier versions of DB2.

The existing DSNZPARM parameters of EVALUNC (process stage 1 predicates on uncommitted data) and SKIPUNCI (skip uncommitted inserts) must also be considered. When both EVALUNC and ConcurrentAccessResolution are requested, DB2 allows the evaluation of uncommitted data to avoid locks. If the row is qualified, then DB2 returns the row based on the ConcurrentAccessResolution option. However, the value of SKIPUNCI no longer applies. If USE CURRENTLY COMMITTED is specified, DB2 skips the uncommitted insert, and if WAIT FOR OUTCOME is specified, DB2 does not skip the uncommitted insert.

Access to currently committed data is available in new-function mode only and is applied only to universal table spaces. The following restrictions also apply:

- Only blocking INSERTs and DELETEs are supported. SELECTs must still wait for any blocking UPDATEs to commit.
- If the SELECT is in contention with an uncommitted INSERT, then access to currently committed data applies to ISOLATION(CS) and ISOLATION(RS) queries.
- If the SELECT is in contention with uncommitted DELETE, then access to currently committed data applies only to ISOLATION(CS) with CURRENTDATA(NO) queries.
- TRIGGERS are not supported.

Planning to access currently committed data does not ensure that your SQL will not wait for locks.

Access to currently committed data is not applicable as follows:

- For table, partition, or table space locks
- When LOCK TABLE IN EXCLUSIVE MODE is used
- When the lock holder is performing mass delete
- If the lock holder has escalated

Along with a number of new and revised messages, DB2 10 introduces two DB2 catalog columns to record when plans packages are bound with the CONCURRENTACCESSRESOLUTION option, as listed in Table 7-2.

Table 7-2 Catalog changes for CONCURRENTACCESSRESOLUTION

Column name	Data type	Description
SYSPACKAGE. CONCUR_ACC_RES	CHAR(1) NOT NULL	Indicates the CONCURRENTACCESSRESOLUTION option when the package was bound or rebound.
SYSPLAN. CONCUR_ACC_RES	CHAR(1) NOT NULL	Indicates the CONCURRENTACCESSRESOLUTION option when the plan was bound or rebound.

To monitor the use of access to currently committed data, DB2 introduces four counters to the Data Manager Statistics Block (DSNDQIST) of IFCID 002, as listed in Example A-2 on page 616. These counters are externalized in a new statistics block of the DB2 Performance Expert Statistics Long and Short reports, called *Use Currently Committed*.

The QISTRCCI field shows the number of rows skipped by read transactions using currently committed semantic encountering uncommitted inserts. The field QISTRCCD shows the number of rows accessed by read transactions using currently committed semantic encountering uncommitted deletes.

7.6 EXPLAIN MODE special register to explain dynamic SQL

Monitoring access paths is critical for good performance. For static SQL you can rebind or bind packages with EXPLAIN(YES) to populate the EXPLAIN tables. However explaining dynamic SQL statements such as JDBC and CLI is a little more complicated. Also, by using existing methods such as using EXPLAIN STMTCACHE, you cannot limit the explain to a single application or user.

DB2 10 makes collecting explain information for any dynamic SQL much easier by introducing a special register CURRENT EXPLAIN MODE, shown in Figure 7-14, to control the behavior of explain for dynamic SQL.

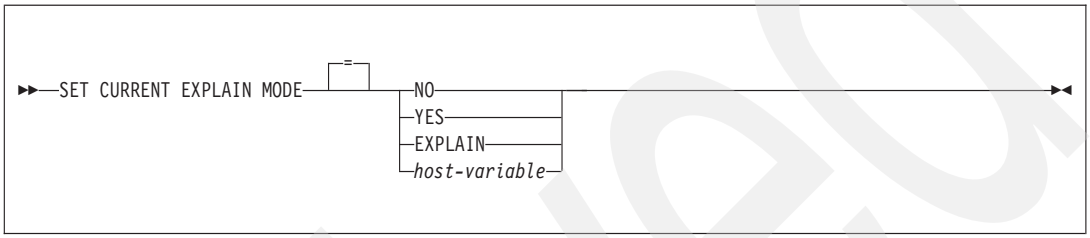


Figure 7-14 SET CURRENT EXPLAIN MODE

This register uses the following options:

- NO, the default, indicates that no explain information is captured during the execution of an explainable dynamic SQL statement.
- YES indicates that explain information is captured in the explain tables as eligible dynamic SQL is prepared and executed.
- EXPLAIN indicates that explain information is captured in the explain tables as eligible dynamic SQL is prepared. However dynamic statements except SET statements, are not executed and a SQLCODE +217 is returned indicating a dynamic SQL statement was not executed.

Note: For YES or EXPLAIN, prepared statements are not written to dynamic statement cache. The rationale is that the intent of this capability is the gathering of access path and run statistics, not caching and performance. We do not expect you to use this EXPLAIN MODE option in a production environment. The DSNZPARM CACHEDYN, therefore, is not required.

The following example uses the CURRENT EXPLAIN MODE special register in a host based language. Although it might not be useful, because you need to change the code. (The explain tables are populated as the SQL is executed.)

```
EXEC SQL DECLARE C1 CURSOR FOR PREP_STMT;
SOURCE_STMT = 'SELECT X, Y, Z FROM SCHEMA.TABLE1 WHERE X < Y ';
EXEC SQL SET CURRENT EXPLAIN MODE = YES;
EXEC SQL PREPARE PREP_STMT FROM SOURCE_STMT;
EXEC SQL OPEN C1;
```

You can, however, explain dynamic SQL in JCC without changing the code. JCC flows the CURRENT EXPLAIN MODE special register setting from the currentExplainMode connection property on behalf of the application. Both JDBC and SQLJ are supported, and the EXPLAIN tables are populated as the SQL is executed.

Archived

For ODBC/CLI applications, you can use the SET keyword to set DB2EXPLAIN in the DSNAOINI initialization file, which is a system-wide setting. In addition to the initialization keyword DB2EXPLAIN, you can set this attribute in DB2 for z/OS ODBC/CLI applications using SQL_ATTR_DB2EXPLAIN with the SQLSetConnectAttr() function at any time. SQLSetConnectAttr() can be called by the application to set attributes that govern aspects of connections and can be used to override settings in the ODBC/CLI initialization file. The SET CURRENT EXPLAIN MODE statement can be prepared and executed directly by the application program and can override any connection level settings.

For both z/OS and LUW ODBC/CLI drivers, CURRENT EXPLAIN MODE = EXPLAIN cannot be set through the DB2EXPLAIN keyword in the initialization file or through the SQLSetConnectAttr() function with SQL_ATTR_DB2EXPLAIN. If an ODBC/CLI application wants to use CURRENT EXPLAIN MODE = EXPLAIN, the SET statement must be prepared and executed directly by the application at the source level.

The <owner>.PLAN_TABLE and <owner>.DSN_STATEMENT_CACHE_TABLE tables must exist where the SQL executes. Otherwise, a SQLCODE -219 results. The <owner>.DSN_FUNCTION_TABLE and <owner>.DSN_STATEMNT_TABLE tables are optional and are populated if they exist. You can find sample DDL for these tables in SDSNSAMP(DSNTESC) and a description in *DB2 10 for z/OS SQL Reference,* SC19-2983.

Rows inserted into EXPLAIN tables by SQL EXPLAIN STMTCACHE and CURRENT EXPLAIN MODE have different COLLIDs. SQL EXPLAIN STMTCACHE uses 'DSNDYNAMICSQLCACHE' and CURRENT EXPLAIN MODE uses 'DSNEXPLAINMODEYES' or 'DSNEXPLAINMODEEXPLAIN'.

Some columns of DSN_STATEMENT_CACHE_TABLE are N/A. Normally, the statistics in the columns of DSN_STATEMENT_CACHE_TABLE are cumulative across executions of the same statement and across threads. If the value of COLLID is DSNEXPLAINMODEYES, certain columns in DSN_STATEMENT_CACHE_TABLE are for a single run of the statement only.

To correlate the information across the explain tables, first find the STMTID[2] for the relevant dynamic statement by searching the STMT_TEXT column of the DSN_STATEMENT_CACHE_TABLE.

For SQL EXPLAIN STMTCACHE;

- Use the following join predicates to join the DSN_STATEMENT_CACHE_TABLE to the PLAN_TABLE:

```
DSN_STATEMENT_CACHE_TABLE.STMTID = PLAN_TABLE.QUERYNO AND
DSN_STATEMENT_CACHE_TABLE.CACHED_TS = PLAN_TABLE.BIND_TIME
```

- Use the following join predicates to join the DSN_STATEMENT_CACHE_TABLE to the DSN_FUNCTION_TABLE or DSN_STATEMNT_TABLE:

```
DSN_STATEMENT_CACHE_TABLE.STMTID = <explain table>.QUERYNO AND
DSN_STATEMENT_CACHE_TABLE.CACHED_TS = <explain table>.EXPLAIN_TIME
```

For CURRENT EXPLAIN MODE, use the column DSN_STATEMENT_CACHE_TABLE.EXPLAIN_TS instead of DSN_STATEMENT_CACHE_TABLE.CACHED_TS in these predicates.

[2] STMTID support and IFCID 401 generation require REBIND in NFM.

Important: Although the CURRENT EXPLAIN MODE special register provides an easy way to explain dynamic SQL without having to change the application, we advise that you not use it indiscriminately. It can generate huge amounts of EXPLAIN data in the EXPLAIN tables, because potentially every prepare or open/execute that is performed is explained.

7.7 Save LASTUSED information for packages

In the past, it was difficult to identify old packages that are no longer executed and that are wasting space in the DB2 catalog. Space in the DB2 catalog, particularly SPT01, has become a concern for many in recent years, particularly as the SPT01 table spaces grows closer to the old 2 GB limit and especially when plan stability is used.

The common method in the past to identify obsolete packages which can safely be freed was to reconcile extracts from Accounting Summary reports or extracts from an accounting performance warehouse, with the DB2 catalog.

DB2 10 now helps make this process easier. DB2 maintains a new column, LASTUSED, in table space SYSIBM.SYSPACKAGE and SYSIBM.SYSPLAN. The column is defined as a date type column that is filled out when the package header is requested from EDM. The column is also maintained for triggers and stored procedures. After BIND (REPLACE) this column is reset to the default value (CURRENT DATE). This column can be updated in DB2 10 conversion mode.

Archived

8

XML

XML brings complete portability to your data. The requirements for a database management system (DBMS) have included support for XML data.

DB2 9 for z/OS introduced the support for XML data type through the use of its pureXML capabilities and hybrid database engine. With DB2 9, XML data previously stored in the traditional relational format, could be stored natively as XML. The introduction of the new data type has implied some changes in the administration processes and programming techniques.

DB2 9 for z/OS maintenance stream and DB2 10 for z/OS added functions and performance improvements. In the first sections of this chapter, we describe the DB2 9 later functions. Then, in the remaining sections, we provide details of the DB2 10 XML functions.

For information about XML and LOB streaming, see 13.11.3, “Streaming LOBs and XML between DDF and DBM1” on page 568,

This chapter contains the following topics:

- DB2 9 XML additional functions
- XML type modifier
- XML schema validation
- XML consistency checking with the CHECK DATA utility
- Support for multiple versions of XML documents
- Support for updating part of an XML document
- Support for binary XML
- Support for XML date and time
- XML in native SQL stored procedures and UDFs
- Support for DEFINE NO for LOBs and XML
- LOB and XML data streaming

© Copyright IBM Corp. 2010. All rights reserved.

8.1 DB2 9 XML additional functions

DB2 9 introduced many enhancements after the general availability (GA) of DB2 9 for z/OS using the maintenance stream both on the area of performance and additional functions.

II14426 is the information APAR that includes all the XML support delivery APARs.

To search for DB2 APARs that contain XML information, go to the following web address:

http://www.ibm.com/support/search.wss?rs=64&r=100&cs=utf-8&rankfile=0&cc=&coll=0&spc=&stc=&apar=include&q1=XML&sort=desc&tc=SSEPEK&ibm-search.x=18&ibm-search.y=11&dc=DB550+D100&dtm

8.1.1 The XMLTABLE function

DB2 9 includes support for the XMLTABLE function. The XMLTABLE SQL/XML function returns a table from the evaluation of XPath expressions. This function was implemented by APARs PK51571, PK51572, and PK51573. Additional APARs are PK57409, PK58914, PM05664, and PM11482. Figure 8-1 shows the XMLTABLE function syntax.

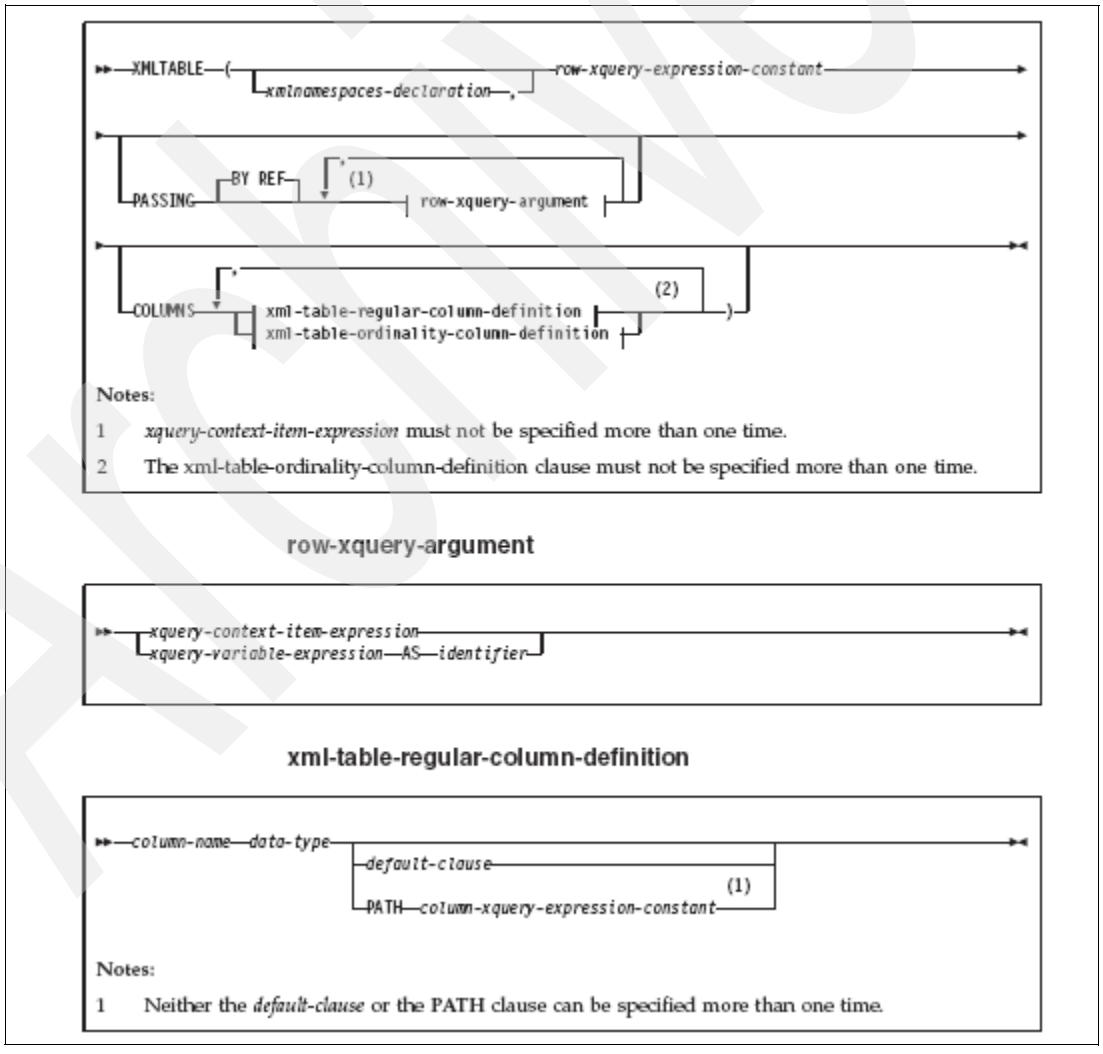


Figure 8-1 The XMLTABLE function syntax

Archive

An overview of the XMLTABLE function

XPath expressions normally return values as a sequence. However, XMLTABLE lets you execute an XPath expression and return values as a table. The table that is returned can contain columns of any SQL data type, including the XML data type.

You can pass variables to the *row* XPath expression that is specified in XMLTABLE. The result of the row XPath expression defines the portions of an XML document that you use to define a row of the returned table. You specify *column* XPath expressions in the COLUMNS clause of the XMLTABLE function to generate the column values of the resulting table. In the COLUMNS clause, you define characteristics of a column by specifying the column name, data type, and how the column value is generated. Alternatively, you can omit the COLUMNS clause and let DB2 generate a single, unnamed XML column.

You can include an XMLNAMESPACES function as the first argument of XMLTABLE to specify the XML namespaces for all XPath expressions in the XMLTABLE function. Namespace declarations in individual XPath expressions override the XMLNAMESPACES argument.

You can specify the contents of a result table column through a column XPath expression that you specify in the PATH clause of XMLTABLE. If you do not specify a PATH clause, DB2 uses the result table column name as the PATH argument. For example, if a result table column name is *@partNum*, and the input XML documents have an attribute named *partNum*, the result table column values are the values of the partNum attribute.

If the column XPath expression that defines a result table column returns an empty sequence, XMLTABLE returns a NULL value in the result table column. XMLTABLE lets you supply a default value instead of a NULL value. You do this by specifying a DEFAULT clause in the column definition.

If you want to generate a sequence number for each row that XMLTABLE generates, you can include a column definition with the FOR ORDINALITY clause. The FOR ORDINALITY clause causes XMLTABLE to generate a column with values that start at 1. If a single document generates more than one row, the sequence number is incremented by 1. For example, if an XML document generates three result table rows, the sequence numbers for those rows are 1, 2, and 3. If the column XPath expression that is specified in the PATH option of XMLTABLE returns a sequence of more than one item, the data type of the result table column must be XML.

Advantages of using the XMLTABLE function

In certain situations, XPath expression results are easier to process if they are in a table than if they are in a sequence. Returning a table instead of a sequence enables the following operations to be performed from within a SQL query context:

- Iteration over results of an XPath expression from within a SQL select

  As shown in Example 8-1, the SQL select iterates over the table that results from executing the XPath expression and customer information in the XMLTABLE function.

Example 8-1 Iteration over results using the XMLTABLE function

```
SELECT X.*
FROM CUSTOMER,
XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
'/customerinfo'
PASSING CUSTOMER.INFO
COLUMNS "CUSTNAME" VARCHAR(30) PATH 'name',
"addr/city" VARCHAR(30)) X
```

- Insertion of values from XML documents into tables

  This technique is a simple form of decomposition, where decomposition is the process of storing fragments of an XML document in columns of relational tables.

- Individual processing of items in a sequence

  If you return the items in a sequence as a single row, with each item in a separate column, it is easier to process the individual items.

- Sorting on values from an XML document

  In the query in Example 8-2, results are sorted by the customer names that are stored in XML documents in the INFO column of the CUSTOMER table.

Example 8-2 Sorting values from an XML document

```
SELECT X.*
FROM CUSTOMER,
XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
'//customerinfo'
PASSING CUSTOMER.INFO
COLUMNS "CUSTNAME" VARCHAR(30) PATH 'name',
"addr/city" VARCHAR(30)) X
ORDER BY X.CUSTNAME
```

- Storing of some XML values as relational and some values as XML

  You can use the XMLTABLE SQL/XML function to retrieve values from within stored XML documents. The retrieved values can then be inserted into a table.

  For example, the XML documents in Example 8-3 are stored in the sample CUSTOMER table.

Example 8-3 Stored XML documents

```
<customerinfo xmlns="http://posample.org" Cid="1001">
 <name>Kathy Smith</name>
  <addr country="Canada">
   <street>25 EastCreek</street>
   <city>Markham</city>
   <prov-state>Ontario</prov-state>
   <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
 <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo xmlns="http://posample.org" Cid="1003">
 <name>Robert Shoemaker</name>
  <addr country="Canada">
   <street>1596 Baseline</street>
   <city>Aurora</city>
   <prov-state>Ontario</prov-state>
   <pcode-zip>N8X-7F8</pcode-zip>
  </addr>
 <phone type="work">905-555-7358</phone>
 <phone type="home">416-555-2937</phone>
 <phone type="cell">905-555-8743</phone>
 <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

Insert values from these documents into a table with the following definition:

```
CREATE TABLE CUSTADDR (
  CUSTNAME VARCHAR(30),
  CUSTSTREET VARCHAR(30),
  CUSTCITY VARCHAR(30)
  CUSTSTATE VARCHAR(30),
  CUSTZIP VARCHAR(30))
```

The INSERT statement in Example 8-4, which uses XMLTABLE, populates CUSTADDR with values from the XML documents.

Example 8-4 Inserting values returned from XMLTABLE

```
INSERT INTO CUSTADDR
SELECT X.*
FROM CUSTOMER,
XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
'//customerinfo'
PASSING CUSTOMER.INFO
COLUMNS
"CUSTNAME" VARCHAR(30) PATH 'name',
"CUSTSTREET" VARCHAR(30) PATH 'addr/street',
"CUSTCITY" VARCHAR(30) PATH 'addr/city',
"CUSTSTATE" VARCHAR(30) PATH 'addr/prov-state',
"CUSTZIP" VARCHAR(30) PATH 'addr/pcode-zip'
) as X
```

After you execute the INSERT statement, the CUSTADDR table looks as shown in Table 8-1.

Table 8-1 Contents of CUSTADDR table after insert of a result table generated by XMLTABLE

CUSTNAME	CUSTSTREET	CUSTCITY	CUSTSTATE	CUSTZIP
Kathy Smith	25 East Creek	Markham	Ontario	N9C 3T6
Robert Shoemaker	1596 Baseline	Aurora	Ontario	N8X-7F8

If an XML document contains multiple occurrences of an element, you can use XMLTABLE to generate a row for each occurrence of the element. You can use a query, as shown in Example 8-5, to create a table in which every phone value is returned in a separate row.

Example 8-5 Returning one row for each occurrence of phone item as a string

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
'$cust/customerinfo/phone' PASSING C.INFO as "cust"
COLUMNS
"CUSTNAME" VARCHAR(30) PATH '../name',
"PHONETYPE" VARCHAR(30) PATH '@type',
"PHONENUM" VARCHAR(15) PATH '.') as X
WHERE CID=1001 OR CID=1003
```

Archived

Table 8-2 shows the result table of the SELECT statement.

Table 8-2 Result table of a query using XMLTABLE to retrieve multiple occurrences of an item

CUSTNAME	PHONETYPE	PHONENUM
Kathy Smith	work	905-555-7258
Robert Shoemaker	work	905-555-7358
Robert Shoemaker	home	416-555-2937
Robert Shoemaker	cell	905-555-8743
Robert Shoemaker	cottage	613-555-3278

The SELECT statement in Example 8-6 returns each phone element as an XML document, instead of a string value.

Example 8-6 Returning one row for each occurrence of phone item as an XML document

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
'$cust/customerinfo/phone' PASSING C.INFO as "cust"
COLUMNS
"CUSTNAME" CHAR(30) PATH '../name',
"PHONETYPE" CHAR(30) PATH '@type',
"PHONENUM" XML PATH '.') as X
WHERE CID=1001 OR CID=1003
```

Table 8-3 shows the result table of the SELECT statement.

Table 8-3 Result table of a query using XMLTABLE to retrieve multiple occurrences of an item

CUSTNAME	PHONETYPE	PHONENUM
Kathy Smith	work	<phone xmlns="http://posample.org" type="work">905-555-7258</phone>
Robert Shoemaker	work	<phone xmlns="http://posample.org" type="work">905-555-7358</phone>
Robert Shoemaker	home	<phone xmlns="http://posample.org" type="home">416-555-2937</phone>
Robert Shoemaker	cell	<phone xmlns="http://posample.org" type="cell">905-555-8743</phone>
Robert Shoemaker	cottage	<phone xmlns="http://posample.org" type="cottage">613-555-3278</phone>

You can specify a default value for any column in the XMLTABLE result table by using a DEFAULT clause. The default value is used if the XPath expression that defines the column returns an empty sequence. In this example, neither document includes an age element.

You can use a query shown in Example 8-7 to create a result table in which the column value is "***No age***" if a document has no age for a customer.

Example 8-7 Specifying a default value for a column in the result table

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
'$cust/customerinfo' PASSING C.INFO as "cust"
COLUMNS
"CUSTNAME" VARCHAR(30) PATH 'name',
"AGE" VARCHAR(30) PATH 'age' DEFAULT '***No age***') AS X
WHERE CID=1001 OR CID=1003
```

Table 8-4 shows the result table of the SELECT statement.

Table 8-4 Result table from a query in which XMLTABLE has a default value for an item

CUSTNAME	AGE
Kathy Smith	*** No age ***
Robert Shoemaker	*** No age ***

You can specify that the result table of an XMLTABLE invocation includes an ordinality column. An ordinality column has the following properties:

- Includes the BIGINT data type
- Starts at one for each document that generates a result table row
- Is incremented by one for each result table row that is generated by a single document

You can use a query as shown in Example 8-8 to create an ordinality column in the XMLTABLE result table.

Example 8-8 Specifying an ordinality column in a result table

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
'$cust/customerinfo/phone' PASSING C.INFO as "cust"
COLUMNS
"SEQNO" FOR ORDINALITY,
"PHONE TYPE" VARCHAR(15) PATH './@type',
"PHONE NUMBER" VARCHAR(15) PATH '.'
) AS X
WHERE CID=1003
```

Table 8-5 shows the result table of the SELECT statement.

Table 8-5 Result table from a query in which XMLTABLE has a ordinality column

SEQNO	PHONE TYPE	PHONE NUMBER
1	work	905-555-7358
2	home	416-555-2937
3	cell	905-555-8743
4	cottage	613-555-3278

Archived

8.1.2 The XMLCAST specification

Casting of SQL data types to XML data types occurs implicitly. The XMLCAST specification simplifies the task of converting an XML value to a SQL data type. This function was added by APARs PK51571, PK51572, and PK51573.

For casting from an XML schema data type to a SQL data type, the XMLCAST specification eliminates the need to serialize an XML value before casting it to a SQL data type. Casting of an XML schema data type to a SQL data type is useful if you want to return the results of the XMLQUERY function[1] as SQL for further processing in a SQL statement, such as comparing XML values to SQL values or using XML values to order result tables.

You can cast the result of XMLQUERY to a SQL data type only when the XPath expression that is specified in the XMLQUERY function returns a sequence that contains one item. Implicit casting of a SQL data type to an XML schema type occurs in the XMLEXISTS[2] predicate or the XMLQUERY or XMLTABLE function, when a column value is passed to an XPath expression. Although you can use the XMLCAST specification to cast a SQL data type to an XML schema data type, it is usually unnecessary. See Example 8-9.

Example 8-9 The XMLCAST specification: Implicit casting

```
SELECT XMLQUERY('/customerinfo/addr passing INFO)
WHERE XMLEXISTS('/customerinfo[name=$n]' PASSING INFO,'Kathy Smith' as "n")
or
SELECT XMLQUERY('/customerinfo/addr passing INFO)
WHERE XMLEXISTS('/customerinfo[name="Kathy Smith"]' PASSING INFO)
```

The SQL character string `'Kathy Smith'` that is not an XML type is implicitly cast to an XML type in the XMLEXISTS predicate. Following the implicit cast, the constant has the XML schema subtype of `xs:string` and is bound to the variable `$n`. The `$n` variable can then be used in the predicate of the XPath expression.

Notice the second coding alternative is more simple than the first alternative. The string `"Kathy Smith"` is specified within double quotation marks.

Example 8-10 shows that XMLQUERY retrieves the XML value from the document. XMLCAST is necessary because SQL cannot sort by XML types.

Example 8-10 The XMLCAST specification: Explicit casting

```
SELECT CID, INFO FROM CUSTOMER
ORDER BY XMLCAST (
        XMLQUERY ('$doc/customerinfo/@cid'
             PASSING INFO AS "doc")
        as integer);
```

In this example the XMLCAST specification casts values in XML documents in the customer table to a SQL data type so that those values can be used in an ORDER BY clause.

[1] The XMLQUERY function is a SQL scalar function that enables you to execute an XQuery expression from within a SQL context.

[2] The XMLEXISTS predicate tests whether an XQuery expression returns a sequence of one or more items.

8.1.3 XML index for XML joins

If an XMLEXISTS predicate contains a join of two tables, the join condition compares two XPath expressions. An XML index that the predicate uses must be on the first table in the join order, and the XPath expression must match both XPath expressions in the join condition.

XML index exploitation for XMLEXISTS predicate was extended by APARs PK55783 and PK81260.

The query in Example 8-11 retrieves XML documents from the CUSTOMER and ORDER tables for which a customer ID in the CUSTOMER table matches the customer ID in the ORDER table. The customer IDs are compared as strings.

Example 8-11 XML index usage by join predicates example 1

```
SELECT INFO FROM CUSTOMER, ORDER
WHERE XMLEXISTS('$x/customerinfo[@Cid = $y/order/customer/@id/fn:string(.)]'
passing CUSTOMER.INFO as "x", ORDER.ORDERINFO as "y")
```

The first table in the join is the ORDER table (that is, ORDER is the outer table). So the query can use the following XML index on the CUSTOMER table (that is, CUSTOMER is the inner table):

```
CREATE INDEX CUST_CID_STR ON CUSTOMER(INFO)
GENERATE KEYS USING XMLPATTERN
'/customerinfo/@Cid'
AS SQL VARCHAR(10)
```

Because the XPath expressions in the join predicate are compared as strings, the index must store entries in the index as the VARCHAR type.

The query in Example 8-12 retrieves XML documents from the ORDER and CUSTOMER tables for which a customer ID in the ORDER table matches the customer ID in the CUSTOMER table. The customer IDs are compared as numeric values.

Example 8-12 XML index usage by join predicates example 2

```
SELECT INFO FROM CUSTOMER, ORDER
WHERE XMLEXISTS('$y/order/customer[@id = $x/customerinfo/@id/xs:decimal(.)]'
passing CUSTOMER.INFO as "x", ORDER.ORDERINFO as "y")
```

The first table in the join is the CUSTOMER table (that is, CUSTOMER is the outer table). So the query can use the following XML index on the ORDER table (that is, ORDER is the inner table):

```
CREATE INDEX ORDER_CID_NUM ON ORDER(ORDERINFO)
GENERATE KEYS USING XMLPATTERN
'/order/customer/@id'
AS SQL DECFLOAT
```

Because the XPath expressions in the join predicate are compared as numeric values, the index must store entries in the index as the DECFLOAT type.

8.1.4 Index enhancements

DB2 9 provides XPath performance enhancements and index support for the following functions:

- fn:exists()
- fn:not()
- fn:upper-case()
- fn:starts-with()
- fn:substring()

These enhancements were added by APARs PK80732, PK80735, and PK96558.

XML index use by predicates that test for node existence

If an XMLEXISTS predicate contains the fn:exists or fn:not function, it matches an XML index that contains the fn:exists or fn:not function.

The query in Example 8-13 retrieves all `customerinfo` documents from the INFO column of the CUSTOMER table for which the addr node has a `zip` attribute.

Example 8-13 Query example using function fn:exists()

```
SELECT INFO FROM CUSTOMER
WHERE XMLEXISTS('$x/customerinfo/addr[fn:exists(@zip)]'
passing CUSTOMER.INFO as "x")
```

The query in Example 8-14 retrieves all customerinfo documents from the INFO column of the CUSTOMER table for which the addr node does not have a zip attribute.

Example 8-14 Query example using function fn:not()

```
SELECT INFO FROM CUSTOMER
WHERE XMLEXISTS('$x/customerinfo/addr[fn:not(@zip)]'
passing CUSTOMER.INFO as "x")
```

Both of these queries can use the following XML index:

```
CREATE INDEX CUST_ZIP_EXISTENCE on CUSTOMER(INFO)
GENERATE KEY USING XMLPATTERN '/customerinfo/addr/fn:exists(@zip)'
AS VARCHAR(1)
```

For queries that test for existence, VARCHAR(1) is the compatible data type for the XML index, because the index key values can be only `T` or `F`.

XML index use by predicates with case-insensitive comparisons

In XML, string comparisons are case-sensitive. However, some applications might require case-insensitive comparisons. You can use the fn:upper-case function to do a case-insensitive comparison.

The query in Example 8-15 retrieves all XML documents from the INFO column of the CUSTOMER table for customers whose child addr has a street whose upper-case is *BAILEY AVE*. The query returns the XML document when the street is *Bailey Ave*, or *bailey ave*, or *Bailey AVE*, and so on.

Example 8-15 Query example using function fn:upper-case()

```
SELECT INFO FROM CUSTOMER
WHERE XMLEXISTS('$x/customerinfo/addr[fn:upper-case(street) = "BAILEY AVE"]'
PASSING CUSTOMER.INFO AS "x")
```

The following index can be used for the predicate:

```
CREATE INDEX CUST_STREET_UPPER on CUSTOMER(INFO)
GENERATE KEY USING XMLPATTERN '/customerinfo/addr/street/fn:upper-case(.)'
AS VARCHAR(50)
```

The fn:lower-case function is not supported for index usage.

XML index use for XMLEXISTS predicate with fn:starts-with function

An XMLEXISTS predicate that contains the fn:starts-with function and an XML index need to meet the following conditions for an index match:

- In the XMLEXISTS predicate, the second argument of the fn:starts-with function must be a string literal.
- The XML index must have the VARCHAR type.
- The pattern expression in the index must match the XPath expression in the predicate, except for the fn:starts-with function.

The query in Example 8-16 includes a predicate that checks whether the `productName` value starts with the string `Lapis`.

Example 8-16 Query example using function fn:starts-with()

```
SELECT PORDER FROM PURCHASEORDER
WHERE XMLEXISTS(
'/purchaseOrder/items/item[fn:starts-with(productName,"Lapis")]'
PASSING PURCHASEORDER.PORDER)
```

The following index matches the predicate:

```
CREATE INDEX POSTRTSW ON PURCHASEORDER(PORDER)
GENERATE KEYS USING XMLPATTERN
'/purchaseOrder/items/item/productName'
AS SQL VARCHAR(20)
```

Archived

XML index use for XMLEXISTS predicate with fn:substring function

An XMLEXISTS predicate that contains the fn:substring function and an XML index need to meet the following conditions for an index match:

- In the XMLEXISTS predicate:
  - The second argument of the fn:substring function must be 1.
  - The operand to which the expression that contains the fn:substring function is compared is a string literal.
  - The third argument of the fn:substring function must be an integer constant that is equal to the length of the string literal.
- The pattern expression in the index must match the XPath expression in the predicate, except for the fn:substring function.

The query in Example 8-17 includes a predicate that checks whether part of the `productName` value is the string `Lapis`.

Example 8-17 Query example using function fn:substring()

```
SELECT PORDER FROM PURCHASEORDER
WHERE XMLEXISTS(
'/purchaseOrder/items/item[fn:substring(productName,1,5)= "Lapis"]'
PASSING PURCHASEORDER.PORDER)
```

The following index matches the predicate:

```
CREATE INDEX POSUBSTR ON PURCHASEORDER(PORDER)
GENERATE KEYS USING XMLPATTERN
'/purchaseOrder/items/item/productName'
AS SQL VARCHAR(20)
```

Example 8-17 uses the fn:substring() function, which is similar to Example 8-16 using the fn:starts-with() function. Notice also that the index definitions are identical.

8.1.5 XML index use by queries with XMLTABLE

This function was implemented by APARs PK51571, PK51572, and PK51573.

If the FROM clause of a query contains an XMLTABLE function with a row-XPath-expression and if DB2 transforms the query to contain an XMLEXISTS predicate, after transformation, the query can use an XML index.

The original query might have an XMLTABLE function with a row-XPath-expression, or the query might be the result of transformation of a SQL predicate to an XPath predicate in a row-XPath-expression.

Suppose that the CUSTOMER table contains the following document in the INFO column:

```
<customerinfo xmlns="http://posample.org" Cid="1010">
 <name>Elaine Decker</name>
  <addr zip="95999">
   <street>100 Oak</street>
   <city>Mountain Valley</city>
   <state>CA</state>
   <country>USA</country>
  </addr>
 <phone type="work">408-555-2310</phone>
</customerinfo>
```

The query in Example 8-18 uses the XMLTABLE function to retrieve the `zip`, `street`, `city`, and `state` elements from `customerinfo` documents as columns in a result table.

Example 8-18 Query example using XMLTABLE (original query)

```
SELECT X.*
FROM CUSTOMER,
 XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
 '$x/customerinfo/addr[@zip=95999]'
 PASSING INFO as "x"
 COLUMNS
  ZIP INT PATH '@zip',
  STREET VARCHAR(50) PATH 'street',
  CITY VARCHAR(30) PATH 'city',
  STATE VARCHAR(2) PATH 'state') AS X
```

The original query cannot use an XML index. However, the original query has a row-XPath-expression that DB2 can transform into an XMLEXISTS predicate. Example 8-19 shows the query after transformation.

Example 8-19 Query example using XMLTABLE (transformed query)

```
SELECT X.*
FROM CUSTOMER,
 XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
 '$x/customerinfo/addr[@zip=95999]'
 PASSING INFO as "x"
 COLUMNS
  ZIP INT PATH '@zip',
  STREET VARCHAR(50) PATH 'street',
  CITY VARCHAR(30) PATH 'city',
  STATE VARCHAR(2) PATH 'state') AS X
WHERE XMLEXISTS('declare default element namespace "http://posample.org";
$x/customerinfo/addr[@zip=95999]');
```

The transformed query can use the following index:

```
CREATE INDEX ORDER_ZIP_NUM ON CUSTOMER(INFO)
GENERATE KEYS USING XMLPATTERN
'declare default element namespace "http://posample.org/";
/customerinfo/addr/@zip'
AS SQL DECFLOAT
```

Archived

The XML index needs to be defined on the INFO column of the CUSTOMER table because the XMLEXISTS predicate in the transformed query uses the INFO column of the CUSTOMER table.

The query in Example 8-20 retrieves the same information as the query in Example 8-19, but a SQL predicate determines which rows are returned.

Example 8-20 Query example using XMLTABLE with WHERE clause (original query)

```
SELECT X.*
FROM CUSTOMER,
 XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
 '$x/customerinfo/addr'
 PASSING INFO AS "x"
 COLUMNS
  ZIP INT PATH '@zip',
  STREET VARCHAR(50) PATH 'street',
  CITY VARCHAR(30) PATH 'city',
  STATE VARCHAR(2) PATH 'state') AS X
WHERE X.ZIP = 95999
```

DB2 can transform the query so that the SQL predicate becomes an XPath predicate in the row-XPath-expression of the XMLTABLE function. Example 8-21 shows the transformed query.

Example 8-21 Query example using XMLTABLE with WHERE clause (First transformation)

```
SELECT X.*
FROM CUSTOMER,
 XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
 '$x/customerinfo/addr[@zip=95999]'
 PASSING INFO AS “x”
 COLUMNS
  ZIP INT PATH '@zip',
  STREET VARCHAR(50) PATH 'street',
  CITY VARCHAR(30) PATH 'city',
  STATE VARCHAR(2) PATH 'state') AS X
```

DB2 can then transform the query again so that the row-XPath-expression becomes an XMLEXISTS predicate. Example 8-22 shows the transformed query.

Example 8-22 Query example using XMLTABLE with WHERE clause (Second transformation)

```
SELECT X.*
FROM CUSTOMER,
 XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
 '$x/customerinfo/addr'
 PASSING INFO as "x"
 COLUMNS
  ZIP INT PATH '@zip',
  STREET VARCHAR(50) PATH 'street',
  CITY VARCHAR(30) PATH 'city',
  STATE VARCHAR(2) PATH 'state') AS X
WHERE XMLEXISTS(
‘declare default element namespace “http://posample.org”;
```

```
$x/customerinfo/addr[@zip=95999]'
 PASSING INFO AS "x")
```

The transformed query can use the following index:

```
CREATE INDEX ORDER_ZIP_NUM ON CUSTOMER(INFO)
GENERATE KEYS USING XMLPATTERN
'declare default element namespace "http://posample.org/";
/customerinfo/addr/@zip'
AS SQL DECFLOAT
```

Additional information: XMLEXISTS from XMLTABLE is for index use purposes only, and it is not evaluated by scan. After the rows are identified by indexes, XMLTABLE is evaluated using scan.

8.1.6 XPath scan improvement

The performance improvement was made by APARs PK80732, PK80735, and PK82631. For the XMLQUERY or XMLEXISTS functions, the performance of the XPath scanning algorithm is optimized as follows:

- Make simple XPath, such as `/a/b/c`, extremely fast
- Make simple XPath predicate, such as `/a/b[c=5]`, extremely fast
- Do not accumulate sequences but predicate results, for example `/a/b[c="abc" and d>5]`
- Push down the search during tree traversal, for example `/a/b[.//c=5]`

Avoid an XMLEXISTS XPath scan after XML indexes are matched as described in the following list:

- XMLEXISTS predicates are stage 2 predicates.
- The XPath in XMLEXISTS is re-evaluated by scan after XML index matches

  This can happen in case of a not exact match or after an update.
- Evaluating XPath using scan is quite CPU intensive.
- Under the following conditions, re-evaluation can be avoided and performance improved:
  - The XPath in the predicate matches exactly with the XPath for the index (prepare)
  - There is no update to the XML document since the start of the index scan (run time)

Figure 8-2 shows how DB2 avoids re-evaluation of the XMLEXISTS function.

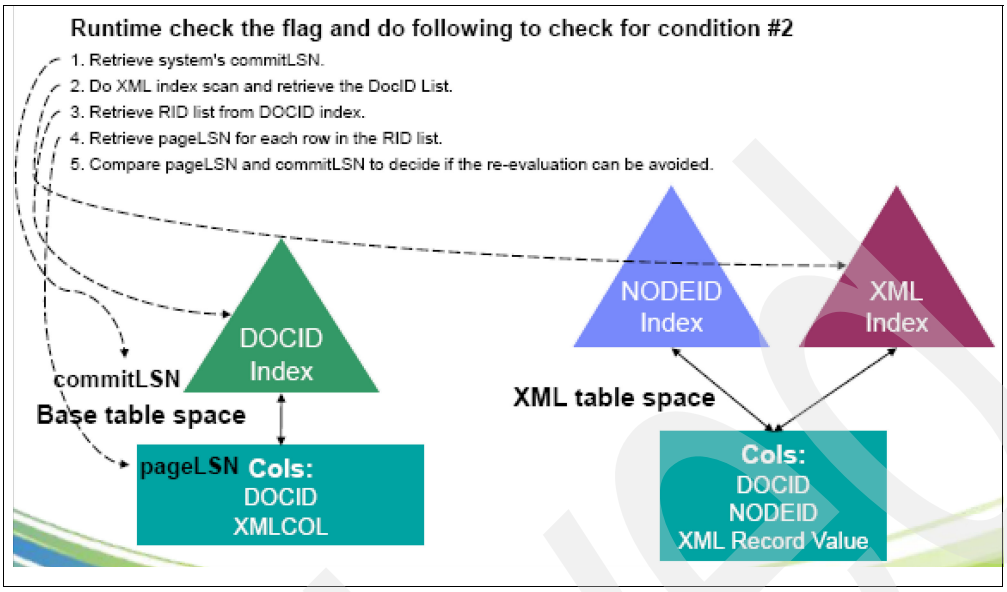


Figure 8-2 Avoid re-evaluation of XMLEXISTS

8.1.7 XPath functions

The service stream (PK55585, PK55831, and PK51537) introduced the following XPath functions:

- fn:name
- fn:local-name
- fn:max
- fn:min
- fn:distinct-values
- fn:upper-case
- fn:lower-case
- fn:translate
- fn:last
- fn:position
- fn:matches
- fn:replace
- fn:tokenize

These functions are described in the updated version of *DB2 9 for z/OS XML Guide*, SC18-9858. In addition, refer to *DB2 10 for z/OS pureXML Guide,* SC19-2981.

8.2 XML type modifier

The XML data type can accept any well-formed XML documents. However, in many cases, users want to store in one XML column documents that have a similar structure or that conform to the same XML schema. DB2 10 introduces the XML type modifier which qualifies the XML data type with a set of one or more XML schemas. The value of an XML column with an XML type modifier must conform to at least one XML schema specified in the type modifier.

When you define an XML column, you can add an *XML type modifier*. An XML type modifier associates a set of one or more XML schemas with the XML data type. You can use an XML type modifier to cause all XML documents that are stored in an XML column to be validated according to one of the XML schemas that is specified in the type modifier.

The XML type modifier can identify more than one XML schema. You might want to associate more than one XML schema with an XML type modifier for the following reasons:

- The requirements for an XML schema evolve over time.

  An XML column might contain documents that describe only one type of information, but some fields in newer documents might need to be different from fields in the older documents. As new document versions are required, you can add new XML schemas to the XML type modifier.

- A single XML column contains XML documents of different kinds.

  An XML column might contain documents that have several different formats. In this case, each type of document needs its own XML schema.

Alternatively, you might want to associate a single XML schema with multiple type modifiers. An XML schema can define many different documents. You might need to separate the XML documents into different columns, but specify the same XML schema in a type modifier for each column.

For example, a sales department might have one XML schema that defines purchase orders and billing statements. You can store purchase orders in one XML column, and billing statements in another XML column. Both XML columns have an XML type modifier that points to the same XML schema, but each column restricts documents with different root elements in the XML schema.

You define an XML type modifier in a CREATE TABLE or ALTER TABLE statement as part of an XML column definition.

Not all XML schemas that the XML type modifier identifies need to be registered before you execute the CREATE or ALTER statement. If the XML type modifier specifies a target namespace, only the XML schemas in that target namespace that exist when the CREATE or ALTER statement is executed are associated with the XML type modifier.

Figure 8-3 shows the XML schemas to which the following examples refer for defining an XML type modifier.

XML schema name	Target Namespace	Schema Location	Registration Timestamp
PO1	http://www.example.com/PO1	http://www.example.com/PO1.xsd	2009-01-01 10:00:00.0000
PO2	http://www.example.com/PO2	http://www.example.com/PO2.xsd	2010-01-01 10:00:00.0000
PO3	NO NAMESPACE	http://www.example.com/PO3.xsd	2010-01-30 10:00:00.0000
PO4	http://www.example.com/PO2	http://www.example.com/PO4.xsd	2010-02-23 08:00:00.000

Figure 8-3 XML schemas

Example 8-23 shows how to specify an XML type modifier for an XML column at create time.

Example 8-23 Specify XML type modifier for XML column at create time

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA ID SYSXSR.PO1))
```

A table for purchase orders contains an XML column named CONTENT. The documents in the XML column need to be validated according to XML schema SYSXSR.PO1, which has already been registered.

To alter an existing XML column to include an XML type modifier or remove an XML type modifier, use ALTER TABLE.

Example 8-24 shows the table definition without XML type modifier specified.

Example 8-24 Table definition without XML type modifier

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML)
```

The table contains several XML documents. The documents in the XML column need to be validated according to XML schema SYSXSR.PO1, which has already been registered. Alter the XML column to add an XML type modifier that specifies SYSXSR.PO1 as shown in Example 8-25.

Example 8-25 Specify XML type modifier for XML column at alter time

```
ALTER TABLE PURCHASEORDERS
  ALTER CONTENT
  SET DATA TYPE XML(XMLSCHEMA ID SYSXSR.PO1)
```

The table space that contains the XML documents for the CONTENT column is put in CHECK-pending status.

You can add an XML schema to the XML type modifier.

Example 8-26 shows an example.Suppose PO2 is a new version of the purchase order schema. You can use the ALTER table statement to reset the XML type modifier of the CONTENT column to include both PO1 and PO2.

Example 8-26 Add an XML schema to the XML type modifier

```
ALTER TABLE PURCHASEORDERS
  ALTER CONTENT
  SET DATA TYPE XML(XMLSCHEMA ID SYSXSR.PO1, ID SYSXSR.PO2)
```

Because the XML schema specified in the old type modifier is a subset of the new type modifier, the existing values of the CONTENT column do not need to be validated again. Thus, the state of the XML table space for the CONTENT column stays unchanged. If the XML schema specified in the old XML type modifier is *not* a subset of the XML schema specified in the new XML type modifier, the XML table space for the column that is being changed is left in the CHECK-pending status.

You can also reset the data type of CONTENT to XML without type modifier.

Example 8-27 shows how to reset XML type modifier for an XML column at alter time.

Example 8-27 Reset XML type modifier for XML column at alter time

```
ALTER TABLE PURCHASEORDERS
  ALTER CONTENT
  SET DATA TYPE XML
```

Note: The existing values of the CONTENT column do not need to be validated again.

Validation is performed automatically for INSERT and UPDATE SQL statements and the LOAD utility if the XML column is defined with the XML type modifier.

The following catalog tables include information for supporting XML type modifiers:

- SYSIBM.SYSXMLTYPMOD contains rows for the XML type modifiers.
- SYSIBM.SYSXMLTYPSCHEMA contains a row for each XML schema specification for one XML type modifier.

Instead of specifying the schema name directly as shown in all the examples, it is possible to specify the URI and LOCATION keywords, so that the schema name can be derived.

Example 8-28 shows how to specify the schema location hint. Both PO2 and PO4 have the same target namespace (`http://www.example.com/PO2`). If you want to use PO2, you can add `LOCATION 'http://www.example.com/PO2.xsd'` after the `URI 'http://www.example.com/PO2'` clause.

Example 8-28 Identify an XML schema by target namespace and schema location

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA URI 'http://www.example.com/PO2'
                        LOCATION 'http://www.example.com/PO2.xsd' ))
```

Example 8-29 shows the XML type modifier using only the URI keyword to identify the XML schema.

Example 8-29 Identify an XML schema by target namespace

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA URI 'http://www.example.com/PO2'))
```

If you execute the CREATE TABLE statement before PO4 is registered, only PO2 is added to the type modifier in SYSIBM.SYSXMLTYPSCHEMA. When PO4 is registered later, the XML type modifier for the CONTENT column remains unchanged. If you execute the CREATE TABLE statement after PO4 is registered, an SQL error occurs because the XML type modifier uses the URI keyword to identify two XML schemas PO2 and PO4. The URI keyword must identify only one XML schema.

If an XML schema does not contain the `targetNamespace` attribute in its schema document, it can be referenced in the XML type modifier by NO NAMESPACE.

In Example 8-30, DB2 chooses PO3 as the XML type modifier.

Example 8-30 No namespace

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA NO NAMESPACE
                        LOCATION 'http://www.example.com/PO3.xsd' ))
```

If an XML schema has more than one global element declaration and if you want to validate the XML value against one of them, you can specify the ELEMENT clause. Assume the purchase order schema has declared two global elements:

- purchaseOrder
- comment

Thus, a document whose root element is either purchaseOrder or comment could be valid according to PO1 and can be stored in the PURCHASEORDERS table. However, this might not be desirable. If you want to store only purchase order documents in the CONTENT column, you can specify ELEMENT "purchaseOrder" in the XML type modifier for CONTENT, as shown in Example 8-31.

Example 8-31 Specifying global element name

```
CREATE TABLE PURCHASEORDERS(
  ID INT NOT NULL,
  CONTENT XML(XMLSCHEMA ID SYSXSR.PO1
                        ELEMENT "purchaseOrder"))
```

8.3 XML schema validation

XML schema validation is the process of determining whether the structure, content, and data types of an XML document are valid according to an XML schema. In addition, XML schema validation strips ignorable white space from the input document.

In DB2 9, XML schema validation is done by invoking the SYSFUN.DSN_XMLVALIDATE user-defined function, which must be invoked within the XMLPARSE function. This validation requires setting up and administering the WLM application environment. You cannot take advantage of DB2 DRDA zIIP redirect due to the user-defined function.

Validating XML parsing "inside the engine" using XMLSS provided with DB2 10 was also retrofitted to DB2 9 with APARs PK90032 and PK90040. To use the SYSIBM.DSN_XMLVALIDATE function, follow these steps:

1. Ensure that the z/OS operating system level is R1.9 with PTF OA25903 or R1.10. If you use this function without this z/OS level, DB2 issues the following message:

   ```
   SQLCODE -904: reason-code 00D50005, resource-type 00003000, resource-name z/OS
   1.9 or 1.10 XML System Services.
   ```

2. Run the following command after each initial program load (IPL):

   ```
   SETPROG LPA,ADD,MODNAME=(GXLIMODV),DSNAME=SYS1.SIEALNKE
   ```

   Otherwise, the SYSIBM.DSN_XMLVALIDATE functions fails with the following message:

   ```
   SQLCODE -904 and reason code 00D50005.
   ```

3. Qualify DSN_XMLVALIDATE explicitly with SYSIBM. When DSN_XMLVALIDATE is unqualified or qualified with SYSFUN, the user-defined function version of DSN_XMLVALIDATE is invoked.

8.3.1 Enhancements to XML schema validation

You can validate an XML document in DB2 10 using one of the following methods:

- Automatically, by including an XML type modifier in the XML column definition in a CREATE TABLE or ALTER TABLE statement. When a column has an XML type modifier, DB2 implicitly validates documents that are inserted into the column or when documents in the column are updated.
- Manually, by executing the new SYSIBM.DSN_XMLVALIDATE built-in function when you insert a document into an XML column or update a document in an XML column.

Validation is optional when you insert data into an XML column with no XML type modifier. Validation is mandatory when you insert data into an XML column with an XML type modifier.

We examine both methods in this section.

XML schema validation with an XML type modifier

You can automate XML schema validation by adding an XML type modifier to an XML column definition. Before schema validation through an XML type modifier can occur, all schema documents that make up an XML schema must be registered in the built-in XML schema repository (XSR).

Figure 8-4 shows the table definition and the entries in the XSR for the schemas used as XML type modifiers in the table definition.

XML schema name	Target Namespace	Schema Location	Registration Timestamp
PO1	http://www.example.com/PO1	http://www.example.com/PO1.xsd	2009-01-01 10:00:00.0000
PO2	http://www.example.com/PO2	http://www.example.com/PO2.xsd	2010-01-01 10:00:00.0000
PO3	NO NAMESPACE	http://www.example.com/PO3.xsd	2010-01-30 10:00:00.0000
PO4	http://www.example.com/PO4	http://www.example.com/PO4.xsd	2010-02-23 08:00:00.000

Table definition:

```
CREATE TABLE PURCHASE_ORDERS(
    ID          INT   NOT NULL
    CONTENT XML (XMLSCHEMA ID SYSXSR.PO1, ID SYSXSR.PO2,
                           ID SYSXSR.PO3, ID SYSXSR.PO4)
                                )
```

Figure 8-4 XML schemas in XML Schema Repository

Figure 8-5 shows how DB2 chooses an XML schema when the XML type modifier specifies multiple schemas.

Archive

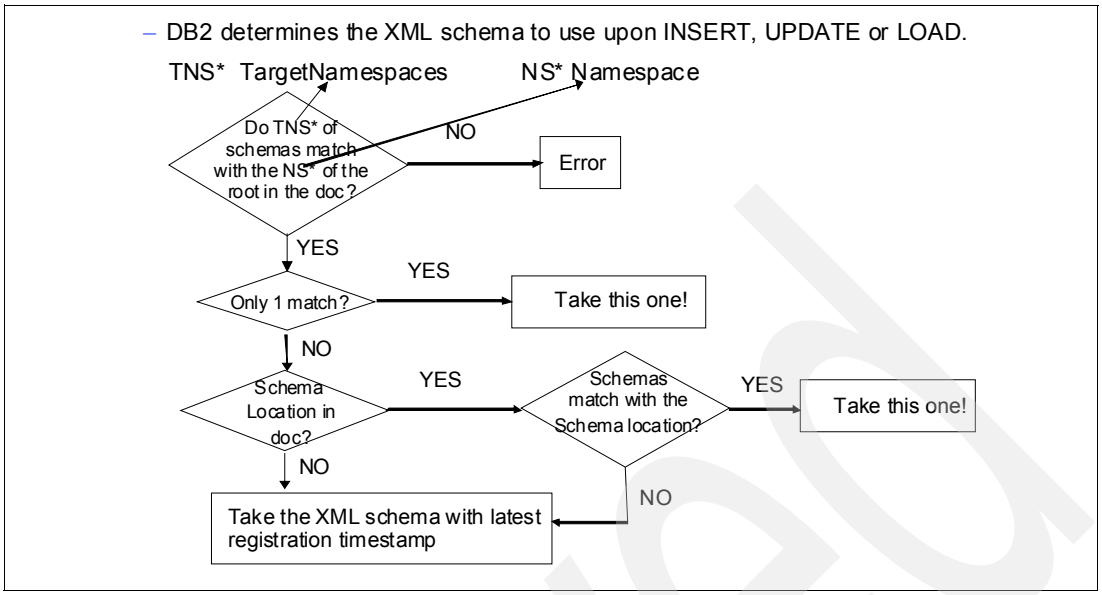


Figure 8-5 Schema determination

You can include more than one XML schema in an XML type modifier. When you insert into or update an XML column, DB2 chooses one XML schema to do validation.

DB2 uses the following process to determine which XML schema to use:

- If the operation is an update operation, and an XML schema that is specified by the XML type modifier has already been used to validate the original document, DB2 uses the same XML schema to validate the updated document.
- If there is only one XML schema whose target namespace matches the namespace name of the root element node in the document that is being validated (the XML *instance document*), DB2 chooses that XML schema to validate the XML document.
- If there is more than one XML schema with a target namespace that matches the namespace name of the root element, DB2 chooses an XML schema by using the *schema location hint*. The root element node of an XML instance document can contain an xsi:schemaLocation attribute. That attribute consists of one or more pairs of URI references, separated by white space. The first member of each pair is a namespace name, and the second member of the pair is a URI that describes where to find an appropriate schema document for that namespace. The second member of each pair is the schema location hint for the namespace name that is specified in the first member.

For example, the following example shows a schema location attribute:

```
xsi:schemaLocation="http://www.example.com/PO2 http://www.example.com/PO4.xsd"
```

The first member of the pair, `http://www.example.com/PO2`, is the namespace name. The second member of the pair, `http://www.example.com/PO4.xsd`, is the URI that provides the schema location hint.

DB2 uses the schema location hint to choose an XML schema in the following way:

1. If the root element node contains an xsi:schemaLocation attribute, DB2 searches the attribute value for a schema location hint with a corresponding namespace name that matches the namespace name in the root element node.

2. If DB2 finds a schema location hint, DB2 uses the hint to identify an XML schema whose schema location URI is identical to the schema location hint. DB2 validates the input document against that schema.
3. If the root element does not contain an xsi:schemaLocation attribute, or the xsi:schemaLocation attribute does not contain a schema location hint with a corresponding namespace name that matches the namespace name in the root element node, DB2 uses the XML schema with the same target namespace and the latest registration time stamp.

The following examples illustrate how DB2 determines the schema that is used for validation from an XML type modifier. In Example 8-32, DB2 chooses XML schema PO1.

Example 8-32 Schema selection for validation from an XML type modifier: Example 1

```
INSERT INTO PURCHASE_ORDERS VALUES(1,
‘<po:purchaseOrder xmlns:po="http://www.example.com/PO1">
...
</po:purchaseOrder>‘);
```

The namespace name in the root element of the instance document is `http://www.example.com/PO1`. This name matches only the target namespace for XML schema PO1.

In Example 8-33, DB2 chooses XML schemas PO2 and PO4 in this order.

Example 8-33 Schema selection for validation from an XML type modifier: Example 2

```
INSERT INTO PURCHASE_ORDERS VALUES(2,
‘<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO2.xsd">
...
</po:purchaseOrder>‘);
```

The namespace name in the root element in the instance document is `http://www.example.com/PO2`. which matches the target namespace of XML schemas PO2 and PO4. The root element of the instance document also contains an xsi:schemaLocation attribute whose value provides the schema location hint, `http:// www.example.com/PO2.xsd`. The schema location hint matches the schema location for XML schema PO2. Therefore, DB2 chooses PO2 to validate the instance document. If validation with PO2 fails, DB2 uses PO4.

In Example 8-34, DB2 chooses XML schemas PO4 and PO2 in this order.

Example 8-34 Schema selection for validation from an XML type modifier: Example 3

```
INSERT INTO PURCHASE_ORDERS VALUES(3,
‘<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO4.xsd">
...
</po:purchaseOrder>‘);
```

The namespace name in the root element in the instance document is `http:// www.example.com/PO2`, which matches the target namespace of XML schemas PO2 and PO4. The root element of the instance document also contains an xsi:schemaLocation attribute whose value provides the schema location hint, `http:// www.example.com/PO4.xsd`. The schema location hint matches the schema location for XML schema PO4. Therefore, DB2 chooses PO4 to validate the instance document. If validation with PO4 fails, DB2 uses PO2.

In Example 8-35, DB2 chooses XML schema PO3.

Example 8-35 Schema selection for validation from an XML type modifier: Example 4

```
INSERT INTO PURCHASE_ORDERS VALUES(4,
'<purchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.example.com/PO3.xsd">
...
</purchaseOrder>');
```

The root element of the instance document has no namespace name. XML schema PO3 has no target namespace. Therefore, DB2 uses PO3 for validation.

Note that, after you update an XML document in a column that has an XML type modifier, DB2 validates again all or part of the document.

- If the XML type modifier includes several XML schemas, DB2 uses the same XML schema for validation that is used for the original validation.
- If you update an entire document, DB2 validates the entire document. However, if you use the XMLMODIFY[3] function to update only a portion of the document, DB2 might need to validate only the updated portion.

XML schema validation with DSN_XMLVALIDATE

Another method to perform XML schema validation is by executing the DSN_XMLVALIDATE built-in function. Before you can invoke DSN_XMLVALIDATE, all schema documents that make up an XML schema must be registered in the XML schema repository. You can use DSN_XMLVALIDATE with a type modifier. DB2 checks if the schema used in DSN_XMLVALIDATE is one of the schemas in the type modifier and skips double validation. This method can be used to override a schema picked by DB2.

There are a number of forms of DSN_XMLVALIDATE:

- DSN_XMLVALIDATE(*string-expression*)
- DSN_XMLVALIDATE(*xml-expression*)
- DSN_XMLVALIDATE(*string-expression*, *varchar-expression*)
- DSN_XMLVALIDATE(*xml-expression*, *varchar-expression*)
- DSN_XMLVALIDATE(*string-expression1*, *string-expression2*, *string-expression3*)
- DSN_XMLVALIDATE(*xml-expression1*, *string-expression2*, *string-expression3*)

For all forms, the first parameter contains the document that you want to validate.

For forms with one parameter, the target namespace and optional schema location of the XML schema must be in the root element of the instance document that you want to validate.

For forms with two parameters, the second parameter is the name of the schema object to use for validation of the document. That object must be registered in the XML schema repository.

[3] The XMLMODIFY function returns an XML value that might have been modified by the evaluation of an XPath updating expression and XPath variables that are specified as input arguments.

For forms with three parameters, the second and third parameter contain the names of a namespace URI and a schema location hint that identify the XML schema object to use for validation of the document. That object must be registered in the XML schema repository

XML schemas used for validation are the same as shown in Figure 8-4 on page 263. However, the following CREATE TABLE statement is used to create the table:

```
CREATE TABLE PURCHASE_ORDERS(
  ID      INT   NOT NULL
  CONTENT XML )
```

Examples of the criteria how DB2 chooses XML schema for DSN_XMLVALIDATE are listed here. In Example 8-36 DB2, chooses XML schema PO1.

Example 8-36 Schema selection for validation for DSN_XMLVALIDATE: Example 1

```
INSERT INTO PURCHASE_ORDERS VALUES(1,
DSN_XMLVALIDATE('<po:purchaseOrder xmlns:po="http://www.example.com/PO1">
...
</po:purchaseOrder>'));
```

The DSN_XMLVALIDATE invocation does not specify an XML schema or target namespace and schema location hint. Thus, DB2 uses the information in the instance document. The namespace name in the root element of the instance document is `http://www.example.com/PO1`. This name matches only the target namespace for XML schema PO1.

In Example 8-37, DB2 chooses XML schema PO2.

Example 8-37 Schema selection for validation for DSN_XMLVALIDATE: Example 2

```
INSERT INTO PURCHASE_ORDERS VALUES(2,
DSN_XMLVALIDATE('<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO2.xsd">
...
</po:purchaseOrder>','SYSXSR.PO2'));
```

The DSN_XMLVALIDATE invocation specifies XML schema SYSXSR.PO2.

In Example 8-38, DB2 chooses XML schema PO4.

Example 8-38 Schema selection for validation for DSN_XMLVALIDATE: Example 3

```
INSERT INTO PURCHASE_ORDERS VALUES(3,
DSN_XMLVALIDATE('<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO4.xsd">
...
</po:purchaseOrder>','http://www.example.com/PO2'));
```

The DSN_XMLVALIDATE invocation specifies namespace `http://www.example.com/PO2`. Two XML schemas, PO2 and PO4, have that target namespace. DB2 uses PO4, because it has the later time stamp.

In Example 8-39, DB2 chooses PO3.

Example 8-39 Schema selection for validation for DSN_XMLVALIDATE: Example 4

```
INSERT INTO PURCHASE_ORDERS VALUES(4,
DSN_XMLVALIDATE('<purchaseOrder
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.example.com/PO3.xsd">
...
</purchaseOrder>'));
```

The DSN_XMLVALIDATE invocation does not specify an XML schema or target namespace and schema location hint. Thus, DB2 uses the information in the instance document. The root element node in the instance document contains an xsi:noNamespaceSchemaLocation attribute with the value `http://www.example.com/PO3.xsd`. Thus, DB2 uses XML schema PO3, which has no target namespace, and the schema location `http://www.example.com/PO3.xsd`.

There have been two versions of DSN_XMLVALIDATE:

- A user-defined function
- A built-in function

The user-defined function is deprecated. Now, DB2 uses the built-in function instead, even in DB2 9.

To move from the DSN_XMLVALIDATE user-defined function to the DSN_XMLVALIDATE built-in function:

- For applications that invoke DSN_XMLVALIDATE using the qualified name SYSFUN.DSN_XMLVALIDATE:
  a. Change the name to SYSIBM.DSN_XMLVALIDATE.

     This change is optional. DB2 drops SYSFUN.DSN_XMLVALIDATE and invalidates packages during migration. Automatic rebinds pick up SYSIBM.DSN_XMLVALIDATE.

  b. Prepare the applications again.
- For applications that invoke DSN_XMLVALIDATE without using the qualified name, you do not need to modify the applications. DB2 uses the SYSIBM.DSN_XMLVALIDATE built-in function automatically.
- Optional: Remove the XMLPARSE function that surrounds DSN_XMLVALIDATE.

  The SYSFUN.DSN_XMLVALIDATE user-defined function must be invoked from within the XMLPARSE function. The SYSIBM.DSN_XMLVALIDATE built-in function does not need to be invoked from within the XMLPARSE function.

The SYSFUN.DSN_XMLVALIDATE user-defined function is deprecated. Use the SYSIBM.DSN_XMLVALIDATE built-in function instead. Even if you explicitly call SYSFUN.DSN_XMLVALIDATE, DB2 runs SYSIBM.DSN_XMLVALIDATE.

8.3.2 Determining whether an XML document has been validated

You can use the SQL XMLXSROBJECTID scalar function to determine whether an XML document that is stored in a table has undergone XML validation and to determine which XML schema was used to validate that document.

XMLXSROBJECTID returns the XSR object identifier of the XML schema that was used to validate the input XML document. The XSR object identifier corresponds to the

XSROBJECTID column in the SYSIBM.XSROBJECTS catalog table. After you call XMLXSROBJECTID, you can use the returned value to query SYSIBM.XSROBJECTS for the XML schema information. If the XML document has not been validated, XMLXSROBJECTID returns 0.

The SQL statement in Example 8-40 calls XMLXSROBJECTID to determine which XML documents in the INFO column of the CUSTOMER table have not been validated. The statement then calls DSN_XMLVALIDATE to validate those documents against XML schema SYSXSR.PO1.

Example 8-40 Search for documents not validated

```
UPDATE CUSTOMER
SET INFO = DSN_XMLVALIDATE(INFO, 'SYSXSR.PO1')
WHERE XMLXSROBJECTID(INFO)=0
```

The SQL statement in Example 8-41 retrieves the target namespaces and XML schema names for the XML schemas that were used to validate XML documents in the INFO column of the CUSTOMER table.

Example 8-41 Retrieve target namespaces and XML schema names used for validation

```
SELECT DISTINCT S.XSROBJECTNAME, S.TARGETNAMESPACE
FROM CUSTOMER C, SYSIBM.XSROBJECTS S
WHERE XMLXSROBJECTID(INFO) = S.XSROBJECTID
```

8.4 XML consistency checking with the CHECK DATA utility

The CHECK DATA utility checks XML relationships and can check the consistency between a base table space and the corresponding XML table spaces. If the base table space is not consistent with any related XML table spaces, the CHECK DATA utility reports the error.

The default behavior of CHECK DATA is to check all objects that are in CHECK-pending status (SCOPE PENDING[4]). However, you can limit the scope of checking by specifying SCOPE REFONLY to check only the base tables or SCOPE AUXONLY to check XML and LOB objects.

You can specify the action that DB2 performs when it finds an error in XML columns by using keyword XMLERROR. XMLERROR REPORT reports only the error. XMLERROR INVALIDATE reports the error and sets the column in error to an invalid status.

You can also specify the action that DB2 performs when it finds an error in LOB or XML columns by using the AUXERROR keyword. AUXERROR REPORT reports only the error. AUXERROR INVALIDATE reports the error and sets the column in error to an invalid status.

You do not normally specify both XMLERROR and AUXERROR.

The CHECK DATA utility checks only for the consistency between the base table and the NODEID index. the CHECK INDEX utility checks consistency between the XML table space and the NODEID index.

There is no function to check that all nodes of all XML documents are stored in the XML table space correctly, especially if XML documents are spanned across multiple rows in the XML table space.

[4] SCOPE PENDING will include all objects in restrictive or advisory state.

The DB2 10 CHECK DATA utility includes the following enhancements:

- Base table consistency with NODEID index (as in DB2 9 CHECK INDEX utility)
- Check consistency between the XML table space and the NODEID index (as in DB2 9 CHECK INDEX utility)
- Check consistency in the document structure for each XML document
- Validate schema if XML columns have type modifier

Figure 8-6 shows the CHECK DATA utility syntax for new keywords.

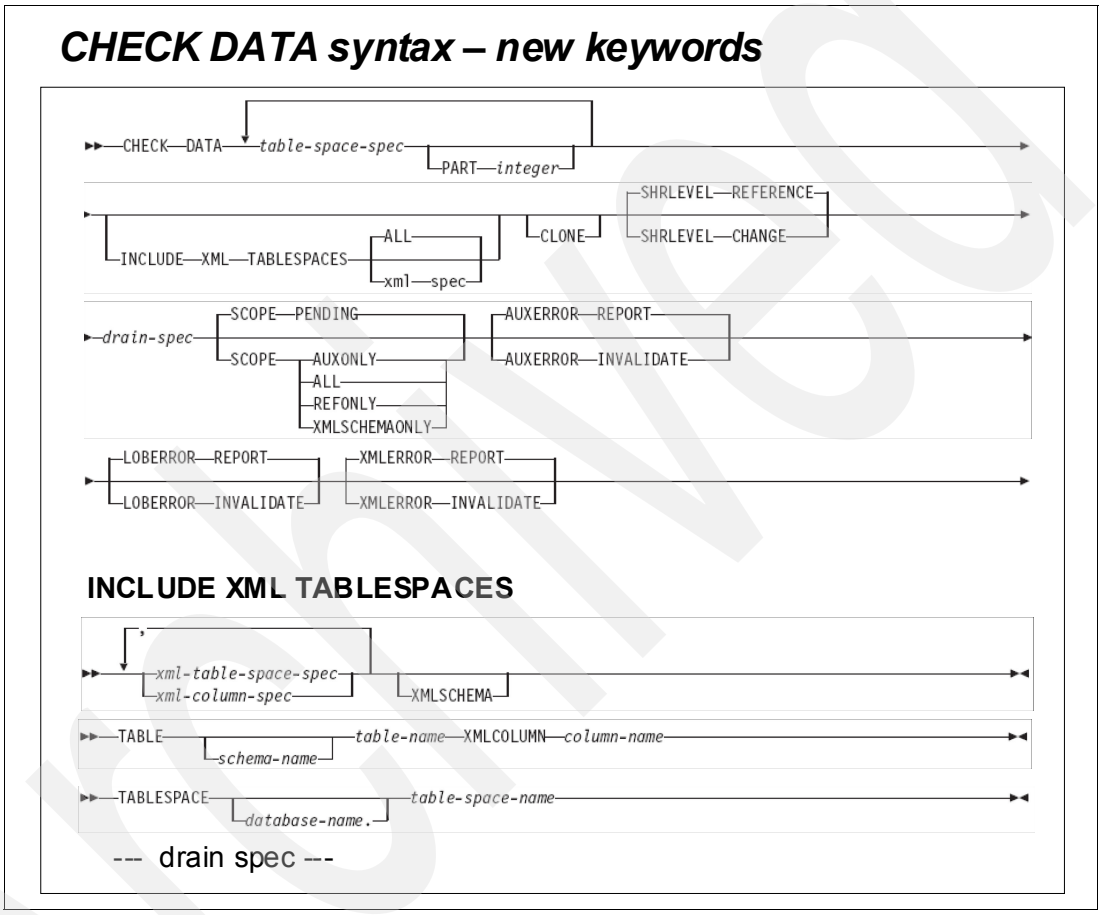


Figure 8-6 CHECK DATA syntax: New keywords

DB2 10 includes the INCLUDE XML TABLESPACES and SCOPE XMLSCHEMAONLY options.

If you specify the INCLUDE XML TABLESPACES option, the CHECK DATA utility can check the structural integrity of XML documents. The CHECK DATA utility verifies the following items for XML objects:

- All rows in an XML column exist in the XML table space.
- All documents in the XML table space are structurally valid.
- Each index entry in the NODEID index has a corresponding XML document.
- Each XML document in the XML table space has corresponding entries in the NODEID index.

- Each entry in the DOCID column in the base table space has a corresponding entry in the NODEID index over the XML table space, if the XML column is not null.
- Each entry in the NODEID index contains a corresponding value in the DOCID column.
- If an XML column has an XML type modifier, all XML documents in the column are valid with respect to at least one XML schema that is associated with the XML type modifier.

If the base table space is not consistent with any related XML table spaces or if a problem is found during any of the previously listed checks, the CHECK DATA utility reports the error.

For XML checking, the default behavior of the CHECK DATA utility is to check only the consistency between each XML column and its NODEID index. However, you can modify the scope of checking by specifying combinations of the CHECK DATA SCOPE keyword and the INCLUDE XML TABLESPACES keyword. For LOBs, the CHECK DATA utility checks for the consistency between the base table and the auxiliary index.

Table 8-6 is a reference table for the CHECK DATA utility invocation based on the combination of the various options, which are applicable for LOBs as well.

Table 8-6 CHECK DATA invocation

CHECK DATA base table space(s)	INCLUDE XML TABLESPACES	XMLSCHEMA	SCOPE	Structure check	Schema validation	XML checks	LOB checks
X			ALL/AUXONLY	-	-	Yes	Yes
X			REFONLY	-	-	-	-
X			XMLSCHEMA ONLY	-	Yes Default: INCLUDE XML TABLESPACES ALL	-	-
X			PENDING	-	-	Yes Base table spaces in CHKP/ACHKP	Yes Base table spaces in CHKP/ACHKP
X	X		ALL/AUXONLY	Yes Specified XMLtable spaces only	-	Yes All XML table spaces	Yes All LOB table spaces
X	X		REFONLY	-	-	-	-
X	X		XMLSCHEMA ONLY	-	Yes Specified XML table spaces only	-	-
X	X		PENDING	-	Yes Specified XML table spaces in CHKP	Yes Base table spaces in CHKP/ACHKP	Yes Base table spaces in CHKP/ACHKP
X	X	X	ALL/AUXONLY	Yes Specified XML table spaces only	Yes Specified XML table spaces only	Yes All XML table spaces	Yes All LOB table spaces
X	X	X	REFONLY	-	-	-	-
X	X	X	XMLSCHEMA ONLY	-	Yes Specified XML table spaces only	-	-

CHECK DATA base table space(s)	INCLUDE XML TABLESPACES	XMLSCHEMA	SCOPE	Structure check	Schema validation	XML checks	LOB checks
X	X	X	PENDING	-	Yes Specified XML table spaces in CHKP	Yes Base table spaces in CHKP/ACHKP	Yes Base table spaces in CHKP/ACHKP

The “XML checks” column indicates that the CHECK DATA utility checks only for the consistency between the base table and the NODEID index.

The “LOB checks” column indicates that the CHECK DATA utility checks for the consistency between the base table and the auxiliary index.

Example 8-42 shows an example of CHECK DATA utility control statement.

Example 8-42 CHECK DATA example

```
CHECK DATA TABLESPACE DSNXDX1A.DSNXSX1D SCOPE ALL
INCLUDE XML TABLESPACES(TABLE XMLTBL XMLCOLUMN XMLCOL) XMLSCHEMA
```

Table space DSNXDX1A.DSNXSX1D contains a table XMLTBL with XML column XMLCOL, which has an XML type modifier. If you specify the statement as shown in Example 8-42, the CHECK DATA utility checks LOB relationships, the base table space, XML relationships, and the structural integrity of XML documents for column XMLCOL and does XML schema validation on the documents for column XMLCOL.

You can use SHRLEVEL REFERENCE or SHRELEVEL CHANGE.

Figure 8-7 shows considerations for SHRLEVEL REFERENCE.

- … **SHRLEVEL REFERENCE and AUXERROR/XMLERROR REPORT**
  - If no problem found, remove CHKP or ACHKP from table spaces
  - If problem found:
    - DOCID of XML document is printed in the job output
    - No further action
- **… SHRLEVEL REFERENCE and AUXERROR/XMLERROR INVALIDATE**
  - If no problem found, remove CHKP or ACHKP from table spaces
  - If problem found:
    - DOCID of XML document is printed in the job output
    - Exception tables automatically generated for XML column (schema validation only)
    - Affected XML documents moved to XML exception table (schema validation only)
    - Corrupted XML document deleted from XML table space
    - Index entries for corrupted XML documents removed from NODEID index
    - Invalid bit set in the XML indicator in the base table space
    - Value indexes not touched/checked by CHECK DATA

Figure 8-7 The CHECK DATA utility: SHRLEVEL REFERENCE considerations

Figure 8-8 shows considerations with SHRLEVEL CHANGE.

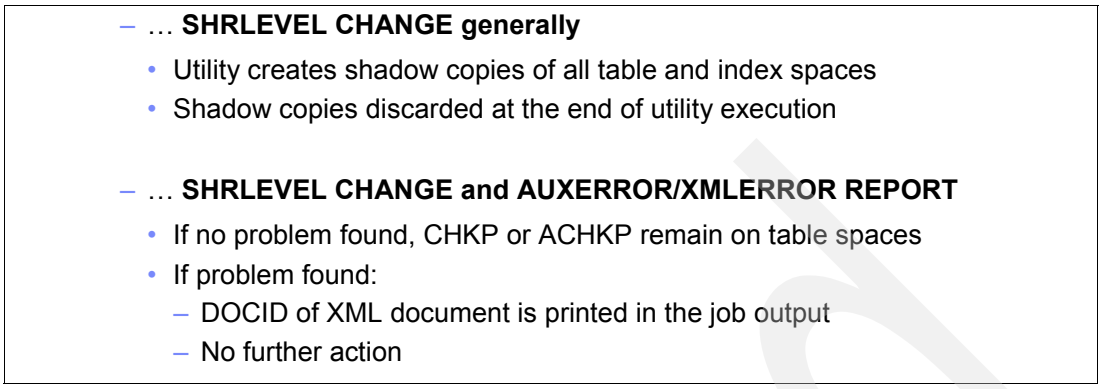


Figure 8-8 CHECK DATA: SHRLEVEL CHANGE considerations (1 of 2)

Figure 8-9 shows more considerations for SHRLEVEL CHANGE.

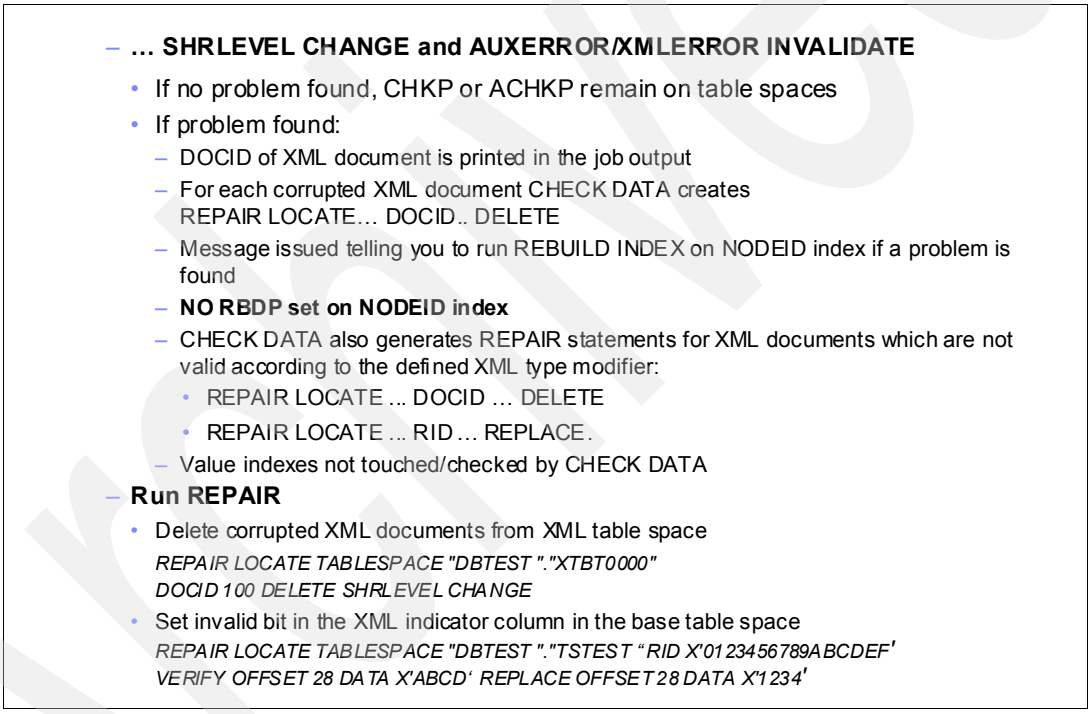


Figure 8-9 CHECK DATA: SHRLEVEL CHANGE considerations (2 of 2)

8.5 Support for multiple versions of XML documents

In DB2 9, the application developers and DBAs have to react to the following situations:

- During the execution of a SQL statement, a row from a base table with an XML column can be inserted into a work file, for example when sort is required for an ORDER BY. The row in the work file does not contain the actual records for the XML document. Instead, the internal reference to the XML document from the XML table is kept in the work file.

  To prevent other transactions from deleting or updating the same XML document, XML locks need to be held for all the rows in the work file. The problem occurs if the XML document is deleted or updated multiple times by the same transaction that inserted the

Archived

row in the work file. When the row in the work file is fetched, DB2 cannot find the expected XML document in the XML table, and the fetch fails with a SQLCODE -270. The work file inconsistency problem occurs only for XML columns. Extra application logic is needed to avoid a SQLCODE -270.

- A SQL statement that modifies the value of an XML column might need to reference the version of the XML document before it is modified. For example, in an UPDATE statement, if there are any columns in a SET clause expression, for each row that is updated, the values of the columns in the expression need to be the values of the columns in the row before the row is updated. Thus, for XML columns, you need to keep the original versions of the XML documents before they are updated. See Example 8-43.

Example 8-43 Self referencing UPDATE of an XML column

```
UPDATE T1 SET XML1 = XML2, XML2 = XML1
```

The updated value for column XML2 must be original value of column XML1. To keep the original versions of the XML columns from being destroyed, DB2 copies the records for the XML documents into memory. If the XML document is very large, the XML storage limit might be reached, resulting in SQLCODE -904. The DBA needs to assess whether the XML storage usage is expected or not and then adjust the DSNZPARM XMLVALA or XMLVALS parameters for the XML storage limit accordingly. If application developers need to update large XML documents, extra application logic might be needed to avoid SQLCODE -904.

To address these issues and others, DB2 10 introduces multiversioning for XML data.

8.5.1 XML versions

Multiple versions of an XML document can coexist in an XML table. The existence of multiple versions of an XML document can lead to improved concurrency through lock avoidance. In addition, multiple versions can save real storage by avoiding a copy of the old values in the document into memory in some cases.

DB2 supports multiple versions of an XML document in an XML column if the *base table space* for the table that contains the XML column is also a *universal table space*. All XML columns in the table support multiple versions.

If the base table space is not a universal table space, it does not support multiple XML versions. To convert the base table space from either segmented or partitioned to universal table space, drop it and re-create it. ALTER and REORG are not sufficient in this case.

With XML versions, when you insert an XML document into an XML column, DB2 assigns a version number to the XML document. If the entire XML document is updated, DB2 creates a new version of the document in the XML table. If a portion of the XML document is updated, DB2 creates a new version of the updated portion. When DB2 uses XML versions, more data set space is required than when versions are not used. However, DB2 periodically deletes versions that are no longer needed. In addition, you can run the REORG utility against the XML table space that contains the XML document to remove unneeded versions. DB2 removes versions of a document when update operations that require the versions are committed, and when there are no readers that reference the unneeded versions.

XML versions are different from table space versions or index versions. The purpose of XML versions is to optimize concurrency and memory usage. The purpose of table space and index versions is to maximize data availability

Example of improved concurrency with XML versions

The following example demonstrates how multiple XML versions can improve concurrency when the same XML documents are modified multiple times within the same transaction.

Suppose that table T1, which is in a universal table space, is defined as follows:

```
CREATE TABLE T1(
  INT1 INT,
  XML1 XML,
  XML2 XML );
```

Table 8-7 shows the data in table T1.

Table 8-7 Data in table T1

INT1	XML1	XML2
350	<A1>111</A1>	<A2>aaa</A2>
100	<A1>111</A1>	<A2>aaa</A2>
250	<A1>111</A1>	<A2>aaa</A2>

An application performs SQL read operations that are represented by the following pseudocode:

```
EXEC SQL
 DECLARE CURSOR C1 FOR
 SELECT INT1, XML1
  FROM T1
  ORDER BY INT1
  FOR READ ONLY;
```

At the same time, another application performs SQL write operations that are represented by the following pseudocode:

```
EXEC SQL UPDATE T1
 SET XML1 = XMLPARSE(DOCUMENT '<B1>222</B1>');
EXEC SQL OPEN CURSOR C1; (Note: Cursor C1 is in another application as described)
EXEC SQL UPDATE T1
 SET XML1 = XMLPARSE(DOCUMENT '<C1>333</C1>');
EXEC SQL FETCH FROM C1 INTO :HVINT1, :HVXML1;
```

With multiple versions, the reading application does not need to hold a lock. Thus, the updating application can complete its update operations without waiting for the reading application to finish. The reading application reads the old versions of the XML values, which are consistent data.

Example of improved storage usage with XML versions

The following example demonstrates how multiple XML versions can result in the use of less real storage when an XML document is the object of a self-referencing update operation.

Assume the same table T1 and data rows.

An application performs SQL operations that are represented by the following pseudocode:

```
EXEC SQL
 UPDATE T1
 SET XML1 = XML2, 1
```

```
  XML2 = XML1 2
 WHERE INT1 = 100;
EXEC SQL
 COMMIT 3;
```

The results of those operations are:

1. When column XML1 is updated, DB2 stores the updated document as a new version in the XML table for column XML1. There are now two versions of the XML document for the second row of column XML1:
   - First version: `<A1>111</A1>`
   - Second version: `<A2>aaa</A2>`
2. When column XML2 is updated, DB2 stores the updated document as a new version in the XML table for column XML2. There are now two versions of each XML document for the second row of column XML2:
   - First version: `<A2>aaa</A2>`
   - Second version: `<A1>111</A1>`
3. The update operations are committed. Thus, the old versions are no longer needed. DB2 deletes those versions from the XML tables for columns XML1 and XML2. (assuming no other readers are interested in reading these values).

Without multiple XML versions, DB2 needs to copy the original versions of the updated documents into memory, so that their values are not lost. For large XML documents, storage shortages might result.

8.5.2 Storage structure for XML data with versions

The storage structure of XML data now supports XML versions, as shown in Figure 8-10.

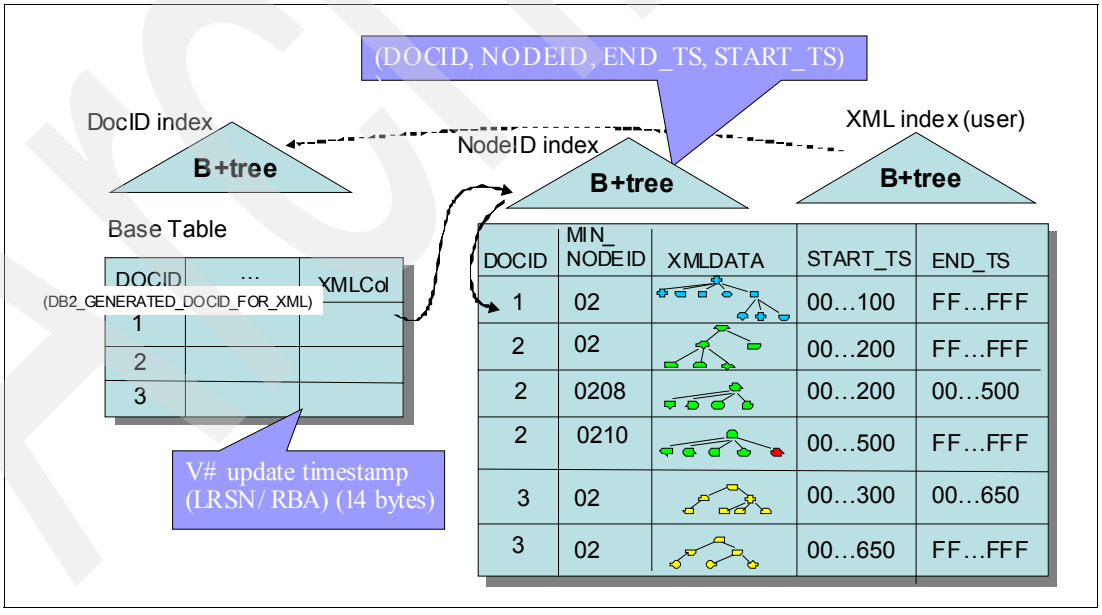


Figure 8-10 Multiversioning scheme

When you create a table with XML columns or use ALTER to change a table to add XML columns, DB2 implicitly creates the following objects:

- A table space and table for each XML column

The data for an XML column is stored in the corresponding table.

DB2 creates the XML table space and table in the same database as the table that contains the XML column (the *base table*). The XML table space is in the Unicode UTF-8 encoding scheme.

If the base table contains XML columns that support XML versions, each XML table contains two more columns than an XML table for an XML column that does not support XML versions. Those columns are named START_TS and END_TS, and they have the BINARY(8) data type. The START_TS column contains the RBA or LRSN of the logical creation of an XML record. The END_TS column contains the RBA or LRSN of the logical deletion of an XML record. The START_TS and END_TS columns identify the rows in the XML table that make up a version of an XML document.

The START_TS column represents the time when that row is created, and the END_TS column represents the time when the row is deleted or updated. The END_TS column contains `X'FFFFFFFFFFFFFFFF'` initially. To avoid compression causing update overflow, columns up to the END_TS column are not compressed in the reordered row format.

- A document ID column in the base table named DB2_GENERATED_DOCID_FOR_XML with data type BIGINT

  We refer to this as the DOCID column. The DOCID column holds a unique document identifier for the XML columns in a row. One DOCID column is used for all XML columns.

  The DOCID column has the GENERATED ALWAYS attribute. Therefore, a value in this column cannot be NULL. However, It can be null for rows that existed before a table is altered to add an XML column.

  If the base table space supports XML versions, the length of the XML indicator column is eight bytes longer that the XML indicator column in a base table space that does not support XML versions. That is, 14 bytes instead of six bytes.

- An index on the DOCID column

  This index is known as a document ID (or DOCID) index.

- An index on each XML table that DB2 uses to maintain document order, and map logical node IDs to physical record IDs

  This index is known as a NODEID index. The NODEID index is an extended, non-partitioning index.

  If the base table space supports XML versions, the index key for the NODEID index contains two more columns than the index key for a node ID index for a base table space that does not support XML versions. These columns are START_TS and END_TS.

Figure 8-11 gives a general idea of how DB2 handles multiversioning for XML data.

Each row in the XML auxiliary table is associated with two temporal attributes (start and end) to represent the period when the row exists. Start represents the time when that row is created, end represents the time when the row is deleted or expired.

For example, an XML document is stored as two rows in the XML auxiliary table at time t0, the second row is modified at t2, DB2 sets that row expired at t2, creates a new row representing the modified version with create time t2. The first row is not changed during this process.

A row in XML auxiliary table is never deleted until the garbage collector cleans it up.

When an XML document is deleted at time t2, all the records for that document are marked expired at t2. When a row of an XML document is updated, all the records for that document are marked expired at t2, the new document is inserted into XML auxiliary table with start time set to t2.

When a part of an XML document is modified, only the existing record or records to be modified expire, and a new version of those records is created.

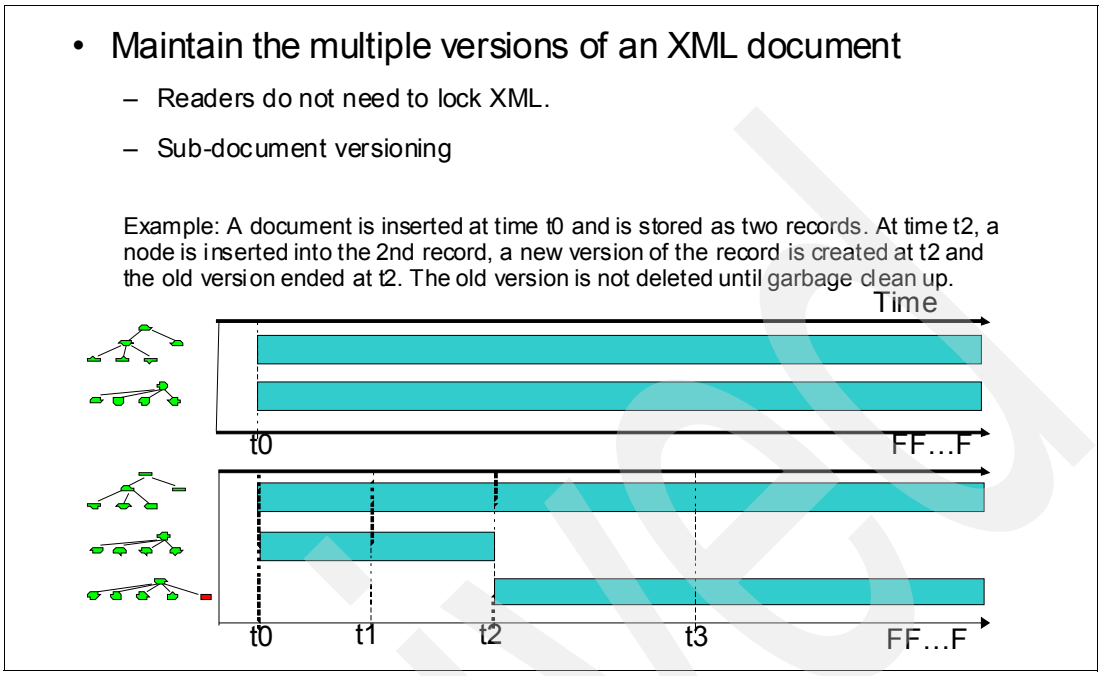


Figure 8-11 Multiversioning for XML data

This storage structure is possible only in new-function mode and for universal table spaces. Storage structure for multiversioning is a prerequisite for several other XML enhancements, such as:

- Access of currently committed data
- “As Of” for time-oriented data
- XML Update with XMLMODIFY
- Removing restrictions for SELECT FROM UPDATE/DELETE for XML

Archived

In DB2 9, APAR PK55966 (PTF UK33456) eliminates the page locks for readers on XML table space. Figure 8-12 shows the XML locking scheme in DB2 9 with the reader's locks stricken out.

SQL	Base Page/Row Lock (Business as usual)	XML Lock	XML Table space Page Lock
INSERT	x page/row lock	x lock, release at commit	Page latch with P-lock (changed from page locking by APAR PK68265)
UPDATE/DELETE	u->x, s->x, x stays x	x lock, release at commit	Page latch with P-lock (changed from page locking by APAR PK68265)
SELECT UR, SELECT CS- CURRENT DATA NO	None	s lock, release at next row fetch	s page lock, release at next row fetch
SELECT CS- CURRENT DATA YES no workfile	s page/row lock, release on next row fetch	s lock, release at next row fetch	s page lock, release at next row fetch
SELECT CS- CURRENT DATA YES workfile	s page/row lock, release on next row fetch	s lock, release at close cursor	s page lock, release at close cursor
SELECT UR, SELECT CS- CURRENT DATA NO, SELECT CS- CURRENT DATA YES with multirow fetch and dynamic scrolling	s page/row lock on rowset, release on next fetch	s lock, release on next fetch	s page lock, release on next fetch
SELECT RR, SELECT RS	s page/row lock	s lock, release at commit	s page lock, release at commit

Figure 8-12 XML locking scheme (with DB2 9 APAR PK55966)

Figure 8-13 shows the kind of locks that are removed and stricken out because of the introduction of multiversioning support in DB2 10.

SQL	Base Page/Row Lock (Business as usual)	XML Lock	XML Table space Page Lock
INSERT	x page/row lock	x lock, release at commit	Page latch (and optional P-Lock)
UPDATE/DELETE	u->x, s->x, x stays x	x lock, release at commit	Page latch (and optional P-Lock)
SELECT UR, SELECT CS- CURRENT DATA NO	None	Conditional lock s lock, release at next row fetch	
SELECT CS- CURRENT DATA YES no workfile	s page/row lock, release on next row fetch	s lock, release at next row fetch	
SELECT CS- CURRENT DATA YES workfile	s page/row lock, release on next row fetch	s lock, release at close cursor	
SELECT UR, SELECT CS- CURRENT DATA NO, SELECT CS- CURRENT DATA YES with Multirow fetch and dynamic scrolling	s page/row lock on rowset, release on next fetch	s lock, release on next fetch	
SELECT RR, SELECT RS	s page/row lock	s lock, release at commit	

Figure 8-13 XML Locking scheme with multiversioning

Archive

With support for multiversioning, you can use the following statement to delete all rows with PID like '%101%':

```
SELECT *
FROM OLD TABLE (DELETE FROM PRODUCT
 WHERE PID LIKE '%101%')
```

This statement returns the deleted rows to SELECT.

In DB2 9, SELECT FROM UPDATE is possible only with the FINAL TABLE keyword and *not* with the OLD TABLE keyword, because old versions of the XML columns are not kept.

8.6 Support for updating part of an XML document

In DB2 9, if you make modifications to part of an XML document stored in a DB2 table, the application cannot specify the required modification to the XML document. Applications that require parts of XML documents to be modified need to break apart the XML document into modifiable pieces, make the modification to a piece of the XML document, and then construct the pieces back into a single XML document. DB2 10 includes support for updating part of an XML document. In this section, we describe this enhancement.

8.6.1 Updates to entire XML documents

To update an entire XML document in an XML column, you use the SQL UPDATE statement. You include a WHERE clause when you want to update specific rows. XML data in an application can be in textual XML format or binary XML format. Binary XML format is valid only for JDBC, SQLJ, and ODBC applications.

You can use XML column values to specify which rows are updated. To find values within XML documents, you need to use XPath expressions. One way of specifying XPath expressions is the XMLEXISTS predicate, which allows you to specify an XPath expression and to determine if the expression results in an empty sequence. When XMLEXISTS is specified in the WHERE clause, rows are updated if the XPath expression returns a non-empty sequence.

The input to the XML column must be a well-formed XML document, as defined in the XML 1.0 specification. The application data type can be XML (XML AS BLOB, XML AS CLOB, or XML AS DBCLOB), character, or binary. When you update data in an XML column, it must be converted to its XML hierarchical format. DB2 performs this operation implicitly when XML data from a host variable directly updates an XML column. Alternatively, you can invoke the XMLPARSE function explicitly when you perform the update operation to convert the data to the XML hierarchical format.

The following examples demonstrate how to update XML data in XML columns. The examples use the MYCUSTOMER table, which has a SMALLINT CID column and an XML INFO column. The examples assume that MYCUSTOMER already contains a row with a customer ID value of 1004.

The XML data that is used to replace existing column data is in the c7.xml file, and looks as shown in Example 8-44.

Example 8-44 XML data

```
<customerinfo xmlns="http://posample.org" Cid="1004">
<name>Christine Haas</name>
<addr country="Canada">
<street>12 Topgrove</street>
<city>Toronto</city>
<prov-state>Ontario</prov-state>
<pcode-zip>N9Y-8G9</pcode-zip>
</addr>
<phone type="work">905-555-5238</phone> <phone type="home">416-555-2934</phone>
</customerinfo>
```

Example 8-45 shows a JDBC sample application. In a JDBC application, read XML data from the c7.xml file as binary data, and use it to update the data in an XML column,

Example 8-45 JDBC application example

```
PreparedStatement updateStmt = null;
String sqls = null;
int cid = 1004;
sqls = "UPDATE Customer SET Info=? WHERE Cid=?";
updateStmt = conn.prepareStatement(sqls);
updateStmt.setInt(2, cid);
File file = new File("c7.xml");
updateStmt.setBinaryStream(1, new FileInputStream(file), (int)file.length());
updateStmt.executeUpdate();
```

Example 8-46 shows a COBOL sample application.

Example 8-46 COBOL example

```
In an embedded COBOL application, update data in an XML column from an XML AS CLOB
host variable:
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 CID SMALLINT VALUE 1004
01 XMLBUF USAGE IS SQL TYPE IS XML AS CLOB(5K).
01 CLOBBUF USAGE IS SQL TYPE IS CLOB(5K).
EXEC SQL END DECLARE SECTION END-EXEC.
* Read data from file C7.xml into host variables XMLBUF and CLOBBUF
* as XML
EXEC SQL UPDATE MYCUSTOMER SET INFO = : XMLBUF WHERE CID = :CID END-EXEC.
* as CLOB
EXEC SQL UPDATE MYCUSTOMER SET INFO = XMLPARSE(:CLOBBUF) WHERE CID = :CID
END-EXEC.
```

In these examples, the value of the CID attribute within the <customerinfo> element is stored in the CID relational column as well. Thus, the WHERE clause in the UPDATE statements use the relational column CID to specify the rows to update. In the case where the values that determine which rows are chosen for update are only within the XML documents themselves, the XMLEXISTS predicate can be used. For example, the UPDATE statement in the previous embedded COBOL application example can be changed to use XMLEXISTS as shown in Example 8-47.

Example 8-47 Row filtering using XMLEXISTS

```
EXEC SQL UPDATE MYCUSTOMER SET INFO = :XMLBUF
WHERE XMLEXISTS ('declare default element namespace "http://posample.org";
/customerinfo[@Cid = $c]'
passing INFO, cast(:CID as integer) as "c");
```

8.6.2 Partial updates of XML documents

To update part of an XML document in an XML column, you can use the SQL UPDATE statement with the XMLMODIFY built-in scalar function.

The XMLMODIFY function specifies a *basic updating expression* that you can use to insert nodes, delete nodes, replace nodes, or replace the values of a node in XML documents that are stored in XML columns.

Note: Before you use XMLMODIFY to update part of an XML document, the column that contains the XML document must support XML versions. You can create the base table that contains XML columns in a universal table space.

The types of basic updating expressions are as follows:

- Insert expression

  Inserts copies of one or more nodes into a designated position in a node sequence.

- Replace expression

  Replaces an existing node with a new sequence of zero or more nodes, or replaces a node's value and preserves the node's identity.

- Delete expression

  Deletes zero or more nodes from a node sequence.

Insert expression syntax

Figure 8-14 shows insert expression syntax.

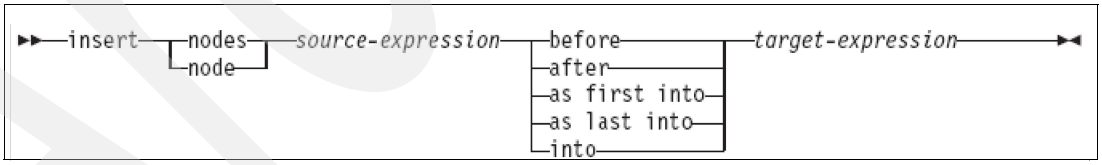


Figure 8-14 Insert expression syntax

Figure 8-15 shows examples of the usage of insert expression with the XMLMODIFY built-in function. The sample XML document is stored in the XML column INFO in table PERSONINFO. The keywords that begin an insert expression can be either insert nodes or insert node, regardless of the number of nodes to be inserted. The `source-expression` and the `target-expression` are XPath expressions that are not updating expressions.

Archived

Insert expression (1 of 3)

Sample XML document:

```
<person xmlns="http://sample">
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

Sample insert statement:

```
update personinfo set info = XMLMODIFY('
 declare default element namespace "http://sample";
  insert node $ins
  after  person/nickName', XMLPARSE( '
      <ename xmlns="http://sample">Joe.Smith@de.ibm.com</ename>') as  "ins") ;
```

Insert operation	Resulting XML document
Insert node $ins **into** person', XMLPARSE(....) (DB2 always inserts as last child)	<person xmlns="http://sample"> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename xmlns="http://sample">Joe.Smith@de.ibm.com</ename>** </person>
Insert node $ins **as last into** person', XMLPARSE(...)	<person xmlns="http://sample"> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename xmlns="http://sample">Joe.Smith@de.ibm.com</ename>** </person>

Figure 8-15 Insert expression examples (1 of 3)

Figure 8-16 shows examples of the usage of insert expression with the XMLMODIFY built-in function.

Insert expression (2 of 3)

```
<person xmlns="http://sample">
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

Insert operation	Resulting XML document
Insert node $ins **as first into** person', XMLPARSE(...)	<person xmlns="http://sample"> **<ename xmlns="http://sample">Joe.Smith@de.ibm.com</ename>** <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> </person>
Insert node $ins **after** person/nickName', XMLPARSE(...)	<person xmlns="http://sample"> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename xmlns="http://sample">Joe.Smith@de.ibm.com</ename>** </person>
Insert node $ins **before** person/nickName', XMLPARSE(...)	<person xmlns="http://sample"> <firstName>Joe</firstName> <lastName>Smith</lastName> **<ename xmlns="http://sample">Joe.Smith@de.ibm.com</ename>** <nickName>Joey</nickName> </person>

Figure 8-16 Insert expression examples (2 of 3)

Figure 8-17 shows an example of the usage of insert expression with the XMLMODIFY built-in function.

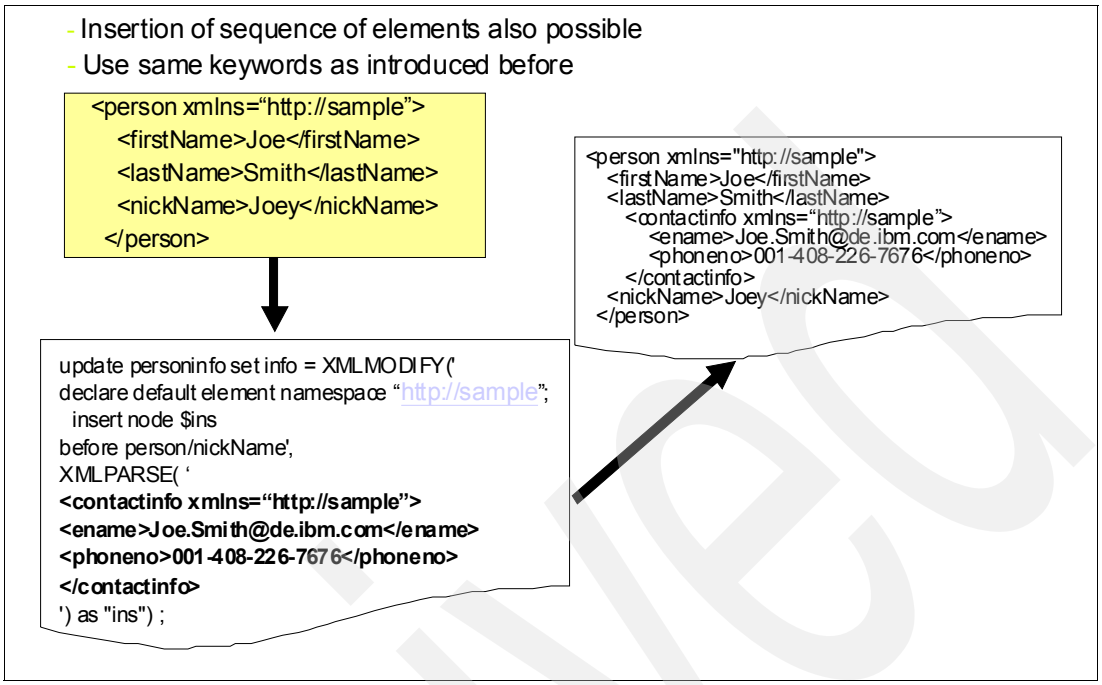


Figure 8-17 Insert expression examples (3 of 3)

Replace expression syntax

Figure 8-18 shows the replace expression syntax.

Figure 8-18 Replace expression syntax

Figure 8-19 shows an example of replacing the value of node with the XMLMODIFY built-in function.

Archive

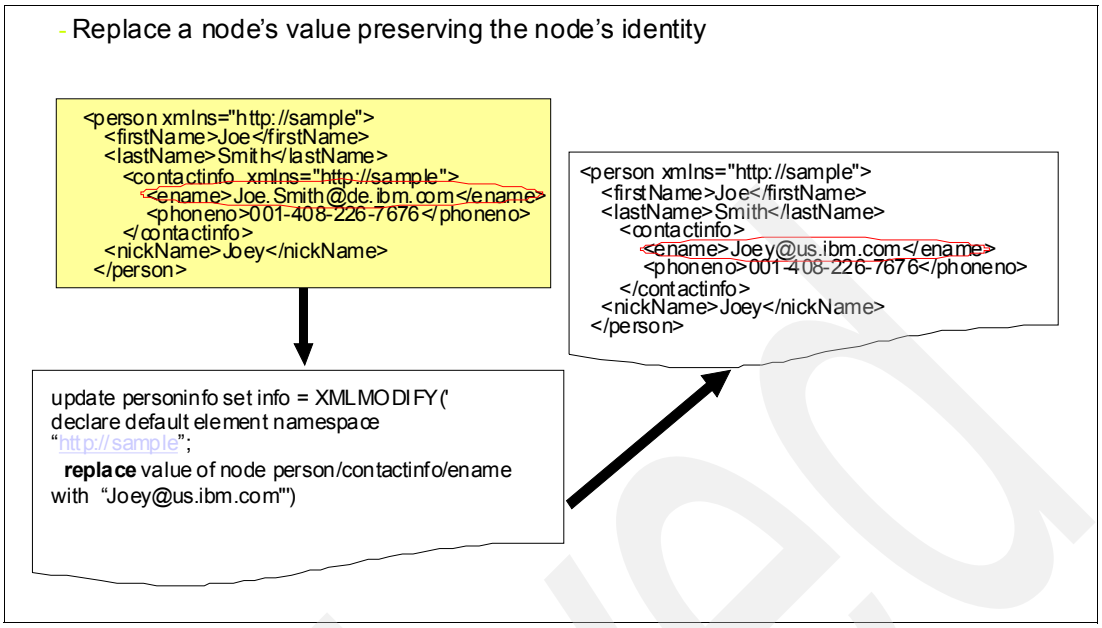


Figure 8-19 Replace expression examples (1 of 2)

Figure 8-20 shows an example of replacing an existing node by new nodes with the XMLMODIFY built-in function.

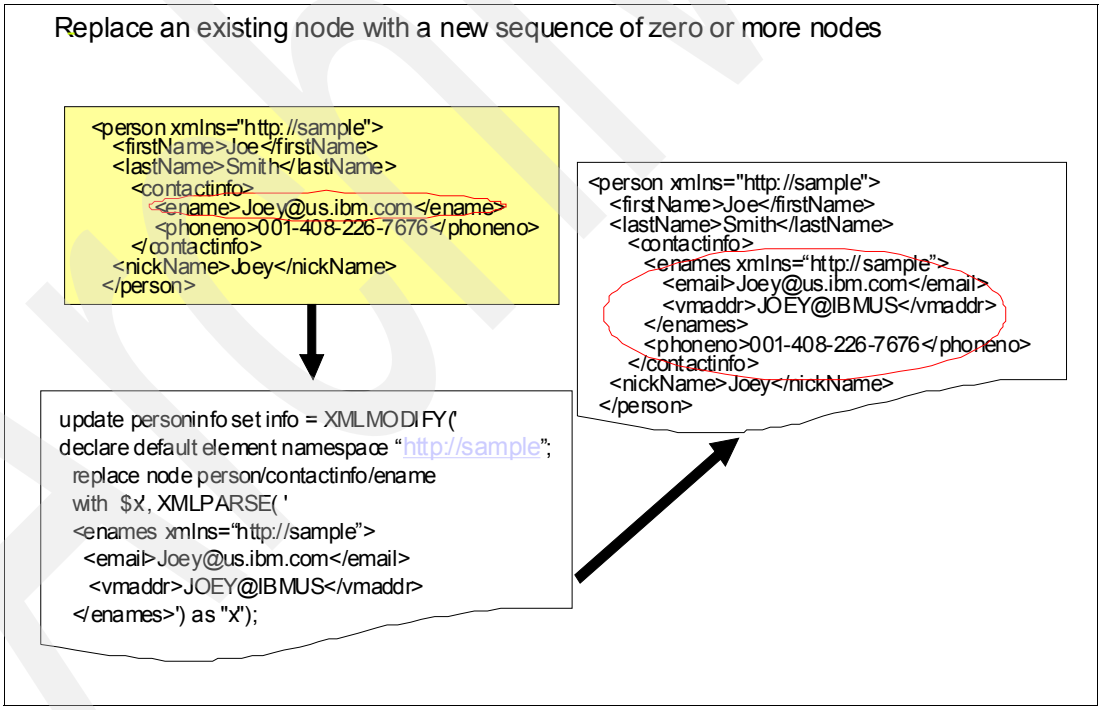


Figure 8-20 Replace expression examples (2 of 2)

Archive

Delete expression syntax

Figure 8-21 shows the delete expression syntax.

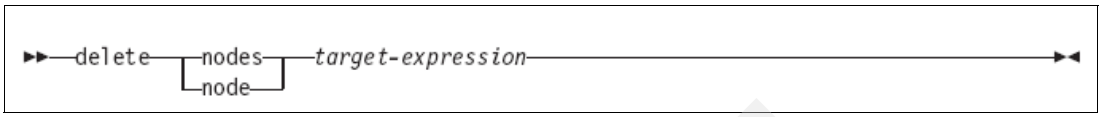


Figure 8-21 Delete expression syntax

Figure 8-22 shows an example of the usage of delete expression with the XMLMODIFY built-in function. The keywords that begin a delete expression can be either delete nodes or delete node, regardless of the number of nodes to be deleted.

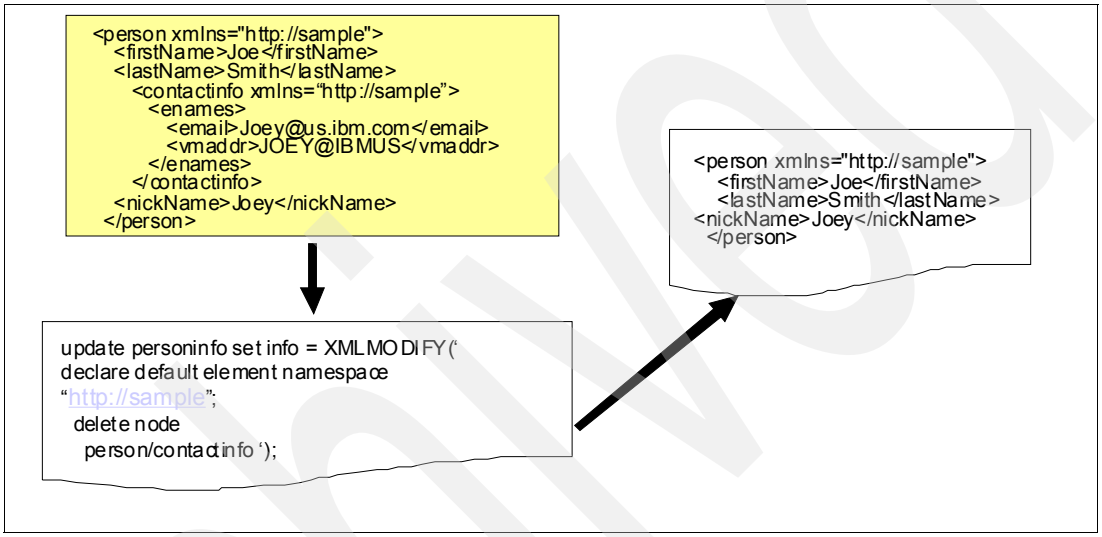


Figure 8-22 Delete expression example

To conclude this discussion, note the following information:

- XMLMODIFY can be used only in the right side of UPDATE SET.
- One update at a time for a document with concurrency control by the base table as follows:
  - Row level locking, page level locking, and other functions
  - Document level lock to prevent UR reader
- Only changed rows in XML table are updated.
- If there is a schema type modifier on the updated XML column, partial validation occurs. Global constraints are not validated.

8.7 Support for binary XML

The binary XML format is formally called *Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format*. Binary XML format is an external representation of an XML value that can be used for exchange of XML data between a client and a data server. The binary representation provides efficient XML parsing, which can result in performance improvements for XML data exchange. Support of binary XML provides performance improvements, because the XML data can be encoded more efficiently.

You can find the binary XML specification at:

http://www.ibm.com/support/docview.wss?uid=swg27019354

Storage and retrieval of binary XML data requires version 4.9 or later of the IBM Data Server Driver for JDBC and SQLJ. If you are using binary XML data in SQLJ applications, you also need version 4.9 or later of the `sqlj4.zip` package.

DB2 10 supports the use of binary XML format between applications and the server.

Binary does *not* mean FOR BIT DATA. Figure 8-23 summarizes the reasons why XML binary format is not the same as FOR BIT DATA.

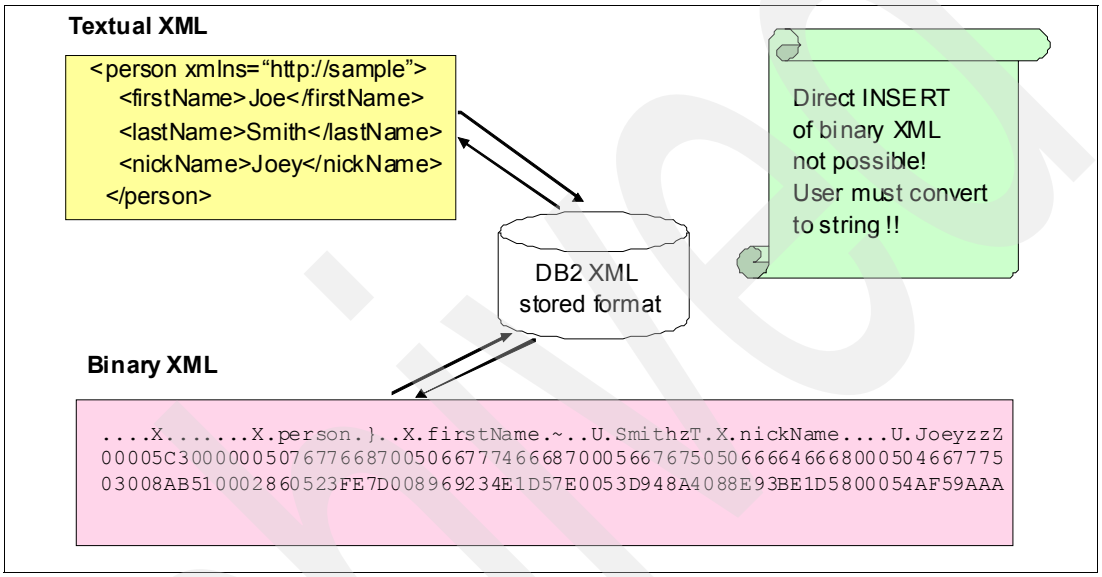


Figure 8-23 Binary XML is not the same as FOR BIT DATA

Binary XML is more efficient for various reasons, such as:

- Binary XML uses string IDs for names only (element names, attribute names, namespace prefixes, and URIs), not an arbitrary string. A string ID can represent some or all occurrences of the same text with an integer identifier.
- Binary XML uses a pre-tokenized format, and all values are prefixed with length. There is no need to look at every byte or to search for the end of element names for values.
- Binary XML on average is about 17% to 46% smaller in size, requiring less virtual storage.
- Binary XML saves DB2 9% to 30% CPU time and 8% to 50% elapsed time during insert.

Retrieval of data as binary XML is supported for remote access using DRDA and for use in JDBC, SQLJ, and ODBC applications. Binary XML is also supported by the UNLOAD and LOAD utilities.

Binary XML does not influence how XML data is stored in DB2. No parsing is needed and no z/OS XML System Services are invoked. However, DB2 DRDA zIIP redirect is not affected by the usage of binary XML.

Archive

The binary representation provides more efficient XML parsing. An XML value can be transformed into a binary XML value that represents the XML document using the following methods:

- In a JDBC or SQLJ application, you can transform an XML value by retrieving the XML column value into an java.sql.SQLXML object and then retrieving the data from the java.sql.SQLXML object as a binary data type, such as InputStream. You cannot get binary XML data out of SQL/XML type. It is implicitly used. JDBC 4.0 or later provides support for the java.sql.SQLXML data type.
- In an ODBC application, you can transform an XML value by binding the XML column to an application variable with the SQL_C_BINARYXML data type and retrieving the XML value into that application variable.
- You can also transform an XML value by running the UNLOAD utility and using one of the following field specifications for the XML output:

  ```
  CHAR BLOBF template-name BINARYXML
  VARCHAR BLOBF template-name BINARYXML
  XML BINARYXML
  ```

Similarly, you can transform a binary value that represents an XML document to an XML value in the following methods:

- In a JDBC or SQLJ application, you can transform a binary value by assigning the input value to an java.sql.SQLXML object and then inserting the data from the java.sql.SQLXML object into the XML column.
- ODBC provides a pass-through capability for binary data. Binary XML has to be generated by the application or retrieved from DB2 for inserting back.
- You can also transform a binary value by running the LOAD utility and using one of the following field specifications for the XML input:

  ```
  CHAR BLOBF BINARYXML
  VARCHAR BLOBF BINARYXML
  XML BINARYXML
  ```

Example 8-48 shows a JDBC example of using SQLXML type. The example shows how to get a DOM tree, or SAX source, or string from SQLXML type. An SQL/XML object can be read only once. Thus, the statements are mutually exclusive. Use one of them.

Example 8-48 Application using JDBC 4.0 (fetch XML as SQLXML type)

```
String sql = "SELECT xml_col from T1";
PreparedStatement pstmt = con.prepareStatement(sql);
ResultSet resultSet = pstmt.executeQuery();

// get the result XML as SQLXML
SQLXML sqlxml = resultSet.getSQLXML(column);
// column must be = 1

// get a DOMSource from SQLXML object
DOMSource domSource = sqlxml.getSource(DOMSource.class);
Document document = (Document) domSource.getNode();

//or: get a SAXSource from SQLXML object
SAXSource saxSource = sqlxml.getSource(SAXSource.class);
XMLReader xmlReader = saxSource.getXMLReader();
xmlReader.setContentHandler(myHandler);
xmlReader.parse(saxSource.getInputSource());
```

```
//or: get binaryStream or string from SQLXML object
InputStream binaryStream = sqlxml.getBinaryStream(); //Or:
String xmlString = sqlxml.getString();
```

Example 8-49 shows a JDBC example of using SQLXML type. The example shows how to insert and update XML using SQLXML.

Example 8-49 Application using JDBC 4.0 (insert and update XML using SQLXML)

```
String sql = "insert into T1 values(?)";
PreparedStatement pstmt = con.prepareStatement(sql);

SQLXML sqlxml = con.createSQLXML();
// create a SQLXML object from the DOM object
DOMResult domResult = sqlxml.setResult(DOMResult.class);
domResult.setNode(myNode);

// create a SQLXML object from a string
sqlxml.setString(xmlString);

// set that xml doucment as the input to parameter marker 1
pstmt.setSQLXML(1, sqlxml);

pstmt.executeUpdate();

Sqlxml.free();
```

Figure 8-24 shows an example of using binary XML in the UNLOAD and LOAD utilities.

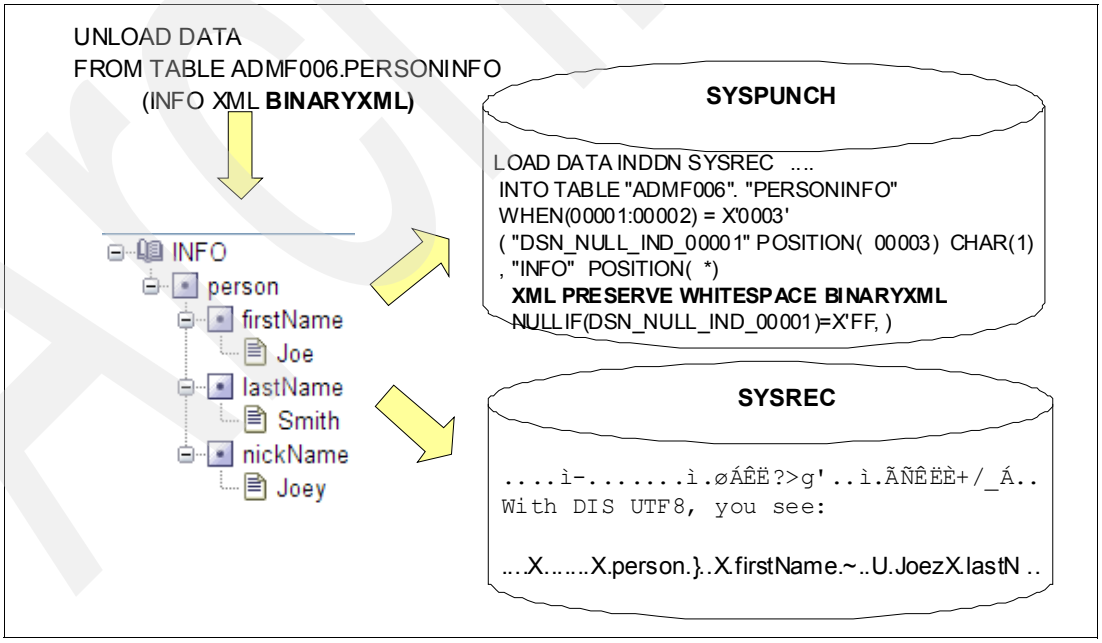


Figure 8-24 Binary XML in the UNLOAD and LOAD utilities

Archived

8.8 Support for XML date and time

In DB2 9, the xs:date, xs:time, and xs:dateTime data types support date and time data. However, they are not natively supported in XPath. XMLCAST and XMLTABLE functions can be invoked on values typed on these data types. You can cast XML xs:date, xs:time, and xs:dateTime data types to SQL DATE, TIME, and TIMESTAMP SQL data types, but you cannot cast SQL DATE, TIME, and TIMESTAMP data types to XML xs:date, xs:time, and xs:dateTime data types. There is no support for XML indexes for these data types.

When developing applications that deal with date or time in XML, keep in mind the following consideration:

- XML applications can model date and time data types only as character strings, which is imprecise when world times (time zones) are involved.
- You cannot do date or time arithmetic on XML data in DB2. Thus, an application developer needs to write application code to perform these functions.

The lack of support for XML indexes with a date or time stamp data type negatively affects the performance of applications that search on a date or time in XML documents.

DB2 10 includes support for XML date and time data types. In this section, we discuss these enhancements.

8.8.1 Enhancements for XML date and time support

DB2 10 supports date and time in XML data types and functions. It provides a time zone feature and arithmetic and comparison operators on date and time data types.

- In addition to the xs:string, xs:boolean, xs:decimal, xs:integer, xs:untypedAtomic data types, DB2 10 adds the following data types that are related to date and time operations:
  - xs:dateTime
  - xs:date
  - xs:time
  - xs:duration
  - xs:yearMonthDuration
  - xs:dayTimeDuration
- DB2 10 supports the following XML functions and operators that deal with date and time:
  - Comparison operators on duration, date, and time values
  - Component extraction functions on duration, date, and time
  - Arithmetic operators on durations
  - Time zone adjustment function on date and time values
  - Arithmetic operators on duration, date, and time
  - Context functions
- DB2 10 supports XMLCAST casts from SQL date and time to XML date and time.
- DB2 10 supports extends XML index creation and exploitation on date and time data types.

Archived

8.8.2 XML date and time support

Figure 8-25 shows the lexical representation for date and time support.

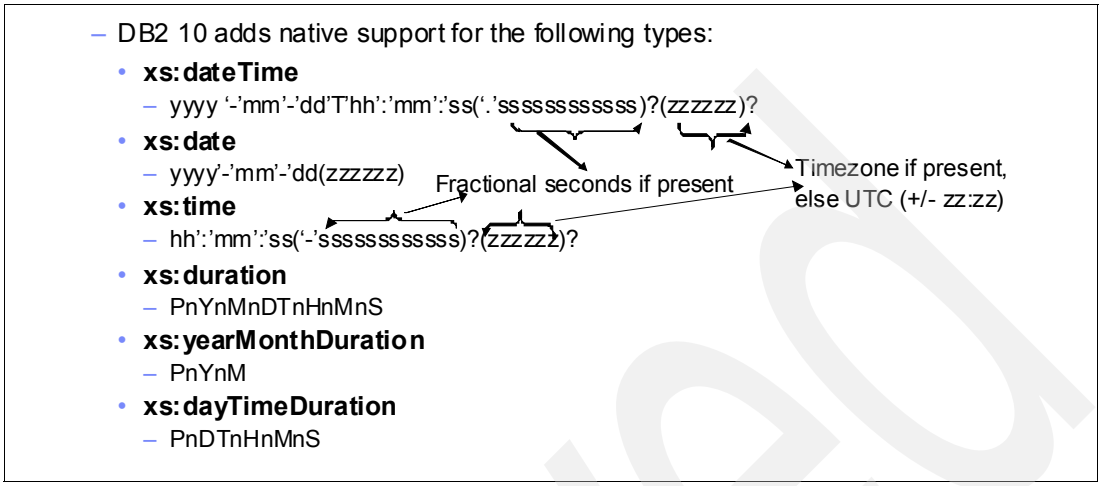


Figure 8-25 XML date and time support

The descriptions for the abbreviations for xs:dateTime, xs:date, and xs:time are as follows:

yyyy	A four-digit numeral that represents the year. The value cannot begin with a negative sign (-) or a plus sign (+). The number 0001 is the lexical representation of the year 1 of the Common Era (also known as 1 AD). The value cannot be 0000.
-	The dash (-) is a separator between parts of the date portion.
mm	A two-digit numeral that represents the month.
dd	A two-digit numeral that represents the day.
T	A separator to indicate that the time of day follows.
hh	A two-digit numeral (with leading zeros as required) that represents the hours. The value must be between -14 and +14, inclusive.
:	The colon (:) is a separator between parts of the time portion.
mm	A two-digit numeral that represents the minute.
ss	A two-digit numeral that represents the whole seconds.
.*sssssssssss*	Optional. If present, a 1-to-12 digit numeral that represents the fractional seconds.
zzzzzz	Optional. If present, represents the time zone. If a time zone is not specified, an implicit time zone of Coordinated Universal Time (UTC), also called *Greenwich Mean Time*, is used.

The lexical form for the time zone indicator is a string that includes one of the following forms:

- A positive (+) or negative (-) sign that is followed by *hh:mm*, where the following abbreviations are used:

hh	A two-digit numeral (with leading zeros as required) that represents the hours. The value must be between -14 and +14, inclusive.
mm	A two-digit numeral that represents the minutes. The value of the minutes property must be zero when the hours property is equal to 14.

+	The plus sign (+) indicates that the specified time instant is in a time zone that is ahead of the UTC time by *hh* hours and *mm* minutes.
-	The dash (-) indicates that the specified time instant is in a time zone that is behind UTC time by *hh* hours and *mm* minutes.

- The literal Z represents the time in UTC (Z represents Zulu time, which is equivalent to UTC). Specifying Z for the time zone is equivalent to specifying +00:00 or -00:00.

Here are a few examples:

- xs:dateTime

  The following form indicates noon on 10 October 2009, Eastern Standard Time in the United States:

  ```
  2009-10-10T12:00:00-05:00
  ```

  This time is expressed in UTC as 2009-10-10T17:00:00Z.

- xs:date

  The following form indicates 10 October 2009, Eastern Standard Time in the United States:

  ```
  2009-10-10-05:00
  ```

  This date is expressed in UTC as 2009-10-10T05:00:00Z

- xs:time

  The following form, which includes an optional time zone indicator, represents 1:20 p.m. Eastern Standard Time, which is five hours behind than UTC:

  ```
  13:20:00-05:00
  ```

The following list describes the abbreviations for duration, yearMonthDuration, and dayTimeDuration:

P	The duration designator.
*n*Y	*n* is an unsigned integer that represents the number of years.
*n*M	*n* is an unsigned integer that represents the number of months.
*n*D	*n* is an unsigned integer that represents the number of days.
T	The date and time separator.
*n*H	*n* is an unsigned integer that represents the number of hours.
*n*M	*n* is an unsigned integer that represents the number of minutes.
*n*S	*n* is an unsigned decimal that represents the number of seconds. If a decimal point appears, it must be followed by one to 12 digits that represent fractional seconds.

Here are a few examples:

- xs:duration

  The following form indicates a duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes:

  ```
  P1Y2M3DT10H30M
  ```

  The following form indicates a duration of negative 120 days:

  ```
  -P120D
  ```

- xs:yearMonthDuration

  The following form indicates a duration of 1 year and 2 months:

  ```
  P1Y2M
  ```

  The following form indicates a duration of negative 13 months:

  ```
  -P13M
  ```

- xs:dayTimeDuration

  The following form indicates a duration of 3 days, 10 hours, and 30 minutes:

  ```
  P3DT10H30M
  ```

  The following form indicates a duration of negative 120 days:

  ```
  -P120D
  ```

Figure 8-26 discusses comparison operators on XML duration, date, and time values of XMLQUERY.

For example,

```
SELECT XMLQUERY('xs:duration("P1Y") = xs:duration("P12M")')
FROM SYSIBM.SYSDUMMY1;
Result: <?xml version="1.0" encoding="IBM037"?>true

SELECT XMLQUERY('xs:duration("P1Y") = xs:duration("P365D")')
FROM SYSIBM.SYSDUMMY1;
Result: <?xml version="1.0" encoding="IBM037"?>false
```

- XPath allows to compare XML duration, date, and time values
- Result is xs:boolean (true/false)
- Possible comparisons:
  - xs:yearMonthDuration < (or >) xs:yearMonthDuration
  - xs:dayTimeDuration < (or >) xs:dayTimeDuration
  - duration-equal
    - Compare any durations to see if they are equal
  - xs:dateTime < (or > or =) xs:dateTime
  - xs:date < (or > or =) xs:date
  - xs:Time < (or > or =) xs:Time

Figure 8-26 XML date and time related comparison operators

Here are a few examples for comparison of durations:

- (xs:duration("P1Y") = xs:duration("P12M")) returns true.
- (xs:duration("PT24H") = xs:duration("P1D")) returns true.
- (xs:duration("P1Y") = xs:duration("P365D")) returns false.
- (xs:yearMonthDuration("P0Y") = xs:dayTimeDuration("P0D")) returns true.
- (xs:yearMonthDuration("P1Y") = xs:dayTimeDuration("P365D")) returns false.
- (xs:yearMonthDuration("P2Y") = xs:yearMonthDuration("P24M")) returns true.
- (xs:dayTimeDuration("P10D") = xs:dayTimeDuration("PT240H")) returns true.
- (xs:duration("P2Y0M0DT0H0M0S") = xs:yearMonthDuration("P24M")) returns true.
- (xs:duration("P0Y0M10D") = xs:dayTimeDuration("PT240H")) returns true.

Archived

Figure 8-27 shows examples for XML date and time comparison with SQL date.

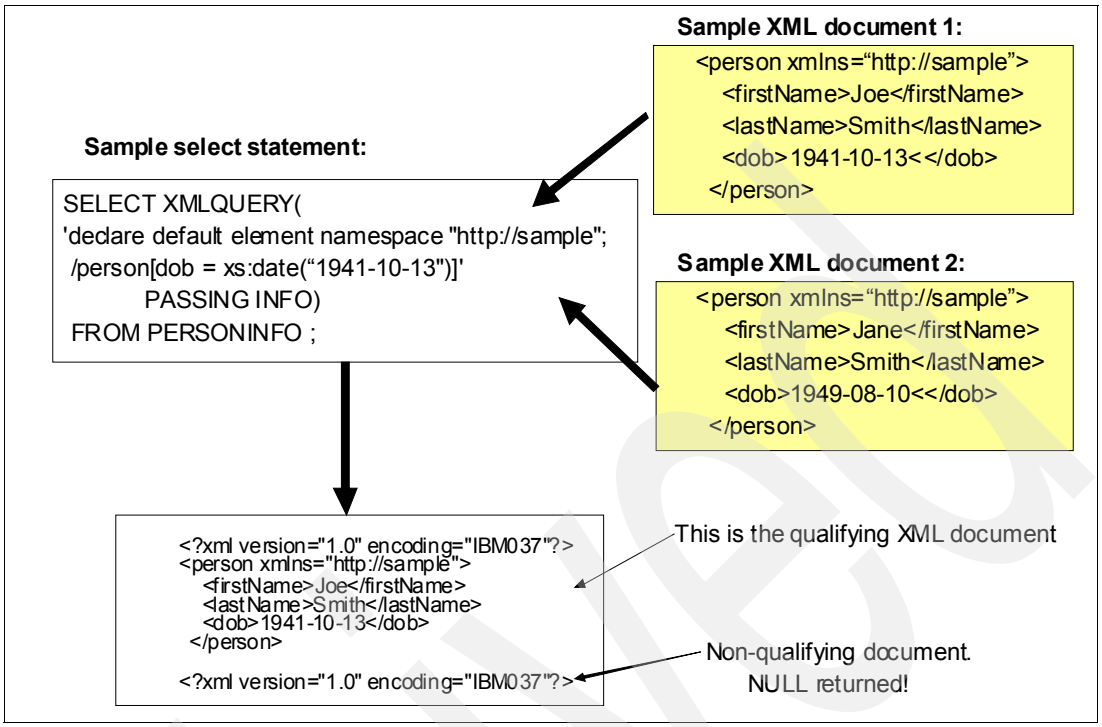


Figure 8-27 XML date and time comparison with SQL date

Figure 8-28 shows how the durations can be used in arithmetic operations.

For example:

```
SELECT XMLQUERY ('xs:yearMonthDuration("P2Y11M") div 1.5')
FROM SYSIBM.SYSDUMMY1;
Result: <?xml version="1.0" encoding="IBM037"?>P1Y11M
```

- Durations may be used in arithmetic operations
- The 10 allowed operations are:
  - **A + B** (returns xs:yearMonthDuration)
  - **A – B** (returns xs:yearMonthDuration)
  - **A * B** (returns xs:yearMonthDuration)
  - **A div C** (returns xs:yearMonthDuration)
  - **A div B** (returns xs:decimal)
  - **D + E** (returns xs:dayTimeDuration)
  - **D - E** (returns xs:dayTimeDuration)
  - **D * E** (returns xs:dayTimeDuration)
  - **D div F** (returns xs:dayTimeDuration)
  - **D div E** (returns xs:decimal)
- A and B are type xs:yearMonthDuration, and C is type numeric
- D and E are type: xs:dayTimeDuration, and F is numeric
- Example:

  xs:yearMonthDuration("P2Y11M") div 1.5)
  Result: 35/1.5=23.33333 => **P1Y11M**

Figure 8-28 Arithmetic operations on XML duration

Here are a few examples for arithmetic operations on XML duration

- `(xs:yearMonthDuration("P2Y11M") + xs:yearMonthDuration("P3Y3M"))` returns an xs:yearMonthDuration value corresponding to 6 years and 2 months: P6Y2M.
- `(xs:yearMonthDuration("P2Y11M") - xs:yearMonthDuration("P3Y3M"))` returns an xs:yearMonthDuration value corresponding to negative 4 months: -P4M.
- `(xs:yearMonthDuration("P2Y11M") * 2.3)` returns an xs:yearMonthDuration value corresponding to 6 years and 9 months: P6Y9M.
- `(xs:yearMonthDuration("P2Y11M") div 1.5)` returns an xs:yearMonthDuration value corresponding to 1 year and 11months: P1Y11M.
- `(xs:yearMonthDuration("P3Y4M") div xs:yearMonthDuration("P1Y4M"))` returns 2.5.
- `(xs:dayTimeDuration("P2DT12H5M") + xs:dayTimeDuration("P5DT12H"))` returns an xs:dayTimeDuration value corresponding to 8 days and 5 minutes: P8DT5M.
- `(xs:dayTimeDuration("P2DT12H") - xs:dayTimeDuration("P1DT10H30M"))` returns an xs:dayTimeDuration value corresponding to 1 day, 1 hour, and 30minutes: P1DT1H30M.
- `(xs:dayTimeDuration("PT2H10M") * 2.1)` returns an xs:dayTimeDuration value corresponding to 4 hours and 33 minutes: PT4H33M.
- `(xs:dayTimeDuration("P1DT2H30M10.5S") div 1.5)` returns an xs:dayTimeDuration value corresponding to 17 hours, 40 minutes, and 7 seconds:PT17H40M7S
- `(xs:dayTimeDuration("P2DT53M11S") div xs:dayTimeDuration("P1DT10H"))` returns 1.4378349.

Figure 8-29 discusses arithmetic operations on XML duration, date, and time.

- Date and time AND duration may be used in arithmetic operations
- The 13 allowed operations are:
  - **A - B** (returns xs:dayTimeDuration) A and B are type xs:date
  - **A – B** (returns xs:dayTimeDuration) A and B are type xs:time
  - **A – B** (returns xs:dayTimeDuration) A and B are type xs:dateTime
  - **A + B** (returns xs:dateTime) A is type xs:yearMonthDuration and B is type xs:dateTime
  - **A + B** (returns xs:dateTime) A is type xs:dayTimeDuration and B is type xs:dateTime
  - **A – B** (returns xs:dateTime) A is type xs:yearMonthDuration and B is type xs:dateTime
  - **A – B** (returns xs:dateTime) A is type xs:dayTimeDuration and B is type xs:dateTime
  - **A + B** (returns xs:date) A is type xs:yearMonthDuration and B is type xs:date
  - **A + B** (returns xs:date) A is type xs:dayTimeDuration and B is type xs:date
  - **A – B** (returns xs:date) A is type xs:yearMonthDuration and B is type xs:date
  - **A – B** (returns xs:date) A is type xs:dayTimeDuration and B is type xs:date
  - **A + B** (returns xs:time) A is type xs:dayTimeDuration and B is type xs:time
  - **A – B** (returns xs:time) A is type xs:dayTimeDuration and B is type xs:time

Figure 8-29 Arithmetic operations on XML duration, date, and time

Here are a few examples for arithmetic operations on XML duration, date, and time:

- `(xs:dateTime("2000-10-30T06:12:00") - xs:dateTime("1999-11-28T09:00:00Z"))` returns an xs:dayTimeDuration value of P336DT21H12M.
- `(xs:date("2000-10-30") - xs:date("1999-11-28"))` returns an xs:dayTimeDuration value of P337D.
- `(xs:time("24:00:00") - xs:time("23:59:59"))` returns an xs:dayTimeDuration value corresponding to "-PT23H59M59S".

- (xs:dateTime("2000-10-30T11:12:00") + xs:yearMonthDuration("P1Y2M")) returns an xs:dateTime value corresponding to the lexical representation "2001-12-30T11:12:00".
- (xs:dateTime("2000-10-30T11:12:00") + xs:dayTimeDuration("P3DT1H15M")) returns an xs:dateTime value corresponding to the lexical representation "2000-11-02T12:27:00".
- (xs:dateTime("2000-10-30T11:12:00") - xs:yearMonthDuration("P1Y2M")) returns an xs:dateTime value corresponding to the lexical representation "1999-08-30T11:12:00".
- (xs:dateTime("2000-10-30T11:12:00") - xs:dayTimeDuration("P3DT1H15M")) returns an xs:dateTime value corresponding to the lexical representation "2000-10-27T09:57:00".
- (xs:date("2000-10-30") + xs:yearMonthDuration("P1Y2M")) returns an xs:date corresponding to 30 December 2001: 2001-12-30.
- (xs:date("2004-10-30Z") + xs:dayTimeDuration("P2DT2H30M0S")) returns an xs:date November 1, 2004-11-01Z. The starting instant of the first argument is the xs:dateTime value {2004, 10, 30, 0, 0, 0, PT0S}. Adding the second argument to this, gives the xs:dateTime value {2004, 11, 1, 2, 30, 0, PT0S}. The time components are then discarded.
- (xs:date("2000-10-30") - xs:yearMonthDuration("P1Y2M")) returns an xs:date 30 August 1999: 1999-08-30.
- (xs:date("2000-10-30") - xs:dayTimeDuration("P3DT1H15M")) returns an xs:date 26 October 2000: 2000-10-26.
- (xs:time("11:12:00") + xs:dayTimeDuration("P3DT1H15M")) returns an xs:time value corresponding to the lexical representation "12:27:00".
- (xs:time("11:12:00") - xs:dayTimeDuration("P3DT1H15M")) returns an xs:time value corresponding to the lexical representation "09:57:00".

Figure 8-30 lists the date and time related XPath functions.

- 24 new functions based on date, time, and duration
  - fn:years-from duration, fn:months-from-duration, fn:days-from-duration, fn:hours-from-duration, fn:minutes-from-duration, fn:seconds-from-duration
  - fn:year-from-dateTime, fn:month-from-dateTime, fn:day-from-dateTime, fn:hours-from-dateTime, fn:minutes-from-dateTime, fn:seconds-from-dateTime, fn:timezone-from-dateTime
  - fn:year-from-date, fn:month-from-date, fn:day-from-date, fn:timezone-from-date
  - fn:hours-from-time, fn:minutes-from-time, fn:seconds-from-time, fn:timezone-from-time
  - fn:current-date, fn:current-dateTime, fn:current-time

Figure 8-30 Date and time related XPath functions

The fn:current-date, fn:current-dateTime, and fn:current-time functions are context functions.

The fn:current-date function returns the current date in the local time zone. The returned value is an xs:date value that is the current date. The time zone component of the returned value is the local time zone.

For example, the following function returns the current date:

```
fn:current-date()
```

If this function were invoked on 02 September 2010, in Pacific Standard Time, the returned value would be `2010-09-02-08:00`.

The fn:current-dateTime function returns the current date and time in the local time zone.

The following function returns an xs:dateTime value that is the current date and time. The time zone component of the returned value is the local time zone. The precision for fractions of seconds is 12.

```
fn:current-dateTime()
```

If this function were invoked on 02 September 2010 at 6:25 in Toronto (time zone -PT5H), the returned value would be 2010-09-02T06:25:30.384724902312-05:00.

The fn:current-time function returns the current time in the local time zone.

The following function returns an xs:time value that is the current time. The time zone component of the returned value is the local time zone. The precision for fractions of seconds is 12.

The following function returns the current time:

```
fn:current-time()
```

If this function were invoked at 6:31 Pacific Standard Time (-08:00), the returned value would be 06:31:35.519003948231-08:00.

Figure 8-31 shows examples of date and time related XPath functions.

- fn:years-from duration (as xs:integer) (result may be negative)
  - fn:years-from duration(xs:yearMonthDuration(“P20Y15M”)
  - result 21
- fn:month-from-dateTime (as xs:integer)
  - fn:month-from-dateTime(xs:dateTime(“2010-02-22T13:20:00-05:00”))
  - result 2
- fn:timezone-from-dateTime (as xs:dayTimeDuration)
  - fn:timezone-from-dateTime(xs:dateTime(“2010-02-22T13:20:00-**05:00**”))
  - result is xs:dayTimeDuration with value –PT5H
- fn:seconds-from-time (as xs:decimal)
  - fn:seconds-from-time(xs:time(“13:20:10.5”))
  - result 10.5

Figure 8-31 Date and time related XPath functions examples

For example:

```
SELECT XMLQUERY('fn:month-from-dateTime(
                 xs:dateTime("2010-02-22T13:20:00-05:00"))')
FROM SYSIBM.SYSDUMMY1 ;
```

Result: `<?xml version="1.0" encoding="IBM037"?>2`

Figure 8-32 discusses time zone adjustment functions on date and time.

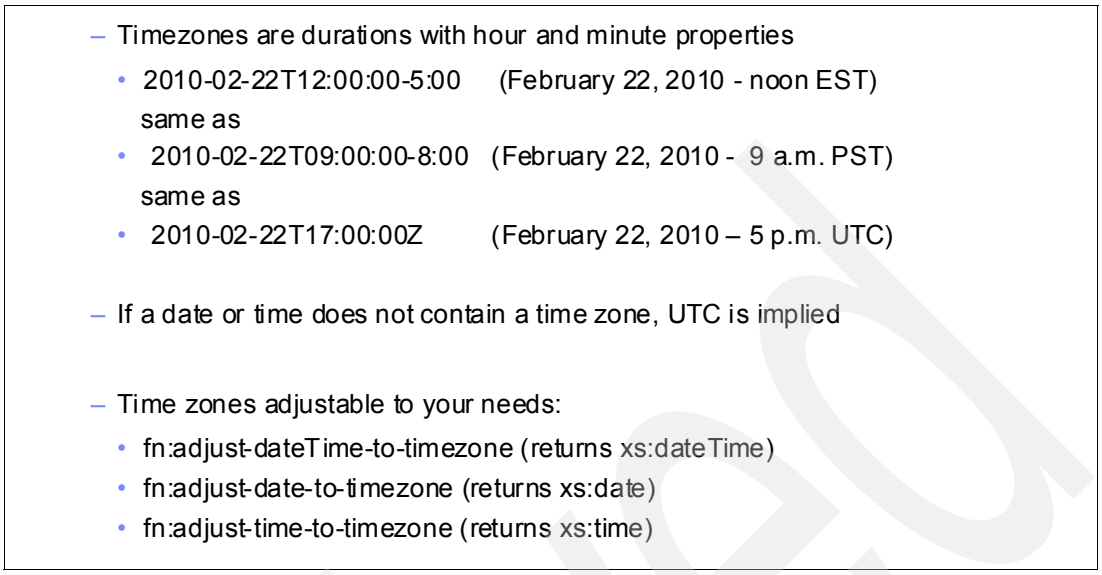


Figure 8-32 Time zone adjustment functions on date and time (1 of 2)

The function `adjust-dateTime-to-timezone` adjusts an xs:dateTime value to a specific time zone or to no time zone at all.

The function `adjust-date-to-timezone` adjusts an xs:date value to a specific time zone or to no time zone at all.

The function `adjust-time-to-timezone` adjusts an xs:time value to a specific time zone or to no time zone at all.

Archived

Figure 8-33 shows some examples.

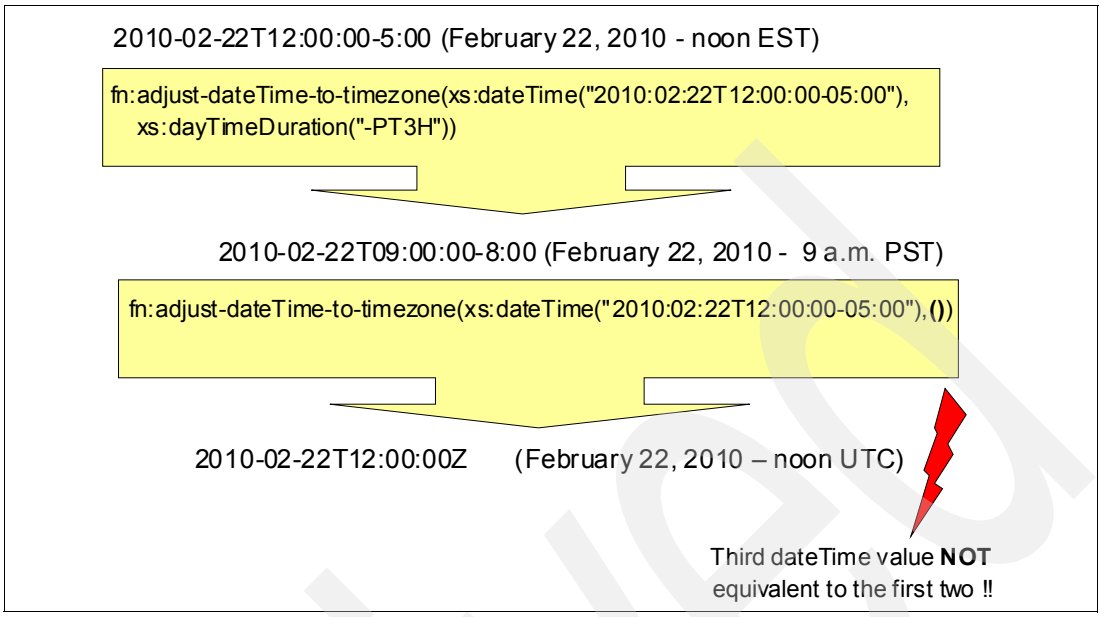


Figure 8-33 Time zone adjustment functions on date and time (2 of 2)

The time zone adjustment functions take a dayTimeDuration for the time zone as the second argument. If it is an empty sequence, it strips the time zone. See Example 8-50.

Example 8-50 Samples of the time zone adjustment functions

```
SELECT XMLQUERY('
fn:adjust-dateTime-to-timezone(xs:dateTime("2010-02-22T12:00:00-05:00"),
    xs:dayTimeDuration("-PT3H"))')
FROM SYSIBM.SYSDUMMY1;
Result: <?xml version="1.0" encoding="IBM037"?>2010-02-22T14:00:00-03:00

SELECT XMLQUERY('
fn:adjust-dateTime-to-timezone(xs:dateTime("2010-02-22T09:00:00-08:00"),
    xs:dayTimeDuration("-PT3H"))')
FROM SYSIBM.SYSDUMMY1;
Result: <?xml version="1.0" encoding="IBM037"?>2010-02-22T14:00:00-03:00

SELECT XMLQUERY('
fn:adjust-dateTime-to-timezone(xs:dateTime("2010-02-22T12:00:00-05:00"), ())')
FROM SYSIBM.SYSDUMMY1;
Result: <?xml version="1.0" encoding="IBM037"?>2010-02-22T12:00:00
```

Archive

Figure 8-34 summarizes XML index improvements for date and timestamp.

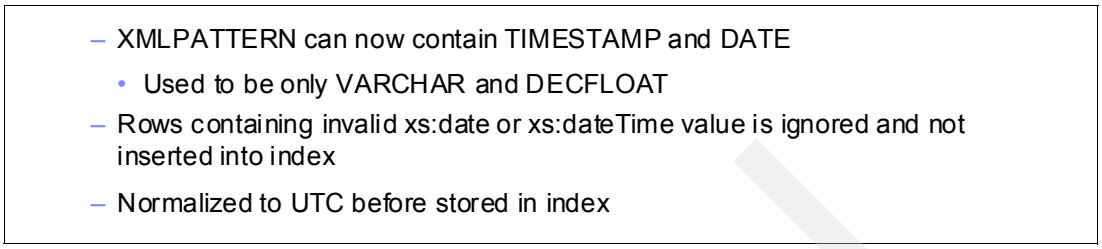


Figure 8-34 XML index improvement for date and time stamp

Here are CREATE INDEX examples:

- Create an index for the date representation of the shipDate element in XML documents in the PORDER column of the PURCHASEORDER table.

```
CREATE INDEX PO_XMLIDX1 ON PURCHASEORDER (PORDER)
GENERATE KEY USING XMLPATTERN '//items/shipDate'
AS SQL DATE
```

- The following query includes a range predicate on a timestamp type. It retrieves documents from the PORDER column of the PURCHASEORDER table for items that have been shipped.

```
SELECT PORDER FROM PURCHASEORDER T1
WHERE XMLEXISTS('$po//item[shipDate < $timestamp]'
PASSING PORDER AS "po", CURRENT TIMESTAMP AS "timestamp")
```

  To be compatible with this query, the XML index needs to include the shipDate nodes among the indexed nodes, and needs to store values in the index as a TIMESTAMP type. The precision of the TIMESTAMP column must be implicitly or explicitly set to 12.

  The query can use the following XML index:

```
CREATE INDEX PORDERINDEX ON PURCHASEORDER (PORDER)
GENERATE KEY USING XMLPATTERN '//item/shipDate' AS SQL TIMESTAMP(12)
```

8.9 XML in native SQL stored procedures and UDFs

DB2 9 does not support the native XML data type in native stored procedures and user defined functions. Applications cannot pass XML values between statements in native SQL procedures. To bypass this limitation, DB2 9 applications must use string or LOB SQL variables, which involves redundant data type conversion and expensive XML serialization and parsing. The lack of support for XML arguments in stored procedures and user-defined functions limits the user's ability to use XML features.

DB2 family compatibility and application portability compatibility and application portability are affected.

DB2 10 introduces the following enhancements to remove the functional limitations and provide DB2 family compatibility and application portability:

- Use the XML data type for IN, OUT, and INOUT parameters
- Use the XML data type for SQL variables inside the procedure and function logic

DB2 10 enhancements are only for native SQL procedures and for SQL user-defined functions (both scalar and table functions).

8.9.1 Enhancements to native SQL stored procedures

In the CREATE PROCEDURE statement for a native SQL procedure, XML can now be specified as the data type of a parameter.

Figure 8-35 shows an example of coding an XML parameter in a native SQL stored procedure.

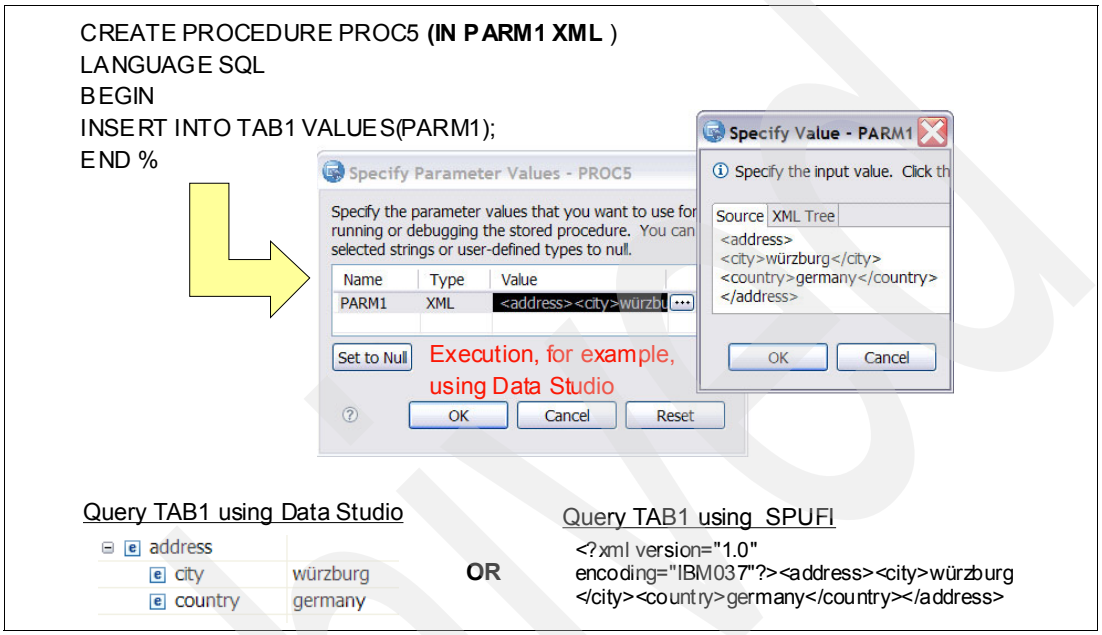


Figure 8-35 Native SQL stored procedure using XML parameter

The CREATE PROCEDURE syntax is extended to allow the XML data type to be specified for parameters. The input parameter PARM1 passes an XML document for being inserted into table TAB1 which has one XML column defined. You can verify whether the XML document is inserted properly by querying the table TAB1 and retrieving the XML column using Data Studio or SPUFI.

Archive

Figure 8-36 shows an example of coding an XML variable in a native SQL stored procedure.

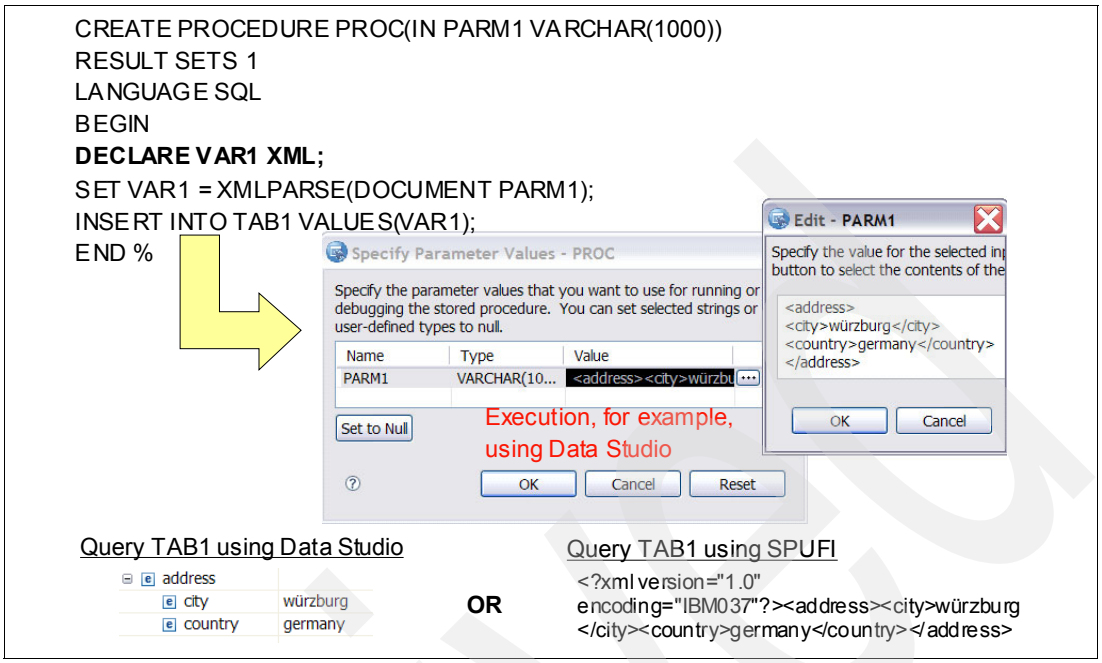


Figure 8-36 Native a SQL stored procedure using XML variable

The DECLARE VARIABLE statement defines a variable's data type in SQL PL. The syntax of the DECLARE VARIABLE statement has been extended to allow for XML to be specified as a data type. In the example in Figure 8-36, the SQL variable VAR1 is declared as an XML variable to hold the XML document. Notice that the XML document is passed to the stored procedure using the input parameter PARM1 which is not an XML data type. DB2 10 allows assignment from string to XML.

Optionally, the XML document can be parsed using the XLPARSE function before it is stored in VAR1 and then inserted into table TAB1. As before, you can verify if the XML document is properly inserted by querying the table TAB1 and retrieving the XML column either by using Data Studio or SPUFI.

8.9.2 Enhancements to user defined SQL scalar and table functions

In the CREATE FUNCTION statement for a scalar user defined function, XML can now be specified as the data type of a parameter. The DECLARE VARIABLE statement can specify XML data type.

Example 8-51 shows an example of coding an XML parameter in a user-defined SQL scalar function.

Example 8-51 User-defined SQL scalar function using XML variable

```
CREATE FUNCTION CANOrder  (BOOKORDER XML, USTOCANRATE  DOUBLE)
  RETURNS XML
  DETERMINISTIC
  NO EXTERNAL ACTION
  CONTAINS SQL
BEGIN ATOMIC
 DECLARE USPrice DECIMAL (15,2);
 DECLARE CANPrice DECIMAL(15,2);
```

```
 DECLARE OrderInCAN  XML;
 SET USPrice = XMLCAST(XMLQUERY('/bookorder/USprice' PASSING BOOKORDER) AS
  DECIMAL( 15,2);
 SET CANPrice = XMLCAST(XMLQUERY('/bookorder/CANprice' PASSING BOOKORDER) AS
  DECIMAL( 15,2);
 IF CANPrice is NULL or CANPrice <=0 THEN
  IF USPrice > 0 THEN
    SET CANPrice = USPrice * USTOCANRATE;
  ELSE
   SET CANPrice = 0;
 END IF;
 SET OrderInCAN =
 XMLDOCUMENT(
   XMLELEMENT(NAME "bookorder",
   XMLQUERY('/bookorder/bookname' PASSING BOOKORDER),
   XMLELEMENT(NAME "CANPrice", CANPrice))
  );
 RETURN OrderInCAN;
END %
```

The SQL function performs the following operations:

1. Declares an XML variable named OrderInCAN, which holds the order with prices in Canadian dollars that is returned to the caller.
2. Retrieves the U.S. price from the input document, which is in the BOOKORDER parameter.
3. Looks for a Canadian price in the input document. If the document contains no Canadian prices, the XMLCAST function on the XMLQUERY function returns NULL.
4. Builds the output document, whose top-level element is bookorder, by concatenating the bookname element from the original order with a CANprice element, which contains the calculated price in Canadian dollars.

Suppose that an input document looks as follows:

```
<bookorder>
<bookname>TINTIN</bookname>
<USprice>100.00</USprice>
</bookorder>
```

If the exchange rate is 0.9808 Canadian dollars for one U.S. dollar, the output document looks as follows:

```
<bookorder><bookname>TINTIN</bookname><CANprice>98.1</CANprice></bookorder>
```

Example 8-52 demonstrates the use of XML parameters in a SQL table function. This function takes three parameters as input:

- An XML document that contains order information
- A maximum price for the order
- The title of the book that is ordered

The function returns a table that contains an XML column with receipts that are generated from all of the input parameters, and a BIGINT column that contains the order IDs that are retrieved from the input parameter that contains the order information document.

Archived

Example 8-52 User-defined SQL table function using XML variable

```
CREATE FUNCTION ORDERTABLE
  (ORDERDOC XML, PRICE DECIMAL(15,2), BOOKTITLE VARCHAR(50))
  RETURNS TABLE (RECEIPT XML, ORDERID BIGINT)
  LANGUAGE SQL
  SPECIFIC ORDERTABLE
  NOT DETERMINISTIC
  READS SQL DATA
 RETURN
  SELECT ORDER.ID, ORDER.RECEIPT
  FROM XMLTABLE(XMLNAMESPACES(DEFAULT 'http://posample.org'),
   '/orderdoc/bookorder[USprice < $A and bookname = $B']'
  PASSING ORDERDOC, PRICE as A, BOOKTITLE as B
  COLUMNS
    ID BIGINT PATH '../@OrderID',
    RECEIPT XML PATH '.')
  AS ORDER;
```

The SQL table function uses the XMLTABLE function to generate the result table for the table that is returned by the function. The XMLTABLE function generates a row for each ORDERDOC input document in which the title matches the book title in the BOOKTITLE input parameter, and the price is less than the value in the PRICE input parameter. The columns of the returned table are the Receipt node of the ORDERDOC input document, and the OrderID element from the ORDERDOC input document.

Suppose that the input parameters have the values PRICE: 200, BOOKTITLE: TINTIN, ORDERDOC:

```
<orderdoc xmlns="http://posample.org" OrderID="5001">
 <name>Jim Noodle</name>
 <addr country="Canada">
  <street>25 EastCreek</street>
  <city>Markham</city>
  <prov-state>Ontario</prov-state>
  <pcode-zip>N9C-3T6</pcode-zip>
 </addr>
 <phone type="work">905-555-7258</phone>
 <bookorder>
  <bookname>TINTIN</bookname>
  <USprice>100.00</USprice>
 </bookorder>
</orderdoc>
```

Table 8-8 shows the returned table.

Table 8-8 Return table

ID	RECEIPT
5001	?xml version="1.0" encoding="IBM037"?> <bookorder> <bookname>TINTIN</bookname> <USprice>100.00</USprice> </bookorder>

8.9.3 Decompose to multiple tables with a native SQL procedure

Figure 8-37 illustrates how a SQL PL stored procedure can be used to decompose an XML document into multiple tables.

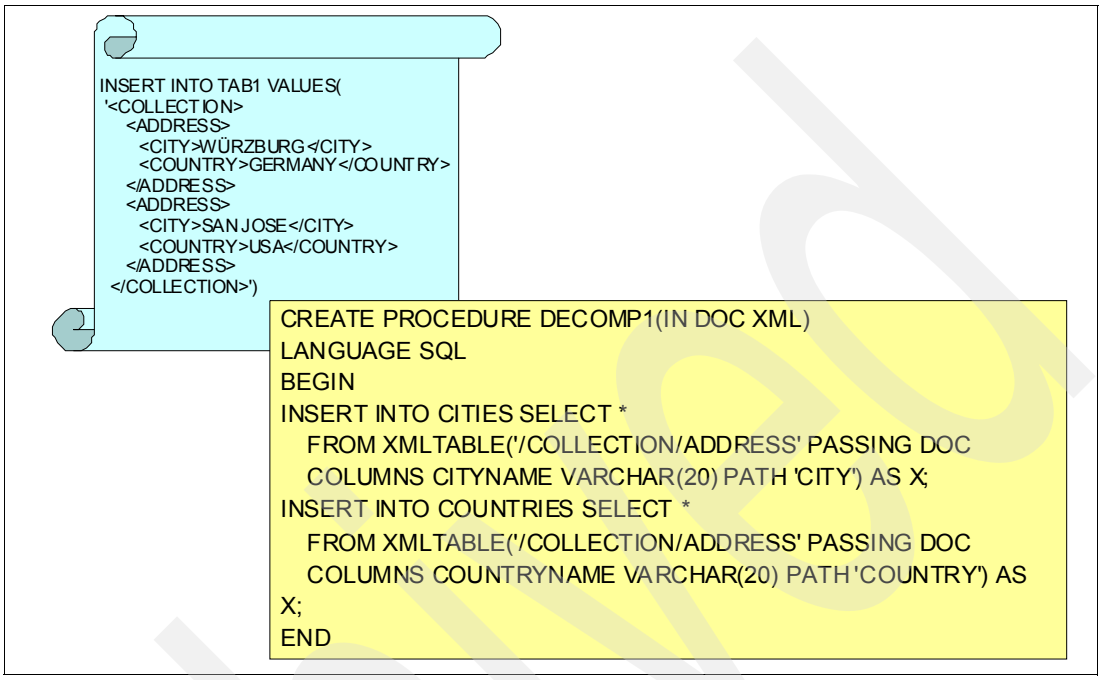


Figure 8-37 Decompose to multiple tables with a native SQL procedure

With XML SQL vars, you can parse once and use it multiple times for decomposition. If the XML parameter type is used and the caller is from JDBC, you can have XML parsed into binary XML in the client.

Archive

Figure 8-38 shows how the XML decomposition to multiple tables can be done in DB2 9.

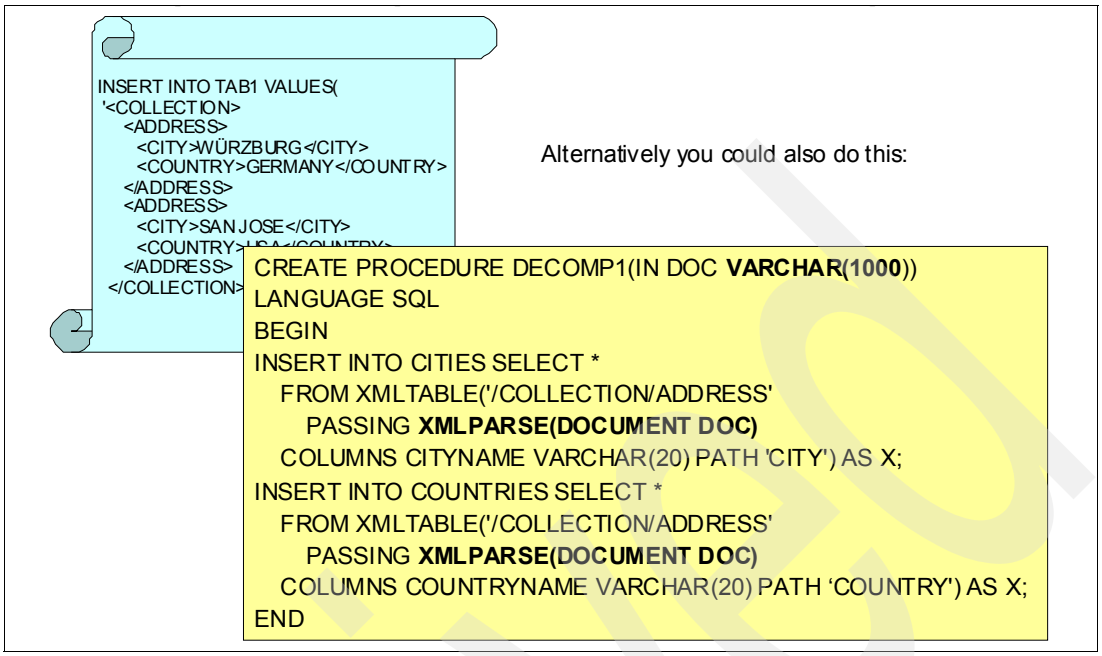


Figure 8-38 Decompose to multiple tables with a native SQL procedure (DB2 9)

Because you cannot specify an XML argument, the example shows the use of VARCHAR. This results in the XML document being parsed multiple times. Thus, if the XML document has to be decomposed into 10 tables, the document is parsed 10 times.

8.10 Support for DEFINE NO for LOBs and XML

Many DB2 installations use software packages and install a suite of applications that create table spaces explicitly or implicitly with the DEFINE NO option. When the DEFINE NO option is used data sets backing the table spaces are not created until a table in that table space is actually used. Thus, installations that use only a subset of modules from a full suite of applications do not experience the overhead of unnecessary data set creation. However, in DB2 9 the DEFINE NO option does not have effect when LOB and XML table spaces are created. Therefore, installations that install a subset of modules from a suite of applications still experience sub-optimal application installation time due to the overhead of creating LOB and XML table spaces and index spaces that are not even used by the subset of modules of interest to the installations. Further, because unused objects are not called out as such, time and resources are often spent on managing and backing up such unused LOB and XML table spaces and index spaces.

The IMPDSDEF subsystem parameter specifies whether DB2 is to define the underlying data set for an implicitly created table space that resides in an implicit database. This DSNZPARM corresponds to the field DEFINE DATA SETS on the installation panel DSNTIP7 INSTALL DB2 - SIZES PANEL 2. Acceptable values are YES or NO. The default is YES. The default value of YES means that the data set is defined when the table space is implicitly created. The value of DEFINE DATA SETS applies only to implicitly created base table spaces. It is not used for implicitly created LOB or XML table spaces.

DB2 10 introduces a solution to allow installations to use the DEFINE NO option in the CREATE TABLESPACE and CREATE INDEX statements or the system parameter,

IMPDSDEF, to defer the creation of underlying VSAM data sets for LOB and XML table spaces and their dependent index spaces when creating tables with LOB or XML columns. Additionally, optimization has been added to inherit the DEFINE NO attribute of the base table space for dependent objects that are implicitly created or explicitly created without the DEFINE option specified.

8.10.1 IMPDSDEF subsystem parameter

The IMPDSDEF subsystem parameter specifies whether DB2 defines the underlying data set for certain implicitly created table spaces and indexes. This parameter applies to:

- Base table spaces
- Index spaces of indexes that are implicitly created on base tables
- Implicitly created LOB or XML table spaces
- Auxiliary indexes
- NODEID indexes
- DOCID indexes

Acceptable values are YES or NO. The default is YES.

YES	Underlying data sets are defined when the table space or index space is created.
NO	Underlying data sets are defined when data is first inserted into the table or index.

8.10.2 Usage reference

To invoke this function, specify DEFINE NO in CREATE TABLESPACE and CREATE INDEX statements for explicitly created LOBs, auxiliary indexes, and XML value indexes. Alternatively, set the IMPDSDEF subsystem parameter to NO for implicitly created LOB or XML table spaces and their dependent indexes. When the base table space has undefined data sets through DEFINE NO or IMPDSDEF=NO, DB2 inherits the undefined state of the base table space to dependent objects for the following cases:

- For explicitly created dependent objects, if DEFINE YES/NO is specified the DEFINE attribute is honored by DB2.
- For explicitly created dependent objects (auxiliary index, XML index, base table index), if the DEFINE attribute is not specified, DB2 inherits the DEFINE attribute from the base table space. The exception is explicitly created LOB table spaces without DEFINE specified. With an explicitly created LOB table space there is no correlation with the base table space until the auxiliary table is created so DEFINE attribute inheritance does not occur.
- For implicitly created dependent objects (base table indexes, and all LOB or XML objects), the DEFINE NO attribute of the base table space is inherited by these dependent objects. Otherwise, the IMPDFDEF subsystem parameter setting is honored for these implicitly created objects.

DBAs and application package providers should consider using the DEFINE NO or IMPDSDEF=NO subsystem parameter if the DDL performance is critical or if the DASD resources are limited. These DEFINE NO options provide better management reliefs on the z/OS DD limits on data set allocation and data usability by deferring the VSAM DEFINE/OPEN until the first insert into the object.

The SPACE column in SYSIBM.SYSTABLEPART catalog table with a value of -1 is used for LOB and XML table spaces to indicate that the table space is undefined. A non-negative

value indicates that the data sets for the table space are defined with the underlying VSAM data sets allocated.

The SPACE column in SYSIBM.SYSINDEXPART catalog table with a value of -1 is used for indexes on LOB and XML table spaces to indicate that the index is undefined. A non-negative value indicates that the data sets for the index space are defined with the underlying VSAM data sets allocated.

DB2 sets the value of SPACE column to 0 after it creates an underlying VSAM data set upon the first insert or LOAD.

The inheritance rules also apply to situations where the LOB or XML columns are added with ALTER statement to an existing table and when new partitions are explicitly added to a partition-by-growth or partition-by-range table space. However, when a partition-by-growth table space grows and implicitly creates a new LOB table space for the new base partition, DB2 uses the IMPDSDEF subsystem parameter to set the DEFINE attribute of the new implicitly created LOB table space.

8.11 LOB and XML data streaming

In DB2 9, the processing of LOBs in a distributed environment with Java Universal Driver on the client side has been optimized for the retrieval of larger amounts of data. This dynamic data format is available only for the Java Common Connectivity (JCC) T4 driver (Type 4 Connectivity). The call-level interface (CLI) of DB2 for Linux, UNIX, and Windows also has this client-side optimization. Many applications effectively use locators to retrieve LOB data regardless of the size of the data that is being retrieved. This mechanism incurs a separate network flow to get the length of the data to be returned, so that the requester can determine the proper offset and length for SUBSTR operations on the data to avoid any unnecessary blank padding of the value. For small LOB and XML data, returning the value directly instead of using a locator is more efficient.

For these reasons, LOB and XML data retrieval in DB2 9 has been enhanced so that it is more effective for small and medium size objects. It is also still efficient in its use of locators to retrieve large amounts of data. This function is known as *progressive streaming*.

DB2 10 for z/OS offers additional performance improvements by extending the support for LOB and XML streaming, avoiding LOB and XML materialization between the distributed data facility (DDF) and DBM1 address space.

For more information, see also 13.11.3, “Streaming LOBs and XML between DDF and DBM1” on page 568.

Archive

9

Connectivity and administration routines

Mainframe systems are adept at accommodating software that has been around for decades, such as IMS, DB2, and CICS. However, they also host web applications that are built in Java and can accommodate the latest business requirements.

DB2 9 and DB2 10 improve and generalize universal drivers for accessing data on any local or remote server without the need of a gateway. In addition, DB2 10 for z/OS provides a number of enhancements to improve the availability of distributed applications, including online CDB changes and online changes to location alias names in addition to connectivity improvements.

In this chapter, we discuss the following topics:

- DDF availability
- Monitoring and controlling connections and threads at the server
- JDBC Type 2 driver performance enhancements
- High performance DBAT
- Use of RELEASE(DEALLOCATE) in Java applications
- Support for 64-bit ODBC driver
- DRDA support of Unicode encoding for system code pages
- DB2-supplied stored procedures

© Copyright IBM Corp. 2010. All rights reserved.

9.1 DDF availability

With DB2 9, making almost any configuration change to the distributed data facility (DDF) can be disruptive. DB2 10 introduces changes to help improve DDF availability by eliminating requirements for recycling DDF and to provide better correlation with remote systems. Most of the changes involve the ability to perform configuration changes without recycling the DDF. These enhancements provide higher DDF availability by providing support to modify the communication database and to modify distributed location aliases and associated IP addresses without disrupting existing connections. These functions are a late addition and require APARs PM26781 (PTF UK63818) and PM26480 (PTF UK63820).

In this section, we discuss the following topics:

- Online communications database changes
- Online DDF location alias name changes
- Domain name is now optional
- Acceptable period for honoring cancel requests
- Sysplex balancing using SYSIBM.IPLIST
- Message-based correlation with remote IPv6 clients

9.1.1 Online communications database changes

The communications database (CDB) is a set of updatable DB2 catalog tables that contain DDF information, such as a remote server's IP address, DRDA port, and other such information. A DDF requester uses this information to establish outbound connections to a server.

When a CDB table, specifically LOCATIONS, IPNAMES, or IPLIST, is updated, the DDF uses the updated values only when it is recycled or when it has not yet connected to that server:

- If DDF has not yet started communicating to a particular location, IPNAMES and LOCATIONS take effect when DDF attempts to communicate with that location.
- If DDF has already started communication, changes to IPNAMES and LOCATIONS take effect the next time DDF is started.

In DB2 10, updates to IPNAMES, LOCATIONS, and IPLIST take effect for a new remote connection that is requested by a new or existing application. Updates do not affect existing connections.

Changes to other CDB tables are not impacted by this enhancement:

- Changes to LUMODES take effect the next time DDF is started or on the initial session to a given LUMODE combination.
- Changes to LUNAMES and LULIST take effect as follows:
  - If DDF has not yet tried to communicate with a particular remote location, rows added to LUNAMES take effect when DDF attempts to communicate with that location.
  - If DDF has already attempted communication with a particular location, rows added to LUNAMES take effect the next time DDF is started.
- Changes to USERNAMES and MODESELECT take effect at the next thread access.

Archived

The output of the -DISPLAY LOCATION and -DISPLAY LOCATION DETAIL commands is enhanced to display these pending changes. See Figure 9-1 for an example of the command output.

```
@dis loc det
DSNL200I  @ DISPLAY LOCATION REPORT FOLLOWS-
LOCATION                                        PRDID    T ATT CONNS
::FFFF:9.30.88.92..447                          DSN10015 R           2
                                                            TRS      1
DISPLAY LOCATION REPORT COMPLETE
```

Figure 9-1 -DISPLAY LOCATION output

9.1.2 Online DDF location alias name changes

Group access uses a DB2 location name that represents all the members of a data sharing group. In contrast, member-specific access uses a *location alias name* that represents only a subset of members of a data sharing group. Remote requesters can use location alias names to establish connections with one or more specific members and have connections balanced to the subset based on capacity.

DB2 V8 introduced DDF location alias names. See *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079 for details. However, new DB2 members cannot be added or removed from the subset without stopping DB2, because the location alias name or names are specified in the bootstrap data set (BSDS), which cannot be modified while DB2 is running.

DB2 10 provides the ability to add, remove, and modify DDF location alias names without stopping either DB2 or DDF, so that DB2 remote traffic is not disrupted. A new DB2 command, -MODIFY DDF ALIAS, is introduced to manage DDF location alias names specified in the DDF communication record of DB2 BSDS. See Figure 9-2. This command can be issued only when DB2 is running, but it can be issued regardless of whether DDF is started or stopped.

For details, see *DB2 10 for z/OS Command Reference*, SC19-2972.

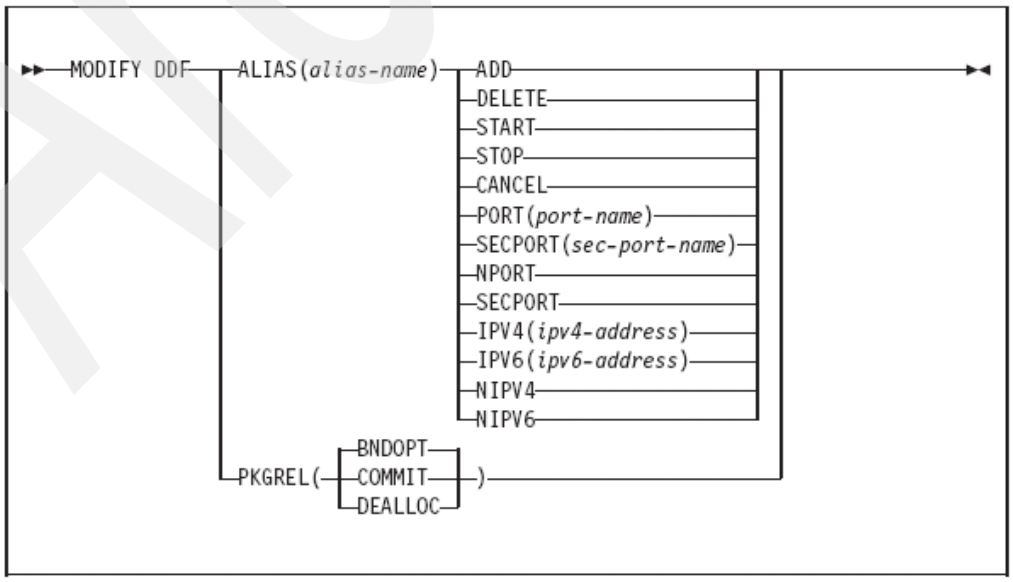


Figure 9-2 -MODIFY DDF ALIAS syntax diagram

The START keyword instructs DB2 to start accepting connection requests to the specified alias if DDF is running. If DDF is not running, then the alias is marked eligible for starting, so that it starts automatically when DDF starts next time.

Similarly, the STOP keyword instructs DB2 to stop accepting new connection requests to the specified alias. Existing data base access threads processing connections to the specified alias remain unaffected. An alias that is stopped is not started automatically when DDF starts.

CANCEL also instructs DB2 to stop accepting new connection requests to the specified alias. However, existing database access threads that are processing connections to the specified alias are cancelled. An alias that is cancelled is not started automatically when DDF starts.

The attributes of an existing alias can be modified only when the alias is stopped. The modified alias attributes take effect when the alias is started. By default, the aliases created through DSNJU003 are started, and those created by the -MODIFY DDF ALIAS command are stopped. DSNJU004 does not print the alias status information. Use the -DISPLAY DDF command report to find the status of an alias.

You can also specify a member-specific IPv4 or IPv6 address for each alias using the same command but not using the DSNJU003 utility.

You use location alias names to make an initial connection to one of the members that is represented by the alias, using member-specific access. The member that received the connection request works in conjunction with Workload Manager (WLM) to return a list of members that are currently active and able to perform work. The list includes member IP addresses and ports and includes a weight for each active member that indicates the member's current capacity. Requester systems use this information to connect to the member or members with the most capacity that are also associated with the location alias. However, with DB2 9, the IP address provided in this list is fixed regardless of how the member might be accessed.

Specifying an IP addresses for location alias names gives you the ability to control the IP addresses that is returned in the weighted server list when you connect to a location alias. However, this support is provided only with the INADDR_ANY approach. Now, when the request to WLM is for a location alias rather than location name, the returned list includes the specified member IP addresses (and ports), and a weight for each active member only in the member-specific list of alias IP addresses, if they exist, or the member-specific IP addresses.

In a data sharing environment, you can configure a DB2 member to accept connections either on any IP address active on the TCP/IP stack, including the IP address specified in the DB2 BSDS (INADDR_ANY approach), or to accept connections only on a specific IP address listed in the TCP/IP PORT statement (BINDSPECIFIC approach). DB2 9 introduced the INADDR_ANY approach and provided an advantage in a data sharing environment over the DB2 V8 function. In DB2 9, if you specify the IP address in the BSDS, you do not have to define a domain name to TCP/IP. In DB2 V8, the domain name must be defined to the TCP/IP host so that a DB2 subsystem can accept connections from remote locations. In addition, DB2 9 does not allow SSL to work with the TCP/IP BIND specific statements. See *DB2 9 for z/OS: Distributed Functions*, SG24-6952-01 and *DB2 9 for z/OS: Configuring SSL for Secure Client-Server Communications*, REDP-4630 for details.

You can use the -DISPLAY DDF command to display the status of each alias and the -DISPLAY DDF ALIAS command to display the associated alias IP addresses, if any. Figure 9-3 shows the output from the -DISPLAY DDF command.

```
DSNL080I  @ DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I LOC1               LU1               -NONE
DSNL084I TCPPORT=446   SECPORT=0     RESPORT=5001  IPNAME=-NONE
DSNL085I IPADDR=::1.3.23.80
DSNL086I SQL    DOMAIN=hello.abc.com
DSNL087I ALIAS              PORT  SECPORT STATUS
DSNL088I ALS1               5004          CNCLD
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Figure 9-3 DISPLAY DDF command

Message DSN088I lists the DDF location alias names defined. The STATUS column can have the following values:

STARTD	Alias is started.
STOPD	Alias is stopped.
CANCLD	Alias is cancelled
STARTG	Alias is starting.
STOPG	Alias is stopping.
CANCLG	Alias is canceling.
STATIC	Alias is static, meaning, a DSNJU003 defined alias.

Archived

Figure 9-4 shows the output from the -DISPLAY DDF DETAIL command.

```
@DISPLAY DDF DETAIL
DSNL080I @ DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION LUNAME GENERICLU
DSNL083I LOC1 LU1 -NONE
DSNL084I TCPPORT=446 SECPORT=0 RESPORT=5001 IPNAME=-NONE
DSNL085I IPADDR=::1.3.23.80
DSNL086I SQL    DOMAIN=hello.abc.com
DSNL086I RESYNC DOMAIN=hello.abc.com
DSNL087I ALIAS              PORT  SECPORT STATUS
DSNL088I ALS1               5004          CANCLD
DSNL088I ALS2               5005          STARTD
DSNL089I MEMBER IPADDR=::1.3.23.81
DSNL090I DT=I  CONDBAT=     64 MDBAT=   64
DSNL092I ADBAT=    0 QUEDBAT=      0 INADBAT=      0 CONQUED=      0
DSNL093I DSCDBAT=      0 INACONN=      0
DSNL100I LOCATION SERVER LIST:
DSNL101I WT IPADDR           IPADDR
DSNL102I 64 ::1.3.23.81
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE

@DISPLAY DDF ALIAS(ALS1) DETAIL
DSNL080I  @ DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL087I ALIAS              PORT  SECPORT STATUS
DSNL088I ALS1               5004          CANCLD
DSNL089I  MEMBER IPADDR= ::1.2.3.4
DSNL089I  MEMBER IPADDR= 2001::2:3:4
DSNL096I  ADBAT=      0 CONQUED=      0 TCONS=      0
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE

@DISPLAY DDF ALIAS(ALS2) DETAIL
DSNL080I  @ DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL087I ALIAS              PORT  SECPORT STATUS
DSNL088I ALS2               5005          STARTD
DSNL089I  MEMBER IPADDR= ::5.6.7.8
DSNL089I  MEMBER IPADDR= 2005::6:7:8
DSNL096I  ADBAT=      0 CONQUED=      0 TCONS=      0
DSNL100I  LOCATION SERVER LIST:
DSNL101I  WT IPADDR           IPADDR
DSNL102I  64 ::5.6.7.8        2005::6:7:8
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Figure 9-4 DISPLAY DDF DETAIL command

Message DSN089I lists the IP addresses that are associated with a DDF location alias name.

When an IP address is associated with a location alias name DB2 tries to activate this IP address as a dynamic VIPA, provided the IP address is specified in the TCP/IP VIPARANGE statement. If the activation fails, no error messages is printed, and the IP address is still returned to the client, who must then be able to use it to route future requests.

Archived

The following enhanced messages are introduced also:

```
DSNL300I csect MODIFY DDF REPORT FOLLOWS:
DSNL302I ALIAS alias1 IS SET TO START
DSNL301I csect MODIFY DDF REPORT COMPLETE
DSNL303I csect MODIFY DDF command failed with REASON = reason-code
DSNL304I csect Alias alias_name already exists in the BSDS
DSNL305I csect Alias alias_name does not exist in the BSDS
DSNL306I csect Maximum limit for Aliases reached
DSNL307I csect Alias parameter alias_parm set to parm_value is invalid
DSNL308I csect Alias IP address ip_address does not exist
DSNL309I csect Member-specific IP addresses not defined in the BSDS
DSNL310I csect Alias port port is duplicate
DSNL311I csect Alias port port does not exist
DSNL312I ALIAS alias_name cannot be started
DSNL313I ALIAS alias_name cannot be stopped or cancelled
DSNL314I Alias alias_name cannot be modified while started
```

9.1.3 Domain name is now optional

With DB2 9, the domain name that is associated with the DDF IP address has to be resolved prior to DDF completing its processing. This requirement of resolving the DB2 domain name is historical. IBM Data Server drivers and other clients, including z/OS requesters, now use the DB2 server's IP address, as their first choice to establish connections to a DB2 server for 2-phase commit resynchronization. The DB2 server also provides its domain name, in case the clients want to use it as a backup to the IP address. However, this domain name does not provide any real use because if a connection made with an IP address fails, either because the target DB2 is down or because of other firewall issues, using a domain name (that resolves to the same IP address), also fails. So, there is practically no benefit of providing a DDF domain name.

DB2 10 tolerates the absence of a domain name when an IP address is defined in the BSDS, because the IP address is not going to change. It is now no longer necessary to configure the domain name if you specify the IP address in the BSDS; however, a domain name is still required if you specify the IP address in the TCP/IP PORT statement. The change is made for both data sharing and non-data sharing environments and is retrofitted to DB2 9.

When using Dynamic VIPA, the IP addresses that are assigned to DB2 by the TCP/IP profile, must still be registered with the domain name server (DNS). However, IP addresses that are registered in the BSDS no longer need to be registered in the DNS.

When a domain name is unavailable, A DSNL523I message that contains an IP address is issued, in lieu of a DSNL519I message that contains a domain name to indicate that DDF is initialized to process TCP/IP requests.

9.1.4 Acceptable period for honoring cancel requests

SQL queries that originate from remote client system applications can sometimes run for hours, consuming database and CPU resources, even though the DB2 access thread (DBAT) was canceled. This situation typically occurs when a remote client terminates its connection to the DB2 server. The impact is that users can be charged for wasted CPU costs that are never materialized to applications, and the only way to terminate the DBAT (that is consuming the cost) is to terminate the entire DB2 subsystem.

Prior versions of DB2 already try to detect these types of situations. However, in an effort to eliminate DB2 outages, DB2 10 detects more of these CPU intensive DBATs.

9.1.5 Sysplex balancing using SYSIBM.IPLIST

You can target remotely connecting to only a subset of DB2 members of a data sharing group in either of the following methods:

- DB2 for z/OS as an application requester using member-specific routing by coding entries in the catalog table SYSIBM.IPLIST
- Routing DRDA requests to a subset of members of a data sharing group, from any DRDA TCP/IP application requestor, such as DB2 Connect™, based on DB2 location alias names, by defining location alias names with valid port numbers in the BSDS

Both techniques are described in detail in *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079.

When you use member subsetting (location alias names with specific port numbers that are defined in the BSDS), DB2 for z/OS already maintains a *weighted* server list of the DB2 data sharing group for each ALIAS name that is used as a subset and returns the appropriate list to the client. However, when using the SYSIBM.IPLIST technique, DB2 for z/OS as a requestor does not maintain this list internally with current weighted list that are returned by the server, thus leading to improper workload balancing.

DB2 10 as a DRDA requestor using SYSIBM.IPLIST to connect to a subset of DB2 members is enhanced to also use the real time weight information that is returned by the server to balance connections.

9.1.6 Message-based correlation with remote IPv6 clients

Historically, correlation of work between a DB2 for z/OS and any associated remote partners has been through the logical unit of work identifier (LUWID). This concept was introduced with the initial SNA distributed support and continued to be used when TCP/IP support was introduced, because an IPv4 (32-bit) address could still be represented successfully (8 or 4-byte character form) in the LUWID.

With the introduction of IPv6 support in DB2 9, an IPv6 (128-bit) address can no longer be represented in an LUWID, and the concept of an *extended correlation token* was introduced. This extended correlation token represented the entire IPv6 address, but DB2 9 provides this extended correlation token only in Display Thread command reports and trace records.

DB2 10 provides the extended correlation token in more messages, making it much easier to correlate message-related failures to the remote client application that is involved in the failure. See Figure 9-5.

```
DSNL027I  @ SERVER DISTRIBUTED AGENT WITH
           LUWID=G91702F8.CD1B.101018185602=16
THREAD-INFO=ADMF001:mask:admf001:db2bp:*:*:*:<9.23.2.248.52507.101018185602>
           RECEIVED ABEND=04E
           FOR REASON=00D3001A
DSNL028I  @ G91702F8.CD1B.101018185602=16
           ACCESSING DATA FOR
             LOCATION ::FFFF:9.23.2.248
             IPADDR ::FFFF:9.23.2.248
```

Figure 9-5 Extended correlation token in messages

The DDF DSNL027I and DSNL030I messages and lock-related messages DSNT318I, DSNT375I, DSNT376I, DSNT377I, and DSNT378I are changed to also contain THREAD-INFO information. The THREAD-INFO information is enhanced in these messages to contain the extended correlation token.

Because all the components of the THREAD-INFO information are delimited by a colon (:), the extended correlation token is enclosed in Less Than (<) and Greater Than (>) characters to help distinguish the THREAD-INFO components from the extended correlation components.

For details about the THREAD-INFO information provided, see message DSNT318I at *DB2 10 for z/OS Messages*, GC19-2979.

9.2 Monitoring and controlling connections and threads at the server

In typical customer environments, DB2 on z/OS is accessed remotely from many different applications. These applications typically have different characteristics. Some applications are more important than others, and some applications have more users than others. These differences present the following challenges:

- To control the number of connections at an application level, DB2 9 includes a set of knobs, CONDBAT and MAXDBAT, that control connections and threads at a subsystem level.
- Rogue applications might flood DB2 9 subsystems, which would consume all connections and affect important business applications.

DB2 9 also includes profiles to identify a query or set of queries. Profile tables contain information about how DB2 executes or monitors a group of statements. Profiles are specified in SYSIBM.DSN_PROFILE_TABLE. In DB2 9, it is possible to identify SQL statements by authorization ID and IP address or to specify combinations of plan name, collection ID, and package name. You can then monitor all statements that are identified in this manner.

SQL statements identified by a profile are executed based on keywords and attributes specified by customers in SYSIBM.DSN_PROFILE_ATTRIBUTES. These tables are used by tools such as the Optimization Service Center as described in *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421.

DB2 10 for z/OS enhances the profile table monitoring facility to support the filtering and threshold monitoring for system related activities, such as the number of connections, the number of threads, and the period of time that a thread can stay idle. This enhancement allows you to enforce the thresholds (limits) that were previously available only at the system level using DSNZPARM, such as CONDBAT, MAXDBAT, and IDTHTOIN, at a more granular level.

This enhancement allows you to control connections using the following categories:

- IP Address (IPADDR)
- Product Identifier (PRDID)
- Role and Authorization Identifier (ROLE, AUTHID)
- Collection ID and Package Name (COLLID, PKGNAME)

The filtering categories are mutually exclusive.

This support is available only for connections coming to DB2 on z/OS over DRDA.

This enhancement also provides the option to define the type of action to take after these thresholds are reached. You can display a warning message or an exception message when the connection, thread, and idle thread timeout controls are exceeded. If the user chooses to display a warning message, a DSNT771I or DSNT772I message is issued, depending on DIAGLEVEL and processing continues. In the case of exception processing, a message is displayed to the console and the action taken (that is queuing, suspension, or rejection) depends on the type of attribute (connection, threads, or idle thread timeout) that is defined.

This monitoring capability is started and available only when you issue the START PROFILE command. After you issue a START PROFILE command, any rows with a Y in the PROFILE_ENABLED column in SYSIBM.DSN_PROFILE_TABLE are now in effect.

Monitoring can be stopped by issuing the STOP PROFILE command. Monitoring for individual profiles can be stopped by updating the PROFILE_ENABLED column in the SYSIBM,DSN_PROFILE_TABLE to N and issuing a START PROFILE command again.

9.2.1 Create the tables

Monitoring using profiles requires the following tables:

- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY

These tables must be created if they do not exist. The HISTORY tables have the same columns as the PROFILE or ATTRIBUTES tables, except that there is a STATUS column instead of a REMARKS column. HISTORY tracks the state of rows in the other tables or why a row was rejected.

9.2.2 Insert a row in DSN_PROFILE_TABLE

DSN_PROFILE_TABLE has one row per monitoring or execution profile. Rows are inserted by users using tools such as SQL Processor Using File Input (SPUFI). A row can apply either to statement monitoring or to system level activity monitoring, but not both. The monitoring can be done based on options such as IP address, authid, plan name, package name, and collection ID. DB2 10 adds role and product ID as additional options to monitor system

activities (connections and threads). To monitor connections or threads, insert a row into DSN_PROFILE_TABLE with the appropriate criteria.

Valid filtering criteria for monitoring system activities can be organized into categories as shown in Table 9-1.

Table 9-1 Categories for system group monitoring

Filtering category	Columns to specify
IP address	Specify only the LOCATION column
Client product identifier	Specify only the PRDID column
Role on authorization identifier	Specify one or all of the following columns: ► ROLE ► AUTHID
Collection identifier or package name	Specify one or all of the following columns: ► COLLID ► PKGNAME

Table 9-2 is an excerpt of columns and a sample of rows in the DSN_PROFILE_TABLE.

Table 9-2 Sample of contents of DSN_PROFILE_TABLE

ROLE	AUTHID	LOCATION	PRDID	COLLID	PKGNAME	PROFILEID	PROFILE_ ENABLED
Null	Tom	Null	Null	Null	Null	20	Y
Null	Null	Null	JCC03570	Null	Null	21	Y
Null	Null	129.42.16.152	Null	Null	Null	22	Y

PROFILEID 20 row defines a profile that matches all the threads that are run by user Tom.

PROFILEID 21 row defines a profile that matches all the threads that are using client driver level JCC03570.

PROFILEID 22 row defines a profile that matches all the threads and connections that are connecting from IP address (location) 129.42.16.152.

Multiple profile rows can apply to an individual execution or process, in which case the more restrictive profile is applied.

9.2.3 Insert a row in DSN_PROFILE_ATTRIBUTES

DSN_PROFILE_ATTRIBUTES relates profile table rows to keywords that specify monitoring or execution and attributes that define how to direct or define the monitoring or execution:

1. Insert a value in the PROFILEID column to identify which profiles use this type monitoring function by entering the matching value from the PROFILEID column of the SYSIBM.DSN_PROFILE_TABLE column.
2. Specify the monitoring function by inserting values into the remaining columns of the DSN_PROFILE_ATTRIBUTES table. DB2 10 for z/OS introduces the following keywords to use for monitoring connections and threads:
   - MONITOR THREADS

The profile monitors the total number of concurrent active threads on the DB2 subsystem. The monitoring function is subject to the filtering on IPADDR, PRDID, ROLE or AUTHID, and COLLID or PKGNAME, which are defined in SYSIBM.DSN_PROFILE_TABLE.

– MONITOR CONNECTIONS

The profile monitors the total number of remote connections from the remote requesters using TCP/IP, which includes the current active connections and the inactive connections. The monitoring is subject to the filtering on the IPADDR column only in SYSIBM.DSN_PROFILE_TABLE for remote connections. Active connections are those connections that are currently associated with an active database access thread or that are queued and waiting to be serviced. Inactive connections are those connections that are currently not waiting and that are not associated with a database access thread.

– MONITOR IDLE THREADS

The profile monitors the approximate time (in seconds) that an active server thread is allowed to remain idle.

3. Insert values in the ATTRIBUTE1 column of DSN_PROFILE_ATTRIBUTES to specify how DB2 responds when a threshold is met. The values can be WARNING or EXCEPTION along with levels of diagnostic information, DIAGLEVEL1 or DIAGLEVEL2.
4. Insert values in the ATTRIBUTE2 column of DSN_PROFILE_ATTRIBUTES to specify the threshold that the monitor uses.

   If MONITOR THREADS is specified as the keyword, then the value in ATTRIBUTE2 column cannot exceed the value specified for MAXDBAT.

   If MONITOR CONNECTIONS is specified as the keyword, then the value in the ATTRIBUTE2 column cannot exceed the value specified in CONDBAT.

   If MONITOIR IDLE THREADS is specified as the keyword, then the value specified in ATTRIBUTE2 column is applied independent of the value specified for IDTHTOIN.

Figure 9-6 is an example of rows in DSN_PROFILE_ATTRIBUTES.

ProfileID	Keywords	Attribute1	Attribute2	Attribute3	Attribute Timestamp
20	MONITOR THREADS	EXCEPTION	10		2009-12-19...
21	MONITOR IDLE THREADS	WARNING	180		2009-12-19...
22	MONITOR CONNECTIONS	EXCEPTION	45		2009-12-19...

Figure 9-6 Contents of DSN_PROFILE_ATTRIBUTES

The first row indicates that DB2 monitors the number of threads that satisfy the scope that is defined by PROFILEID 20 in SYSIBM.DSN_PROFILE_TABLE, which is all threads using authid TOM. When the number of the threads in the DB2 system exceeds 10, which is defined in ATTRIBUTE2 column, a DSNT772I message is generated to the system console. DB2 queues or suspends the number of any new coming connection request up to 10, the defined exception threshold in ATTRIBUTE2, because EXCEPTION is defined in the ATTRIBUTE1 column. When the total number of threads that are queued or suspended reaches 10, DB2 starts to fail the connection request with SQLCODE -30041, meaning that there can be up to 10 threads for this profile *plus* up to 10 queued connections for this profile.

The second row monitors idle threads for PROFILEID 21, that is any thread using client driver level JCC03570. When the thread stays idle for more than 3 minutes, DB2 issues a DSNT771I console message and lets the thread continue stay idle because the thread is monitored by PROFILEID 21 with a warning action defined.

The third row monitors connections for PROFILEID 22, which is any connection coming from IP address 129.42.16.152. The profile attribute table definition shows that when more than 45 concurrent connections come to DB2 from the same IP address (129.42.16.152), a DSNT772I message is sent to the console, and the connection is failed.

9.2.4 Activate profiles

After the profiles are defined, you can activate them dynamically using the DB2 START PROFILE command.

9.2.5 Deactivate profiles

You can deactivate profiles dynamically using the DB2 STOP PROFILE command.

9.2.6 Activating a subset of profiles

To activate a subset of profiles, delete that row from SYSIBM.DSN_PROFILE_TABLE, or change the PROFILE_ENABLED column value to N. Then, reload the profile table by issuing the DB2 START PROFILE command.

9.3 JDBC Type 2 driver performance enhancements

The DB2 database system provides driver support for client applications and applets that are written in Java using JDBC and for embedded SQL for Java (SQLJ). JDBC is an application programming interface (API) that Java applications use to access relational databases. DB2 support for JDBC lets you write Java applications that access local DB2 data or remote relational data on a server that supports DRDA. SQLJ provides support for embedded static SQL in Java applications. In general, Java applications use JDBC for dynamic SQL and SQLJ for static SQL. However, because SQLJ can inter-operate with JDBC, an application program can use JDBC and SQLJ within the same unit of work.

The DB2 product includes support for the following types of JDBC driver architecture:

- Type 2

  This implementation of the JDBC driver is written partly in the Java and partly in native code. The drivers use a native client library specific to the data source to which they connect. The native component allows the type 2 JDBC driver to connect to DB2 and issue SQL directly using cross memory services.

- Type 4

  This implementation includes drivers that are pure Java and implements the network protocol for a specific data source.

  The client connects directly to the DB2 on z/OS over a network through the Distributed Data Facility (DDF) using DRDA protocol.

DB2 for z/OS supports a single driver that combines type 2 and type 4 JDBC implementations. When running Java applications on z/OS that connect to DB2 on z/OS on

the same LPAR, use type 2 connectivity. For applications that connect to DB2 on z/OS over the network, use type 4 connectivity. The recommendation to use type 2 connectivity is based on performance.

Type 2 connectivity uses cross-memory services when connecting to DB2 on z/OS, whereas type 4 has network latency. Recent enhancements to type 4 connectivity and performance in some cases are actually better than type 2. Areas in which type 4 performs better than type 2 include cases when applications return multiple rows (cursor processing) and progressive streaming of LOBS and XML. These areas implement limited block fetch capability. This enhancement in the JDBC type 2 driver in DB2 10 now eliminates these performance issues.

9.3.1 Limited block fetch

With *limited block fetch*, the DB2 for z/OS server attempts to fit as many rows as possible in a query block. DB2 transmits the block of rows over the network. Data can also be pre-fetched when the cursor is opened without needing to wait for an explicit fetch request from the requester.

JDBC Type 2 driver behavior without limited block fetch

Figure 9-7 depicts the flow of calls from a Java application running on z/OS to DB2 on z/OS on the same LPAR using type 2 connectivity to DB2 9.

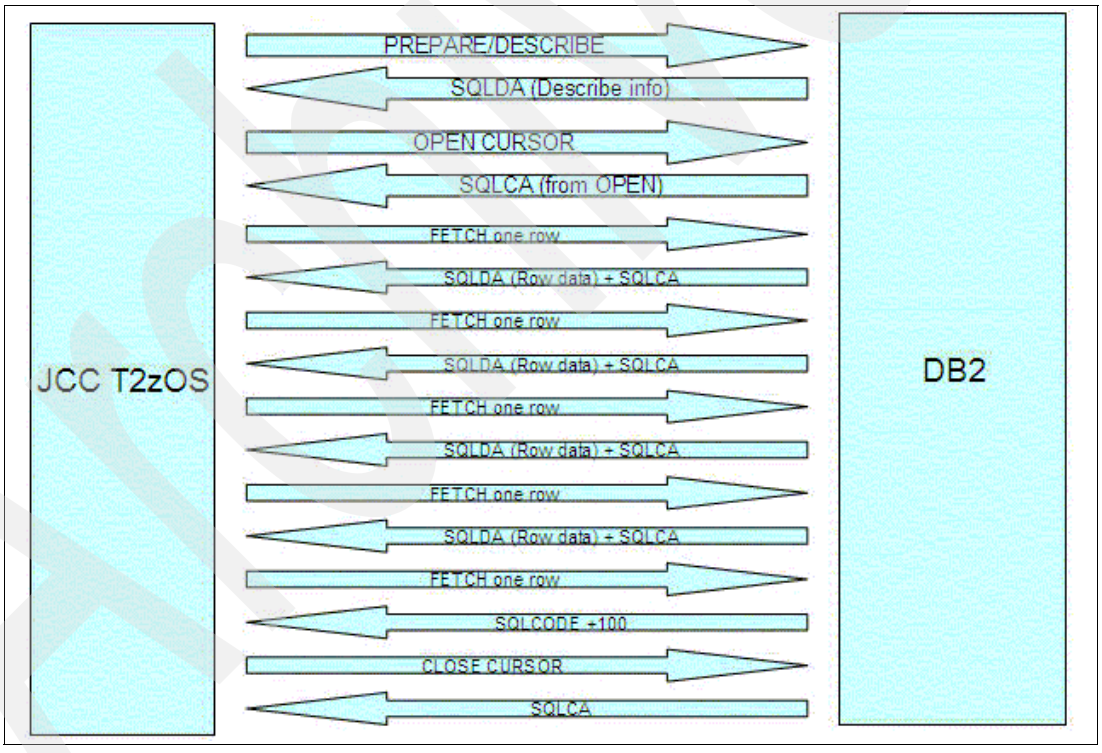


Figure 9-7 JDBC type 2 with DB2 9

In this case, the Java application running on z/OS opens a connection to DB2, prepares an SQL statement for execution, and executes the SQL statement. Let us assume that the execution of the SQL statement qualifies 100 rows. The Java application then issues a fetch statement to get each row. Each fetch operation results in a call to DB2 to return the data. After the application has processed the row, it issues the next fetch statement, which means for this example, 100 fetch calls. Each fetch call is a separate call to DB2 by the JDBC driver to fetch one row. Then, after all the rows have been fetched and processed by the application,

it issues close, which is another call to the DB2 server. This process is irrespective whether the application runs in a application server such as WebSphere Application Server or in a stand-alone Java application.

JDBC Type 2 driver behavior with limited block fetch

Now let us examine the behavior in DB2 10. Figure 9-8 depicts the sequence of calls from a Java application running on z/OS to DB2 on z/OS on the same LPAR using type 2 connectivity to DB2 10.

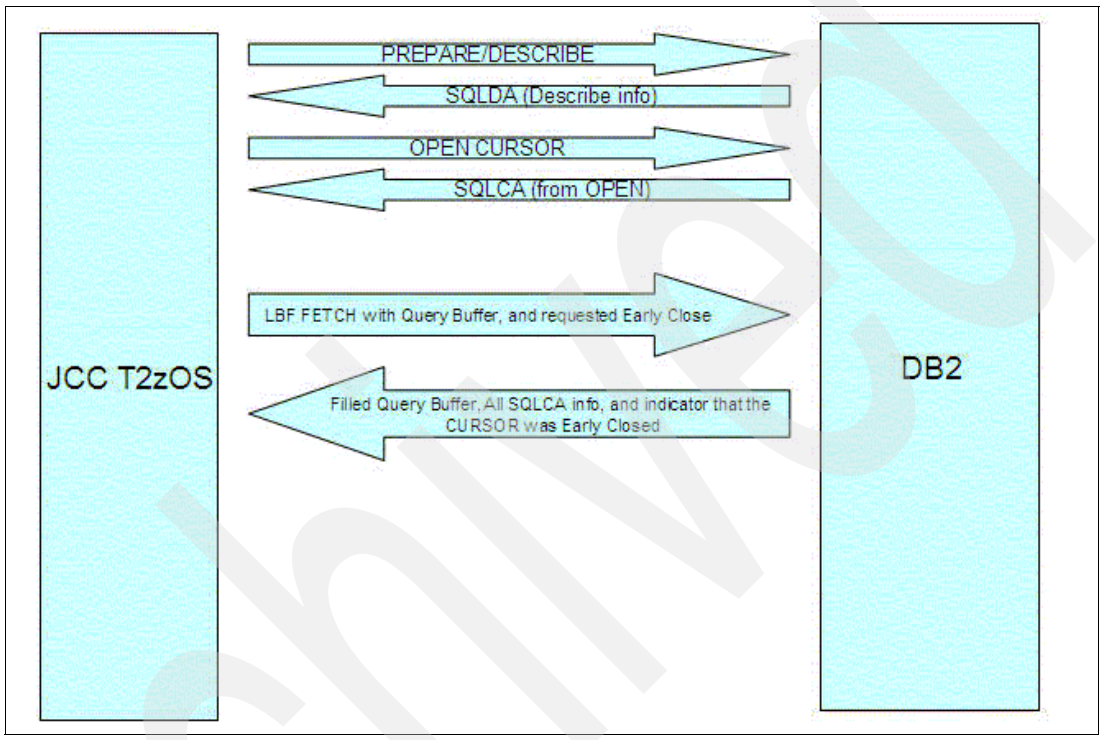


Figure 9-8 JDBC type 2 with DB2 10

In Figure 9-8, the Java application running on z/OS opens a connection to DB2, prepares a SQL statement for execution, and executes the SQL statement. This action results in DB2 returning multiple rows. Let us assume that the execution of the SQL statement qualifies 100 rows. The Java application then issues a fetch statement to get each row. Because the JDBC type 2 driver is enhanced to provide limited block fetch capability, the driver in this case returns as many rows as possible that fit in a buffer from a single fetch call issued by the application from DB2. The number of rows that are returned depends on the buffer size, which is controlled by the DataSource/Connection queryDataSize property. The default is 32 KB. Valid values for a DB2 10 for z/OS target are 32, 64, 128, or 256 KB.

It is important to note that a block of rows is returned to the Java application. Thus, all the rows that fit the buffer size (32 KB is the default) are available in the JVM. It is also important to note that only one fetch is issued by the application, and the application still has not processed rows. From the application side, it has issued a single fetch statement to fetch and process a single row. In this case, the driver working with DB2 on z/OS has already fetched all rows and placed them in memory in the application address space. After processing the first row, when the application issues the next fetch statement to get the next row, the driver actually returns the row that is in memory in the application address space and does not make a call to DB2.

JDBC type 2 driver is also enhanced to implement early close of the cursors after all the rows are returned. Instead of a separate call to close the cursor from the application, the driver closes the cursor in DB2 implicitly.

A common question is how this process is different from *multi-row fetch*. Multi-row fetch also returns multiple rows in a single call to DB2. However, multi-row fetch has limited block fetch. Multi-row fetch returns columns from each row into arrays that are declared in application programs. JDBC specification does not currently support arrays. Thus, the driver has to convert the arrays into rows after the fetch.

9.3.2 Conditions for limited block fetch

This limited block fetch capability and early close of cursor is available only when the following conditions are met:

- A forward scrollable and non-updateable cursor is used
- Fully materialized lob is false and progressive streaming is true
- Type 2 Multi-Row fetch is disabled

9.4 High performance DBAT

Prior to DB2 10, the release option of packages is not honored in distributed applications. Packages when allocated were de-allocated at commit for distributed applications, mainly to help with performance functions, such as issue DDL, run utilities, BIND packages, and other functions that were impacted if the packages were not de-allocated after commit. Although this behavior helped, it came with a price. The performance analysis of inactive connection processing saw that a large CPU expenditure occurred in package allocation and de-allocation with a lesser CPU expense occurring due to the processing to pool a DBAT and then associating a pooled DBAT with a connection.

DB2 10 includes the following enhancements:

- CPU reduction of inactive connection processing
- An easy method to switch between RELEASE (COMMIT) and RELEASE (DEALLOCATE) for existing applications without having to rebind, so that activities such as running DDL, running utilities, and other activities can happen without an outage

9.4.1 High performance DBAT to reduce CPU usage

DB2 10 DRDA database access threads (DBATs) are allowed to run, accessing data under the RELEASE BIND option of the package. If a package that is associated with a distributed application is bound with RELEASE (DEALLOCATE), the copy of the package is allocated to the DBAT up until the DBAT is terminated. The DBATs hold package allocation locks even while they are not being used for client unit-of-work processing. However, to minimize the number of different packages that can possibly be allocated by any one DBAT, distributed data facility (DDF) does not pool the DBAT and disassociates it from its connection after the unit-of-work is ended. After the unit-of-work is ended, DDF cuts an accounting record and deletes the WLM enclave, such as inactive connection processing. Thus, the client application that requested the connection holds onto its DBAT, and only the packages that are required to run the application accumulate allocation locks against the DBAT.

The key thing to remember is that even if one package among many is bound with RELEASE(DEALLOCATE), then the DBAT becomes a high performance DBAT, provided that

it meets the requirements. Also, similar to inactive connection processing, the DBAT is terminated after 200 (not user changeable) units-of-work are processed by the DBAT. The connection at this point is made inactive. On the next request to start a unit-of-work by the connection, a new DBAT is created or a pooled DBAT is assigned to process the unit-of-work. Normal idle thread time-out detection is applied to these DBATs.

If the DBAT is in flight processing a unit-of-work and if it has not received the next message from a client, DDF cancels the DBAT after the IDTHTOIN value has expired. However, if the DBAT is sitting idle, having completed a unit-of-work for the connection, and if it has not received a request from the client, then the DBAT is terminated (not cancelled) after POOLINAC time expires.

9.4.2 Dynamic switching to RELEASE(COMMIT)

In DB2 10 new-function mode, the default behavior is to honor the bind option for distributed application threads. You can switch between modes dynamically using the MODIFY DDF command:

- The MODIFY DDF PKGREL BNDOPT command causes DDF to honor the bind option that is associated with the package of a distributed application.
- The MODIFY DDF PKGREL COMMIT command causes DDF to request that any package used for remote client processing be allocated under RELEASE(COMMIT) BIND option rules, regardless of the value of the package's RELEASE bind option.
- To establish RELEASE(DEALLOC) behavior, you need to explicitly BIND with RELEASE(DEALLOC) and issue PKGREL(BNDOPT).

  JCC and DB2 Connect 9.7 FP3a has changed the default BIND option to RELEASE(DEALLOC).

Ideally, during maintenance windows or low periods of activity, you can use the COMMIT option to switch from the other options so that you can run utilities, DDL, bind packages, and other applications.

9.4.3 Requirements

This honoring of packages bind options for distributed application threads is available only under the following conditions:

- KEEPDYNAMIC YES is not enabled.
- CURSOR WITH HOLD is not enabled.
- CMTSTAT is set to INACTIVE.

9.5 Use of RELEASE(DEALLOCATE) in Java applications

Rebinding the JCC packages, which are used by a remote connections to DB2 10 for z/OS, with the RELEASE(DEALLOCATE) and KEEPDYNAMIC(NO) options can save CPU by eliminating the need for package allocation and deallocation. You can bind the packages using the DB2Binder utility or through the client configuration assistant, or you can rebind them at the DB2 10 for z/OS system.

Instead of using the default NULLID collection, you might consider different collection IDs for packages bound with different options, such as RELEASE DEALLOCATE or COMMIT. You might also consider defining different instances of the IBM Data Server Driver for JDBC and

SQLJ to be used at run time. For example, using the DB2Binder utility, you can bind the DB2 Data Server Driver packages at the DB2 for z/OS server using a `-collection` option with a value such as RELDEAL for packages bound with the RELEASE(DEALLOCATE) option and the `-collection` option with a value such as RELCOMM for packages bound with RELEASE(COMMIT).

When using a collection ID other than the default NULLID, ensure that your data source `property: jdbcCollection` is set with the correct collection ID.

Both the older CLI-based JDBC driver and the IBM Data Server Driver for JDBC and SQLJ type 2/4 connectivity use the same packages. They are also called *DB2 CLI packages*. You can find more information in *DB2 Version 9.5 for Linux, UNIX, and Windows Call Level Interface Guide and Reference, Volume 2*, SC23-5845.

Example 9-1 shows a REBIND job of the packages with the two options.

Example 9-1 REBIND job example

```
//WDRJCCRB   JOB A,MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,
//  USER=&SYSUID,NOTIFY=&SYSUID,REGION=0M,TIME=1440
/*JOBPARM S=S34
//USRLIB  JCLLIB ORDER=WP1G.A10.PROCLIB
//JOBLIB   DD DISP=SHR,DSN=WP1G0.S34.SDSNLOAD
//*
//JCCRBD  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(WP1G)
  REBIND PACKAGE (RELDEAL.SYSLH100) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH101) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH102) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH200) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH201) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH202) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH300) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH301) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH302) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH400) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH401) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLH402) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN100) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN101) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN102) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN200) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN201) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN202) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN300) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN301) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN302) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN400) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN401) RELEASE(DEALLOCATE)
  REBIND PACKAGE (RELDEAL.SYSLN402) RELEASE(DEALLOCATE)
```

Under WebSphere Application Server, consider the following factors for local or Type 2 connections:

- jdbcCollection does not apply to Type 2 Connectivity on DB2 for z/OS.
- WebSphere on z/OS users should set the aged timeout connection pool option to 120 seconds.

You can find more information in *DB2 9 for z/OS: Distributed Functions*, SG24-6952, and *DB2 9 for z/OS: Packages Revisited*, SG24-7688.

See also:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.java.doc/doc/t0024156.html

9.6 Support for 64-bit ODBC driver

The DB2 9 implementation of the DB2 z/OS ODBC driver does not have 64-bit support. Many applications that use different products, such as WebSphere Message Broker, that support 64 bit cannot use the memory relief because the ODBC driver does not support 64 bit.

DB2 10 enhances the ODBC driver to provide 64-bit support. For APIs that currently accept pointer arguments, the pointer values are expanded to 64 bits, allowing for addressing up to 16 exabytes. Predominantly, the change should not impact existing applications except for the need to recompile the current ODBC applications that need to use 64-bit support with the LP64 compiler option.

However, when using a 64-bit ODBC driver, keep in mind the following considerations:

- You need to compile the application with the LP64 compiler option.
- All APIs that currently accept pointer arguments or return pointer values are modified to accommodate 64-bit values. Applications are required to pass only 64-bit pointer values as input and output for all pointer arguments. Pointers that are returned as output values to the applications are also expanded to 64 bit.
- For APIs that have function arguments, such as SQLSetConnectAttr() and SQLSetStmtAttr(), that are used to pass both pointers (8 bytes) and unsigned integer data (4 bytes), the argument data must be contained in a 64-bit pointer variable. Depending on the attributes specified, the driver either uses the argument data as an address or casts it to an unsigned 32-bit integer. Because the integer values that are passed do not exceed the maximum value for an unsigned 32-bit integer, no truncation occurs.
- All ODBC symbolic data types and defined C types that are used for declaring variables and arguments of integer type are assigned a base C type of *int* or *unsigned int*, which ensures that a 4-byte integer value is preserved.
- The following C type definitions are added to the ODBC standard:
  - SQLLEN
  - SQLULEN

  These definitions are used specifically to declare integer function arguments in 64-bit environments. The base C types for SQLLEN and SQLULEN can be compiled to either 32 bit or 64 bit, depending on the platform and the CLI/ODBC driver that the application is using.

  On the DB2 for z/OS platform, SQLLEN and SQLULEN are mapped as 32-bit integer type to be consistent with the 32-bit and 64-bit CLI drivers. When running in 64-bit

environments, you need to change arguments that were previously defined with SQLINTEGER and SQLUINTEGER to SQLLEN and SQLULEN, where appropriate, to maximize the application's portability.

SQLINTEGER and SQLUINTERGER type definitions are changed to have a base C type of *int* and *unsigned int* respectively, so that applications can continue to use them to declare 32-bit integer values. Declare 64-bit integer values using the SQLBIGINT type definitions, as it does today.

Because ints and longs are both 32 bit in 31-bit environments, the current driver uses them indiscriminately while implicitly or explicitly assuming that they are interchangeable. This use introduces a problem because some of C defined types, such as SQLHENV, SQLHDBC, and SQLHSTMT, are defined as type definitions of long. For these C defined types, the base C type is changed to int to preserve them as 32-bit integers.

- Currently, the ODBC driver uses the C wchar_t (wide character) data type to represent UCS-2 encoded data. This data type works in 31-bit environments because the lengths of wchar_t and UCS-2 data are both 2 bytes. Under LP64, wchar_t is defined as a typedef of a 32-bit unsigned integer and, therefore, can no longer be used to represent UCS-2 data. To accommodate this change, the driver is changed to map SQLWCHAR to *unsigned short* instead of wchar_t and continues to handle the SQLWCHAR data type as a 2-byte UCS-2 string.

Using addressable storage: Although 64-bit mode provides larger addressable storage, the amount of data that can be sent or retrieved to and from DB2 by an ODBC application is limited by the amount of storage that is available below the 2 GB bar. For example, an application cannot declare a 2 GB LOB above the bar and insert the entire LOB value into a DB2 LOB column.

9.7 DRDA support of Unicode encoding for system code pages

DB2 10 includes DRDA support of Unicode encoding (code page 1208) for system code pages, such as DRDA command parameters and reply message parameters. This enhancement can provide improved response time and less processor usage for remote CLI and JDBC applications by removing the need for the drivers to convert DRDA instance variables between EBCDIC and Unicode.

9.8 Return to client result sets

Prior to DB2 10, a stored procedure could only return result sets to the immediate caller. If the stored procedure is in a chain of nested calls, the result sets must be materialized at each intermediate nesting level, typically through a declared global temporary table (DGTT).

DB2 10 introduces 'Return to Client Result Set' support. With this enhancement, a result set can be returned from a stored procedure at any nesting level directly to the client calling application. No materialization through DGTTs is required. The new syntax on the DECLARE CURSOR statement is:

```
WITH RETURN TO CLIENT
```

Result sets that are defined WITH RETURN TO CLIENT are not visible to any stored procedures at the intermediate levels of nesting. They are only visible to the client that issued

the initial CALL statement. This feature is not supported for stored procedures called from triggers or functions, either directly or indirectly.

9.9 DB2-supplied stored procedures

Several useful stored procedures that provide server-side database and system administration functions are supplied by DB2 and installed using the DB2 installation jobs. These stored procedures are often used by vendor products, but they also enable you to write client applications that perform advanced DB2 and z/OS functions, such as utility execution, application programming, performance management, and data administration functions.

The following jobs install and validate the installation of DB2-supplied routines. These jobs are configured with the options that you specified on the DSNTIPR1 installation panel and the DSNTIPRA through DSNTIPRP panels during the installation or migration procedure.

- Job DSNTIJRT installs and configures all DB2-supplied routines.
- Job DSNTIJRV validates the installation of the routines.

For more information about these installation jobs, see 12.13, “Simplified installation and configuration of DB2-supplied routines” on page 523.

Depending on the function that is provided, you can find a description of the stored procedures and information about how to install and execute them in the following documentation:

- Chapter 24. “DB2-supplied stored procedures” of *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7604
- Chapter 7. “Working with additional capabilities for DB2” of *DB2 10 for z/OS Installation and Migration Guide,* GC19-2974
- Appendix B. “DB2-supplied stored procedures” of *DB2 10 for z/OS Utility Guide and Reference*, SC19-2984
- Chapter 14. “Calling a stored procedure from your application” of *DB2 10 for z/OS Application Programming and SQL Guide*, SC19-2969
- Chapter 8. “Installing the IBM Data Server Driver for JDBC and SQLJ” of *DB2 10 for z/OS Application Programming Guide and Reference for Java*, SC19-2970
- Chapter 6. “XML schema management with the XML schema repository (XSR)” of *DB2 10 for z/OS pureXML Guide,* SC19-2981
- Chapter 15. “Scheduling administrative tasks” and Appendix B. “Stored procedures for administration” of *DB2 10 for z/OS Administration Guide*, SC19-2968
- Chapter 11. “Designing DB2 statistics for performance” of *DB2 10 for z/OS Managing Performance*, SC19-2978

9.9.1 Administrative task scheduler

The administrative task scheduler starts as a task on the z/OS system during DB2 startup or initialization. The administrative task scheduler remains active unless it is explicitly stopped, even when DB2 terminates.

Every DB2 subsystem has a coordinated administrative task scheduler address space that it can start with a z/OS started task procedure. See Figure 9-9.

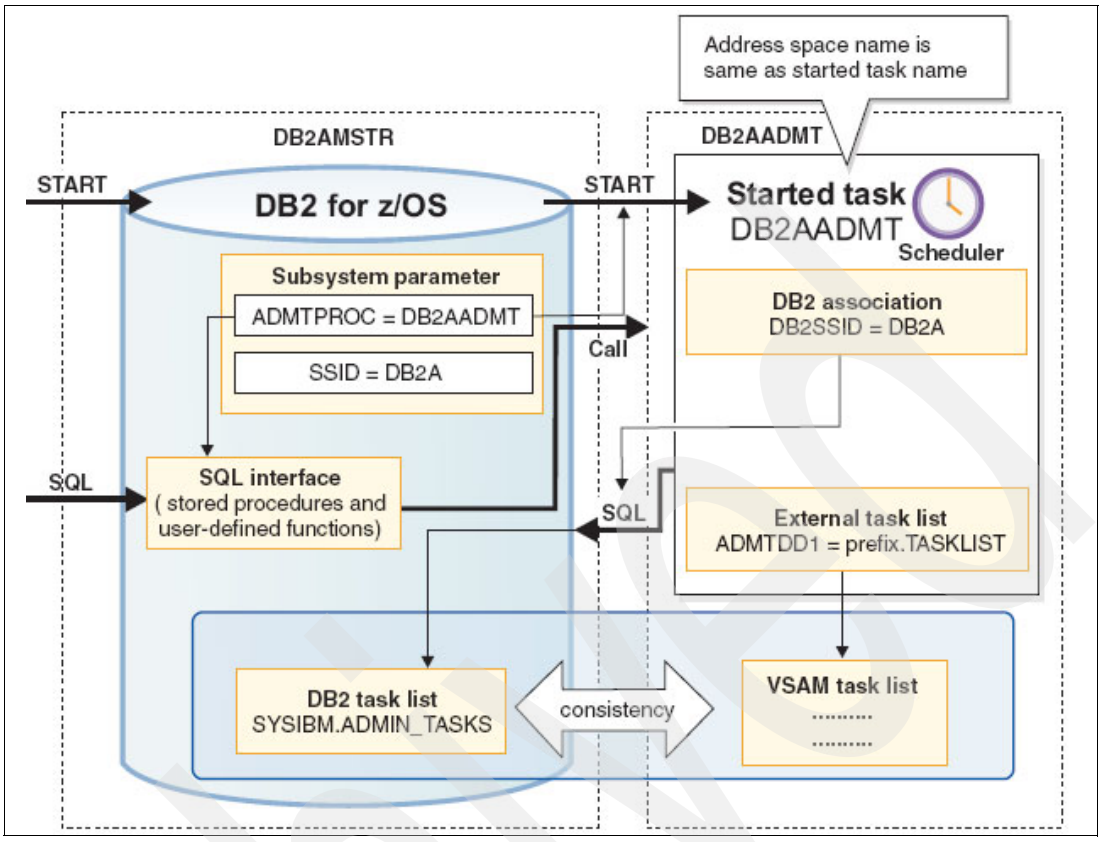


Figure 9-9 Administrative task scheduler

The administrative task scheduler has an SQL interface that currently consists of the following DB2-supplied stored procedures:

- SYSPROC.ADMIN_TASK_ADD

  The ADMIN_TASK_ADD stored procedure adds a task to the task list of the administrative task scheduler and runs in a WLM-established stored procedure address space using the Resource Recovery Services attachment facility to connect to DB2. It takes the task information in input and provides the task name and return code in output.

- SYSPROC.ADMIN_TASK_REMOVE

  The ADMIN_TASK_REMOVE stored procedure removes a task from the task list of the administrative task scheduler. If the task is currently running, it continues to execute until completion, and the task is not removed from the task list. If other tasks depend on the execution of the task that is to be removed, the task is not removed from the task list of the administrative task scheduler.

- SYSPROC.ADMIN_TASK_CANCEL

  The ADMIN_TASK_CANCEL stored procedure attempts to stop the execution of a task that is currently running.

  For a task that is running, this stored procedure cancels the DB2 thread or the JES job that the task runs in, and issues a return code of 0 (zero). If the task is not running, the stored procedure takes no action, and issues a return code of 12.

- SYSPROC.ADMIN_TASK_UPDATE

  The ADMIN_TASK_UPDATE stored procedure updates the schedule of a task that is in the task list for the administrative task scheduler. If the task that you want to update is running, the changes go into effect after the current execution finishes.

Generally MONITOR1 authority is required for the execution of the stored procedures. See APAR PM02658 for the availability of these routines with DB2 9.

The stored procedures make currently use of the following user-defined table functions to access the contents of the SYSIBM.ADMIN_TASKS and SYSIBM.ADMIN_TASKS_HIST scheduler tables:

- DSNADM.ADMIN_TASK_LIST

  The ADMIN_TASK_LIST() function returns a table with one row for each of the tasks that are defined in the administrative scheduler task list.

- DSNADM.ADMIN_TASK_OUTPUT

  The ADMIN_TASK_OUTPUT(task-name, num-invocations) function return the output parameter values and the result set values of the n-th execution of the specified stored procedure task, where n is the value of the input parameter num-invocations. This function returns a table containing one row for each output parameter value or result set value.

- DSNADM.ADMIN_TASK_STATUS()

  The ADMIN_TASK_STATUS function returns a table with one row for each task that is defined in the administrative scheduler task list. Each row indicates the status of the task for the last time it was run. You can review the last execution status of a task and identify any messages or return codes that were passed back to the administrative task scheduler The task status is overwritten as soon as the next execution of the task starts.

- DSNADM.ADMIN_TASK_STATUS(max-history)

  The ADMIN_TASK_STATUS(max-history) function accesses the history of the execution statuses and returns the contents. max-history is the maximum number of execution statuses that is returned per task. If max-history is NULL, all available execution statuses are returned. If max-history is 1, only the latest execution status is returned for each task (the same as DSNADM.ADMIN_TASK_STATUS()).

These routines and the task scheduler setup are described in *DB2 10 for z/OS Administration Guide,* SC19-2968.

9.9.2 Administration enablement

The following stored procedures were a late addition to DB2 9:

- SYSPROC.ADMIN_INFO_SMS

  ADMIN_INFO_SMS provides a DB2 interface to SMS for storage space alerts. The ADMIN_INFO_SMS stored procedure returns space information about copy pools and their storage groups and volumes.

- SYSPROC.ADMIN_INFO_SYSPARM

  The ADMIN_INFO_SYSPARM stored procedure returns the system parameters, application defaults module, and IRLM parameters of a connected DB2 subsystem, or member of its data sharing group.

- SYSPROC.ADMIN_INFO_SYSLOG

  ADMIN_INFO_SYSLOG is a new stored procedure (provided by APAR PM22091) that returns the system log entries. Filters, such as search string, system name, begin and end

date, begin and end time, and max entries, are supplied that can limit the system log entries that are returned.

- SYSPROC.ADMIN_INFO_SQL

  ADMIN_INFO_SQL procedure collects DDL, statistics, and column statistics providing information to help identifying query performance. This procedure is meant to be used by DB2 provided programs and tools. The procedure is invoked by the DSNADMSB program, which has functions equivalent to PLI8 for DB2 optimization problems diagnosis.

  ADMIN_INFO_SQL is included in DB2 10 and retrofitted into DB2 9 with APAR PM11941. Two DB2 10 APARs apply: PM25635 and PM24808.

  ADMIN_INFO_SQL has several input and output parameter. In addition, a result set is returned containing several scripts with all the data that was collected, such as DDL, statistics, and other data. The result set can be returned to the caller or written to data sets on the host.

  Before executing DSNADMSB, complete the following tasks:

  a. Create SYSPROC.ADMIN_INFO_SQL in DB2 and bind it prior to running this job. Use job DSNTIJRT to create and bind DB2-supplied stored procedures.
  b. Bind the package and plan for DSNADMSB prior to running this job. Use job DSNTIJSG to bind the package and plan for DSNADMSB.
  c. Decide what output vehicle will be used, that is data sets or job stream. Because the data can get quite large, it is important to have space available to hold the results. It is difficult to predict the size of the output as tables can have a lot of dependencies. Average space is usually 2-3 MB, but some of the larger workloads can be up to 20 MB. Make sure the column stats switch is N unless it is required because this setting can consume a lot of space.
  d. Copy the sample JCL (member DSNTEJ6I in DB2HLQ.SDSNSAMP library). Follow the directions suggested in the header notes by modifying the libraries and input parameters.

  For details, refer to the white paper Capture service information about DB2 for z/OS with SYSPROC.ADMIN_INFO_SQL, which is available from:

  http://www.ibm.com/developerworks/data/library/techarticle/dm-1012capturequery/index.html?cmp=dw&cpb=dwinf&ct=dwnew&cr=dwnen&ccy=zz&csr=121610

9.9.3 DB2 statistics routines

DB2 10 delivers the following stored procedures that are used to monitor and execute RUNSTATS autonomically:

- SYSPROC.ADMIN_UTL_MONITOR

  SYSPROC.ADMIN_UTL_MONITOR is a stored procedure that provides functions that enable analysis of database statistics. These functions include alerts for out-of-date, missing, or conflicting statistics, summary reports, and detailed table-level reports that describe generated RUNSTATS statements.

- SYSPROC.ADMIN_UTL_EXECUTE

  ADMIN_UTL_EXECUTE is a stored procedure that solves alerts stored in the SYSIBM.SYSAUTOALERTS catalog table within the maintenance windows that are defined by the SYSIBM.SYSAUTOTIMEWINDOWS catalog table.

- SYSPROC.ADMIN_UTL_MODIFY

ADMIN_UTL_MODIFY is a stored procedure that maintains the SYSIBM.SYSAUTORUNS_HIST and SYSIBM.SYSAUTOALERTS catalog tables.

The ADMIN_UTL_MODIFY stored procedure removes all entries in the SYSIBM.SYSAUTORUNS_HIST table that are older than a configurable threshold and removes all entries in the SYSIBM.SYSAUTOALERTS table that are older than the configured threshold and are in COMPLETE state.

When configured and scheduled in the Administrative Task scheduler, the ADMIN_UTL_MONITOR stored procedure monitors statistics on the given tables and writes alerts when inadequate statistics are identified. The ADMIN_UTL_EXECUTE stored procedure then invokes RUNSTATS USE PROFILE within defined maintenance windows to resolve the problem. The ADMIN_UTL_MODIFY stored procedure can be scheduled periodically to clean up the log file and alert history of both of the other stored procedures.

These routines and the set up are described in *DB2 10 for z/OS Managing Performance,* SC19-2978.

The following stored procedures are configured (DSNTIJRT), and your authorization ID has CALL privileges for each:

- ADMIN_COMMAND_DB2
- ADMIN_INFO_SSID
- ADMIN_TASK_ADD
- ADMIN_TASK_UPDATE
- ADMIN_UTL_EXECUTE
- ADMIN_UTL_MODIFY
- ADMIN_UTL_MONITOR
- ADMIN_UTL_SCHEDULE
- ADMIN_UTL_SORT
- DSNUTILU
- DSNWZP

The following user-defined functions are configured (DSNTIJRT), and your authorization ID has call privileges for each:

- ADMIN_TASK_LIST()
- ADMIN_TASK_STATUS()

Your authorization ID has privileges to select and modify data in the following catalog tables:

- SYSIBM.SYSAUTOALERTS
- SYSIBM.SYSAUTORUNS_HIST
- SYSIBM.SYSAUTOTIMEWINDOWS
- SYSIBM.SYSTABLES_PROFILES

Your authorization ID has privileges to read data in the following catalog tables:

- SYSIBM.SYSTABLESPACESTATS
- SYSIBM.SYSTABLESPACE
- SYSIBM.SYSDATABASE
- SYSIBM.SYSTABLES
- SYSIBM.SYSINDEXES
- SYSIBM.SYSKEYS
- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSCOLDIST
- SYSIBM.SYSDUMMY1
- SYSIBM.UTILITY_OBJECTS

Archive

Archived

Part 3

Operation and performance

Technical innovations in operational compliance (both regulatory and governance) help teams work more efficiently, within guidelines, and with enhanced auditing capabilities.

DB2 Utilities Suite for z/OS Version 10 (program number 5655-V41) delivers full support for the significant enhancements in DB2 10 for z/OS and delivers integration with the storage systems functions, such as FlashCopy, exploitation of new sort options (DB2 Sort), and backup and restore. The key DB2 10 performance improvements are an overall reduction in CPU time for many types of workloads, deep synergy with System z hardware and z/OS software, improved performance and scalability for inserts and LOBs, and improved SQL optimization.

DB2 10 also improves I/O performance. In some cases, DB2 10 saves disk space, which improves performance. DB2 10 also reduces the elapsed time for many workloads by improving parallelism, by reducing latch contention for many types of operations, and by eliminating the UTSERIAL lock.

This version also resolves the issue of virtual storage used below the 2 GB bar and allows more concurrent threads.

Installation and migration use the established conversion mode and new-function mode statuses. The major additional function is the possibility of migrating to DB2 10 from both DB2 9 and DB2 V8. We point out the key steps and situations where skip level migration might be convenient.

This part contains the following chapters:

- Chapter 10, “Security” on page 337
- Chapter 11, “Utilities” on page 425
- Chapter 12, “Installation and migration” on page 471
- Chapter 13, “Performance” on page 533

© Copyright IBM Corp. 2010. All rights reserved.

Archived

10

Security

For regulatory compliance reasons (for example, Basel II, Sarbanes-Oxley, EU Data Protection Directive), and other reasons such as accountability, audit ability, increased privacy and security requirements, many organizations focus on security functions when designing their IT systems. DB2 10 for z/OS provides a set of options that improve and further secure access to data held in DB2 for z/OS to address these challenges.

In this chapter, we highlight the following security topics:

- Reducing the need for SYSADM authorities
- Separating the duties of database administrators from security administrators
- Protecting sensitive business data against security threats from insiders, such as database administrators, application programmers, and performance analysts
- Further protecting sensitive business data against security threats from powerful insiders such as SYSADM by using row-level and column-level access controls
- Simplifying security administration by preserving dependent privileges during SQL REVOKE
- Using the RACF profiles to manage the administrative authorities
- Auditing access to business sensitive data through policy-based SQL auditing for tables without having to alter the table definition
- Auditing the efficiency of existing security policies using policy-based auditing capabilities
- Benefitting from security features introduced recently by z/OS Security Server, including support for RACF password phrases (z/OS V1R10) and z/OS identity propagation (z/OS V1R11)

This chapter includes the following sections:

- Policy-based audit capability
- More granular system authorities and privileges
- System-defined routines
- The REVOKE dependent privilege clause
- Support for row and column access control
- Support for z/OS security features

© Copyright IBM Corp. 2010. All rights reserved.

10.1 Policy-based audit capability

DB2 provides a variety of authentication and access control mechanisms to establish rules and controls. However, to protect against and to discover and eliminate unknown or unacceptable behaviors, you need to monitor data access. DB2 10 assists you in this task by providing a powerful and flexible audit capability that is based on audit policies and categories, helping you to monitor application and individual user access, including administrative authorities. When used together with the audit filtering options introduced in DB2 9 for z/OS, policy-based auditing can provide granular audit reporting. For example, you can activate an audit policy to audit an authorization ID, a role, and DB2 client information.

The auditing capability is available in DB2 10 new-function mode (NFM) mode.

10.1.1 Audit policies

An *audit policy* provides a set of criteria that determines the categories that are to be audited. Each category determines the events that are to be audited. You can define multiple audit policies based on your audit needs.

Each policy has a unique name assigned, which you use when you complete the following tasks:

- Create an audit policy by inserting a row into the SYSIBM.SYSAUDITPOLICIES table
- Enable an audit policy by issuing a START trace command
- Display the status of an activated audit policy by issuing a DISPLAY TRACE command
- Disable an audit policy by issuing a STOP TRACE command

Additional information: For information about how to collect DB2 traces in SMF or GTF, refer to *DB2 10 for z/OS Managing Performance,* SC19-2978.

Audit categories

There are several security events that must be audited to comply with laws and regulations and to monitor and enforce the audit policy. For example, auditing is usually performed for the following events:

- Changes in authorization IDs
- Unauthorized access attempts
- Altering, dropping, and creating database objects
- Granting and revoking authorities and privileges
- Reading and changing user data
- Running DB2 utilities and commands
- Activities performed under system administrative and database administrative authorities

DB2 10 groups these events into *audit categories*. Audit categories were introduced in DB2 for LUW. The implementation in DB2 10 allows consistency through IBM DB2 server platforms.

In DB2 10, you can use the audit categories shown in Table 10-1 to define audit policies.

Table 10-1 Audit categories

Audit category	Description
CHECKING	Denied access attempts due to inadequate DB2 authorization (IFCID140) and RACF authentication failures (IFCID 83)
CONTEXT	Utility start, objects or phase change, or utility end (IFCID 23, 24, 25)
DBADMIN	Generates IFCID 361 trace records when an administrative authority, DBMAINT, DBCTRL, DBADM, PACKADM, SQLADM, System DBADM, DATAACCESS, ACCESSCTRL, or SECADM satisfies the privilege that is required to perform an operation. Database filtering can be performed through the DBNAME attribute for DBADM, DBCTRL, and DBMAINT authorities. All databases are audited if no database name is specified in the DBNAME Policy column. Collection ID filtering can be performed through the COLLID attribute for the PACKADM authority. All collection IDs are audited if no collection ID is specified in the COLLID column. If the DB2/RACF Access Control Authorization Exit is active, only the operations that are performed by the SECADM authority are audited.
EXECUTE	Trace record is generated for every unique table change or table read including the SQL statement ID. The SQL statement ID is used in the IFCID 143 and IFCID 144 trace records to record any changes or reads by the SQL statement identified in the IFCID 145 trace records. One audit policy can be used to audit several tables of a schema by specifying the table names with the SQL LIKE predicate (for example TBL%). Only tables residing in universal table space (UTS), classic partitioned table spaces, and segmented table spaces can be audited. Table aliases cannot be audited. An audit policy can audit clone tables and tables that implicitly created for LOBs or XML columns. Generates trace record for all unique table access in a unit of work. The qualifying tables are identified in policy columns OBJECTSCHEMA, OBJECTNAME and OBJECTTYPE. If the audit policy is started while an SQL query is running, access to the table will be audited for the next SQL query that accesses the table.
OBJMAINT	Table altered or dropped. Audit policy specifies the tables to be audited. One audit policy can be used to audit several tables of a schema by specifying the table names with the SQL LIKE predicate (for example TBL%). You can audit only tables defined in UTS, traditional partitioned table spaces, and segmented table spaces. An audit policy can also audit clone tables and tables that are implicitly created for XML columns. Audit object type to be specified in OBJECTTYPE column. Default OBJECTTYPE column value of blank means auditing of all supported object types.
SECMAINT	Grant and revoke privileges or administrative authorities (IFCID 141) and create and alter trusted contexts (IFCID 270) and roles. SECMAINT category also includes IFCID 271 (create, alter, drop row permissions and column masks).

Archived

Audit category	Description
SYSADMIN	Generates IFCID 361 trace records when an administrative authority, installation SYSADM, installation SYSOPR SYSCTRL, or SYSADM satisfies the privilege required to perform an operation. If the DB2/RACF Access Control Authorization Exit is active, only the operations that are performed by the installation SYSADM and installation SYSOPR authorities are audited.
VALIDATE	New or changed authorization IDs (IFCID 55, 83, 87, 169, 269, 312), establishment of new or user switching within a trusted connections

Additional information: For details about DB2 10 audit categories refer to *DB2 10 for z/OS Administration Guide,* SC19-2968.

Creating an audit policy in SYSIBM.SYSAUDITPOLICIES

You create an audit policy by inserting a row into the SYSIBM.SYSAUDITPOLICIES catalog table. Each row in SYSIBM.SYSAUDITPOLICIES represents an audit policy. An audit policy can be identified uniquely by its policy name.

The policy name is stored in the AUDITPOLICYNAME table column. This column is also the unique index key. SYSIBM.SYSAUDITPOLICIES provides one column for each audit category.

An audit category is selected for auditing by populating its corresponding category policy column. The columns of the SYSIBM.SYSAUDITPOLICIES table and its valid category column values are listed in Table 10-2.

Table 10-2 The SYSIBM.SYSAUDITPOLICIES catalog table

Column name	Data type	Description
ALTERDTS	TIMESTAMP NOT NULL WITH DEFAULT	Alter timestamp
AUDITPOLICYNAME	VARCHAR(128) NOT NULL	Policy name
CHECKING	CHAR(1) NOT NULL WITH DEFAULT	► A: Audit all failures ► blank: None IFCID 140
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	Name of package collid. If no collid specified all collids are audited. Applies only to authority PACKADM in category DBADMIN IFCID 83
CONTEXT	CHAR(1) NOT NULL WITH DEFAULT	► A: Audit all utilities ► blank: None IFICD 23, 24, 25
CREATEDTS	TIMESTAMP NOT NULL WITH DEFAULT	Insert timestamp
DB2START	CHAR(1) NOT NULL WITH DEFAULT	Automatic policy start at DB2 startup, except for DB2 restart light. Up to 8 policies can be started at DB2 startup. ► Y: Automatic start ► N: No automatic start

Column name	Data type	Description
DBADMIN	VARCHAR(128) NOT NULL WITH DEFAULT	Database administrative tasks. ► *: Audit all, ► B: System DBADM ► C: DBCTRL ► D: DBADM ► E: SECADM ► G: ACCESSCTRL ► K: SQLADM ► M: DBMAINT ► P: PACKADM ► T: DATAACCESS ► zero length: Audit none Can be a concatenated string of multiple parameters, IFCID 361
DBNAME	VARCHAR(24) NOT NULL WITH DEFAULT	Database name. If no database specified all databases are audited. Applies only to authorities DBADM, DBCTRL, DBMAINT authorities in column DBADMIN
EXECUTE	CHAR(1) NOT NULL WITH DEFAULT	Tables qualifying through OBJECTSCHEMA, OBJECTNAME, OBJECTTYPE will be audited, ► A: Audit any SQL activity of that application process ► C: Audit SQL insert, update, or delete of that application process ► blank: None Traces bind time information about SQL statements qualified by OBJECTSCHEMA, OBJECTNAME, OBJECTTYPE IFCID 143, 144, 145
OBJECTNAME	VARCHAR(128) NOT NULL WITH DEFAULT	Object name. Applies only to categories OBJMAINT, EXECUTE. Can be specified in SQL LIKE notation
OBJECTSCHEMA	VARCHAR(128) NOT NULL WITH DEFAULT	Schema name. Applies only to categories OBJMAINT, EXECUTE
OBJECTTYPE	CHAR(1) NOT NULL WITH DEFAULT	► C: Clone Table ► P: Implicit table for XML column ► T: Table ► Blank: All Applies only to OBJMAINT and EXECUTE categories
OBJMAINT	CHAR(1) NOT NULL WITH DEFAULT	► A: Audit alter or drop activities when a table qualifies through OBJECTSCHEMA, OBJECTNAME, and OBJECTTYPE ► blank: None IFCID 142
SECMAINT	CHAR(1) NOT NULL WITH DEFAULT	► A: Audit all ► blank: None IFCID 141 grant, revoke IFCID 270 trusted context created or altered

Archived

Column name	Data type	Description
SYSADMIN	VARCHAR(128) NOT NULL WITH DEFAULT	System administrative tasks: ► *: Audit all ► I: Install SYSADM ► L: SYSCTRL ► O: SYSOPR ► R: install SYSOPR ► S: SYSADM ► zero length: Audit none Can be a concatenated string of multiple parameters, IFCID 361
VALIDATE	CHAR(1) NOT NULL WITH DEFAULT	► A: Audit all ► blank: None Assignment or change of authorization ID: IFCID 55, 83, 87, 169, 312 Trusted context establishment or user switch: IFCID 269

Additional information: For details about all catalog table columns of SYSIBM.SYSAUDITPOLICIES. refer to *DB2 10 for z/OS SQL Reference,* SC19-2983.

Added and changed instrumentation facility component identifiers

Table 10-3 lists the instrumentation facility component identifiers (IFCIDs) that are added or changed to support DB2 audit policies and the administrative authorities. For more information regarding these IFCIDs, refer to Appendix A, “Information about IFCID changes” on page 615.

Table 10-3 New and changed IFCIDs

IFCID	Description
106	System parameters in effect Changed to account for the following DSNZPARMs: ► SECADM1 ► SECADM2 ► SECADM1_TYPE ► SECADM2_TYPE ► SEPARATE_SECURITY ► REVOKE_DEP_PRIVILEGES
140	Authorization failures Changed to include constants for new authorities
141	Explicit grants and revokes Changed to include constants for new authorities
143	Changed to include unique statement identifier
144	Changed to include unique statement identifier
145	Changed to trace the entire SQL statement and to include the unique statement identifier
361	Added to support auditing of administrative authorities
362	Added for auditing the start audit trace command This trace is started automatically when a start trace command with the AUDTPLCY keyword is issued. IFCID 362 documents whether an audit policy was started successfully

Additional information: For a complete list of DB2 10 IFCIDs, refer to *DB2 10 for z/OS What's New*, GC19-2985.

Policy name considerations

The SYSIBM.SYSAUDITPOLICIES catalog table has a unique key constraint defined on the AUDITPOLICYNAME column. If you try to insert a policy row using an AUDITPOLICYNAME value that already exists in the table, you receive SQLCODE -803 (duplicate key) as you would for any other user table. Thus, plan your naming standards for audit policy names carefully.

Create audit policy samples

We created three audit policies for use and reference throughout this chapter. Example 10-1 and Example 10-2 show the SQL insert statements that we used for policy creation.

Audit all access to the DB2R5.EMP and DB2R5.DEPT tables

Example 10-1 inserts a policy row into SYSIBM.SYSAUDITPOLICY. The table row provides the following attribute settings:

```
AUDITPOLICYNAME     AUDEMP
OBJSCHEMA           DB2R5
OBJNAME             EMP and DEPT
OBJTYPE             T to indicate table type
EXECUTE             A to audit all access
```

Example 10-1 Audit all SQL for multiple tables

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
(AUDITPOLICYNAME, OBJECTSCHEMA, OBJECTNAME, OBJECTTYPE, EXECUTE)
VALUES('AUDEMP','DB2R5','EMP','T','A');
INSERT INTO SYSIBM.SYSAUDITPOLICIES
(AUDITPOLICYNAME, OBJECTSCHEMA, OBJECTNAME, OBJECTTYPE, EXECUTE)
VALUES('AUDDEPT','DB2R5','DEPT','T','A');
```

Audit use of SYSADM authorities

Example 10-2 inserts the AUDSYSADMIN audit policy of the SYSADMIN category into the SYSIBM.SYSAUDITPOLICIES table. The table row has the following column settings:

```
AUDITPOLICYNAME     AUDSYSADMIN
SYSADMIN            S to audit SYSADMIN
```

Example 10-2 Audit policy for category SYSADMIN

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
   (AUDITPOLICYNAME, SYSADMIN) VALUES('AUDSYSADMIN','S');
```

Audit policy to start automatically during DB2 startup

Example 10-3 updates the AUDSYSADMIN audit policy in the SYSIBM.SYSAUDITPOLICIES table to mark the policy for automatic start during DB2 startup.

Example 10-3 Mark audit policy for automatic start during DB2 startup

```
UPDATE SYSIBM.SYSAUDITPOLICIES
  SET DB2START = 'Y'
  WHERE AUDITPOLICYNAME = 'AUDSYSADMIN';
```

Archived

Important: Only the install SYSADM and the SECADM authority have the privilege to insert, update, or delete rows in SYSIBM.SYSAUDITPOLICIES and only SECADM can grant these privileges to others.

If DSNZPARM SEPARATE_SECURITY is set to NO, the SYSADM authority implicitly holds the SECADM system privilege and can, therefore, perform any work that requires SECADM authority.

For simplicity, we refer only to the SECADM authority when we discuss the privilege of being able to modify SYSIBM.SYSAUDITPOLICIES as SECADM, install SYSADM, or SYSADM with DSNZPARM SEPARATE_SECURITY set to NO.

Secure Audit trace

APARs PM28296, PM26977, PM28543, PM30394, and PM32217 add support for secure audit policy trace and removal of some system authorities for DBADM.

A new value 'S' is added to SYSIBM.SYSAUDITPOLICIES - DB2START column. If 'S' is specified, then the audit policy will be automatically started at DB2 start up and can only be stopped by user with SECADM authority or will be stopped at DB2 stop.

- If multiple audit policies are specified to be started at DB2 start up, some with DB2START = 'Y' and some with DB2START = 'S', then two traces will be started, one for audit policies with DB2START='Y' and another for audit policies with DB2START='S'. To stop the audit policies started at DB2 start up, all the audit policies that are assigned the same trace number must be stopped simultaneously. Then the policies can be started individually, as needed.
- After adding the entry in SYSAUDITPOLICIES with DB2START='S', the trace can be started to get this behavior.
- Any user with necessary privilege can start the trace specified with DB2START = 'S' and the SECADM restriction applies only during stop trace.
- If the audit policy has already been started, then after updating the DB2START column, the audit policy needs to be stopped and restarted to get the secure behavior.

Starting and stopping DB2 audit policies

Policy-based auditing is operated using start, stop, and display audit trace commands. In DB2 10, these commands are enhanced to support an AUDTPLCY parameter (see Figure 10-1). Using the policy name to manage audit traces provides additional simplification, because DB2 chooses the IFCID information that is appropriate for the selected audit policy.

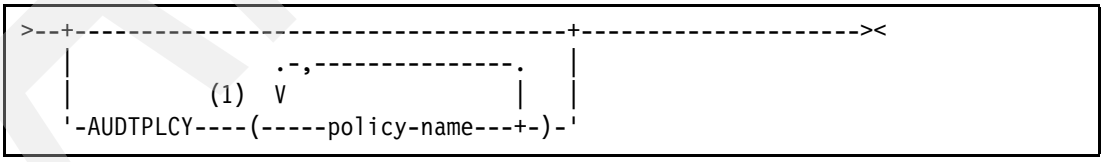


Figure 10-1 Start audit trace parameter AUDTPLCY

The AUDTPLCY parameter can take multiple policy names. We recommend to provide as many audit policies as possible in the AUDPLCY keyword, because DB2 starts only one trace for all policies that are specified, which helps to avoid the system limit of up to 32 started traces. However, you cannot stop an individual audit policy if that policy was started with other policies within the AUDTPLCY keyword. AUDTPLCY keyword is mutually exclusive with the IFCID or CLASS keyword.

In the scenario illustrated in Figure 10-2, we started the two audit policies that we defined in Example 10-1 on page 343 within one single start trace command. We then tried to stop just the AUDEMP audit policy, and the command failed with DB2 message DSNW137I.

```
-START TRACE(AUDIT) AUDTPLCY(AUDEMP,AUDDEPT)
....
-STOP  TRACE(AUDIT) AUDTPLCY(AUDEMP)
DSNW137I  -DB0B  SPECIFIED TRACE NOT ACTIVE
DSN9022I  -DB0B DSNWVCM1 '-STOP TRACE' NORMAL COMPLETION
-STOP  TRACE(AUDIT) AUDTPLCY(AUDEMP,AUDDEPT)
DSNW131I  -DB0B STOP TRACE SUCCESSFUL FOR TRACE NUMBER(S) 03
DSN9022I  -DB0B DSNWVCM1 '-STOP TRACE' NORMAL COMPLETION
-START TRACE(AUDIT) AUDTPLCY(AUDDEPT) DEST(GTF)
DSNW130I  -DB0B AUDIT TRACE STARTED, ASSIGNED TRACE NUMBER 03
DSNW192I  -DB0B AUDIT POLICY SUMMARY
AUDIT POLICY AUDDEPT STARTED
END AUDIT POLICY SUMMARY
DSN9022I  -DB0B DSNWVCM1 '-START TRACE' NORMAL COMPLETION
```

Figure 10-2 Use of AUDTPLCY in start, stop trace commands

Start, display, and stop audit trace samples

We ran the following commands to start, display, and stop the audit policies:

- Start the AUDSYSADMIN audit policy successfully for the DB0BSYSADM role after the policy is defined (see Figure 10-3). The audit trace records are written to destination GTF, which requires the GTF trace for DB2 to be started.

```
-START   TRACE(AUDIT) AUDTPLCY(AUDSYSADMIN) DEST(GTF) ROLE(DB0BSYSADM)
DSNW130I  -DB0B AUDIT TRACE STARTED, ASSIGNED TRACE NUMBER 04
DSNW192I  -DB0B AUDIT POLICY SUMMARY
AUDIT POLICY AUDSYSADMIN STARTED
END AUDIT POLICY SUMMARY
DSN9022I  -DB0B DSNWVCM1 '-START TRACE' NORMAL COMPLETION
```

Figure 10-3 Start AUDSYSADMIN audit policy

- Start multiple audit policies by specifying multiple audit policies in the start trace AUDTPLCY keyword (see Figure 10-4).

```
-START TRACE(AUDIT) AUDTPLCY(AUDEMP,AUDDEPT) DEST(GTF)
DSNW130I  -DB0B AUDIT TRACE STARTED, ASSIGNED TRACE NUMBER 04
DSNW192I  -DB0B AUDIT POLICY SUMMARY
AUDIT POLICY AUDEMP STARTED
AUDIT POLICY AUDDEPT STARTED
END AUDIT POLICY SUMMARY
DSN9022I  -DB0B DSNWVCM1 '-START TRACE' NORMAL COMPLETION
```

Figure 10-4 Start multiple audit trace policies with one start trace command

- Display status of the audit policy that was activated in Figure 10-3. The DISPLAY command was issued with detail level 1 and 2 to provide all detail information (see Figure 10-5). The DISPLAY TRACE output shows the name of the audit policy that we started and the role name for which the trace data is filtered.

```
-DISPLAY TRACE(AUDIT) AUDTPLCY(AUDSYSADMIN) DETAIL(1,2)
DSNW127I  -DBOB CURRENT TRACE ACTIVITY IS -
TNO TYPE   CLASS        DEST QUAL IFCID
04  AUDIT  *            GTF  YES
*********END OF DISPLAY TRACE SUMMARY DATA*********
DSNW143I  -DBOB CURRENT TRACE QUALIFICATIONS ARE -
DSNW152I  -DBOB BEGIN TNO 04 QUALIFICATIONS:
ROLE: DBOBSYSADM
END TNO 04 QUALIFICATIONS
DSNW185I  -DBOB BEGIN TNO 04 AUDIT POLICIES:
ACTIVE AUDIT POLICY: AUDSYSADMIN
END TNO 04 AUDIT POLICIES
DSNW148I  -DBOB ******END OF DISPLAY TRACE QUALIFICATION DATA******
DSN9022I  -DBOB DSNWVCM1 '-DISPLAY TRACE' NORMAL COMPLETION
```

Figure 10-5 Display audit policy AUDSYSADM

- Stop the audit trace activated in Figure 10-3 on page 345. The STOP AUDIT TRACE command specifically stops the AUDSYSADMIN audit policy (see Figure 10-6).

```
-STOP    TRACE(AUDIT)  AUDTPLCY(AUDSYSADMIN    )
DSNW131I  -DBOB STOP TRACE SUCCESSFUL FOR TRACE NUMBER(S) 03
DSN9022I  -DBOB DSNWVCM1 '-STOP TRACE' NORMAL COMPLETION
```

Figure 10-6 Stop audit policy AUDSYSADMIN

As shown in Figure 10-5 on page 346 and in Figure 10-6, using the AUDTPLCY parameter is sufficient to display and stop the DB2 audit policies. You do not need to provide the trace number that was assigned by DB2 during TRACE START.

Error situations

DB2 10 introduces reason codes and messages that are related to audit policy processing. These codes and messages can assist in problem assessment and determination. When we tested our audit policy scenarios, we experienced the following common error situations:

- Start the AUDSYSADMINFAIL audit policy. Because the AUDSYSADMINFAIL audit policy contains an invalid parameter setting, DB2 returns error reason code 00E70022, as shown in Figure 10-7.

```
-START TRACE(AUDIT)  AUDTPLCY(AUDSYSADMINFAIL) DEST(GTF)
DSNW192I  -DBOB AUDIT POLICY SUMMARY
AUDIT POLICY AUDSYSADMINFAIL NOT STARTED, REASON 00E70022
END AUDIT POLICY SUMMARY
DSN9023I  -DBOB DSNWVCM1 '-START TRACE' ABNORMAL COMPLETION
```

Figure 10-7 Start AUDTPLCY reason code 00E70022

- Start the AUDSYSADMIN audit policy created in Example 10-2 on page 343. Because the AUDSYSADMIN audit policy is not defined, the START command fails with reason code 00E70021, as shown in Figure 10-8.

```
-START TRACE(AUDIT)  AUDTPLCY(AUDSYSADMIN    ) DEST(GTF)
DSNW192I  -DBOB AUDIT POLICY SUMMARY
AUDIT POLICY AUDSYSADMIN NOT STARTED, REASON 00E70021
END AUDIT POLICY SUMMARY
DSN9023I  -DBOB DSNWVCM1 '-START TRACE' ABNORMAL COMPLETION
```

Figure 10-8 Start AUDTPLCY reason code 00E70021

- Start the AUDPHONE and AUDSYSADMIN audit policies. The AUDPHONE audit policy references the PHONENO table. The START command fails with reason code 00E70024, because the table referenced in the PHONENO table does not exist. DB2 does not verify the existence of a table when you insert an audit policy into SYSIBM.SYSAUDITPOLICIES. See Figure 10-9.

```
-START TRACE(AUDIT) AUDTPLCY(AUDPHONE,AUDSYSADMIN) DEST(SMF)
DSNW130I  -DBOB AUDIT TRACE STARTED, ASSIGNED TRACE NUMBER 03
DSNW192I  -DBOB AUDIT POLICY SUMMARY
AUDIT POLICY AUDSYSADMIN STARTED
AUDIT POLICY AUDPHONE NOT STARTED, REASON 00E70024
END AUDIT POLICY SUMMARY
DSN9022I  -DBOB DSNWVCM1 '-START TRACE' NORMAL COMPLETION
```

Figure 10-9 Start AUDTPLCY reason code 00E70024

Auditing start audit trace commands

DB2 10 externalizes an IFCID 362 trace record each time an audit policy starts or fails. In case a start policy command fails, DB2 issues the DSNW196I warning message. For example, a start trace AUDTPLCY command can fail when the SYSIBM.SYSAUDITPOLICY table row contains an invalid value in any of the category columns or if you try to start a policy that is not yet created (as shown in Figure 10-7 on page 346 and Figure 10-8 on page 347).

Figure 10-10 shows the sample report of an IFCID 362 trace record formatted by IBM Tivoli OMEGAMON XE for DB2 Performance Expert Version V5R1 (OMEGAMON PE).

```
SYSOPR  DBOB   C66CDC2F49AA 'BLANK'     'BLANK'            'BLANK'
SYSOPR   022.AUDT 'BLANK'     00:14:05.66735526     48   1 362 START/STOP TRC NETWORKID:
'BLANK' BL01          N/P                  AUDITPOLICY

        STATUS          : SUCCESS            TYPE           : START TRACE
     DB2 START UP   : 'BLANK'        DATABASE NAME  : 'BLANK'
      AUDIT CATEGORIES : EXECUTE
      AUDIT POLICY NAME: AUDDEPT
      TABLE SCHEMA NAME: DB2R5
          TABLE NAME       :  'AUDDEPT'
```

Figure 10-10 OMEGAMON PE IFCID 362 RECTRACE report

Additional information: For information about DB2 start, stop, and display trace command options, refer to *DB2 10 for z/OS Command Reference,* SC19-2972.

Changing an audit policy

You can change an existing audit policy using an SQL update while an audit trace is active for that policy. However, updating the policy using SQL does not change the audit trace criteria that was selected by DB2 during the audit trace start. To reflect policy changes in the audit trace, use the DB2 command interface to stop and start the audit policy.

You can keep track of audit policy changes by having an appropriate audit policy started. In our DB2 10 environment, we ran an update on SYSIBM.SYSAUDITPOLICY under SYSADM authority while the audit policy AUDSYSADMIN was started. Because of this policy, DB2 wrote an IFCID 361 trace record as a result of the update statement in Example 10-3 on page 343.

Figure 10-11 shows the OMEGAMON PE record trace report that we created for IFCID 361.

```
PRIMAUTH CONNECT  INSTANCE   END_USER     WS_NAME
ORIGAUTH CORRNAME CONNTYPE   RECORD TIME  DESTNO ACE IFC DESCRIPTION
PLANNAME CORRNMBR            TCB CPU TIME           ID
-------- -------- ---------- ------------ ------ --- --- -----------
DB2R5    DB2R5X   TSO        17:36:53.164   421    3 361 AUDIT ADMIN
DSNTEP10 'BLANK'             N/P                        AUTHORITIES

        AUTHORITY TYPE   : SYSADM          AUTHID TYPE       : AUTHORIZATION ID
        PRIVILEGE CHECKED:  53 PRIVILEGE : UPDATE OBJECT TYPE : TABLE OR VIEW
        AUTHORIZATION ID :  DB2R5
        SOURCE OBJ QUALIF:  SYSIBM
        SOURCE OBJ NAME  :  SYSAUDITPOLICIES

        SQL STATEMENT:
        UPDATE SYSIBM.SYSAUDITPOLICIES SET EXECUTE = 'A'
         WHERE AUDITPOLICYNAME IN ('AUDEMP','AUDDEPT')
```

Figure 10-11 OMEGAMON PE IFCID 361 record trace

Reason codes

To handle error situations that can occur during start, stop, and display audit trace commands using the new AUDTPLCY keyword, DB2 10 introduces reason codes. For details about these reason codes, refer to *DB2 10 for z/OS Codes,* GC19-2971.

Messages

DB2 10 introduces message IDs that are related to audit policy processing. These messages can assist you in problem determination and in system automation. For details about these messages for DB2 audit policies, refer to *DB2 10 for z/OS Messages,* GC19-2979.

10.1.2 Authorization

The following privileges are required to manage the policy-based auditing process:

- INSERT privilege on table SYSIBM.SYSAUDITPOLICY for defining the policy
- TRACE privilege for activating the policy using the START AUDIT TRACE command

In DB2 10, both privileges are implicitly held by the SECADM authority, which simplifies DB2 auditing because only one authority is required for audit policy management and activation. Because the SECADM authority can grant the privilege to perform inserts on the SYSIBM.SYSAUDITPOLICY catalog table, the management task of defining policies can be delegated to other users, for example to users with TRACE authority. The SECADM authority has the implicit privilege to issue the start, stop, and display trace commands. Any user who

has the necessary privileges to issue DB2 commands can specify the AUDTPLCY keyword on the start, stop, and display trace commands.

The following authorities implicitly hold select privilege on the SYSIBM.SYSAUDITPOLICY catalog table:

- ACCESSCTRL
- DATAACCESS
- System DBADM
- SQLADM
- SYSCTRL
- SYSADM

10.1.3 Creating audit reports

With DB2 10, you can use OMEGAMON PE or other third-party report writer to format the auditing IFCID records for auditing. If you prefer to use a third-party report writer, check with your ISV for compatibility with DB2 10 for z/OS. In our environment, we set up the JCL sample shown in Figure 10-12 to create reports with OMEGAMON PE V5R1.

```
//DB2R5PM   JOB (999,POK),'JOSEF',CLASS=A,
// MSGCLASS=T,NOTIFY=&SYSUID,REGION=0M
/*JOBPARM S=SC63,L=9999
//DB2PM   EXEC PGM=DB2PM,REGION=0K
//STEPLIB  DD DISP=SHR,DSN=OMPE.V510.BETA.TKANMOD
//INPUTDD  DD DISP=SHR,DSN=DB2R5.GTFDB2.SC63.D100809.T203916
//SYSIN DD *
AUDIT
       REPORT
        LEVEL(DETAIL)
        TYPE(ALL)
       TRACE
        TYPE(ALL)
  RECTRACE
       TRACE
        LEVEL(LONG)
  EXEC
/*
```

Figure 10-12 OMEGAMON PE V1R5 JCL sample

10.1.4 Policy-based SQL statement auditing for tables

DB2 10 implements policy-based SQL statement auditing of tables. The table audit capability enables you to dynamically enable SQL statement auditing for tables that do or do not have the table AUDIT clause set in their table definition.

When table-based SQL statement auditing is active, DB2 writes IFCID 143, 144, and 145 trace records for every SQL statement that performs changes on (SQL insert, update, or delete) or reads from (SQL select) tables that are identified in the audit policy. IFCID 145 records bind time information, including the full SQL statement text and a unique SQL statement ID. The SQL statement ID is used in the IFCID 143 and IFCID 144 trace records to record any changes or reads that are performed by the SQL statement identified in the IFCID 145 trace records.

Archived

For the tables identified in the policy, DB2 writes one audit record for each SQL statement ID accessing the table. DB2 writes additional audit records when a table identified by the audit policy is read or changed using a different SQL statement ID.

You create and manage the audit policy for SQL statement auditing for tables using the interfaces that we described in 10.1.1, “Audit policies” on page 338.

DB2 10 introduces the following IFCID changes to support the new SQL statement ID based auditing for tables:

- IFCID 143 includes a unique statement identifier.
- IFCID 144 includes a unique statement identifier.
- IFCID 145 traces the entire SQL statement text and include a unique statement identifier.

For dynamic SQL statements, the unique statement identifier (STMT_ID) is derived from the dynamic statement cache, and for embedded static SQL statements, the STMT_ID is derived from the SYSIBM.SYSPACKSTMT.STMT_ID catalog table column.

You cannot use the table-based auditing with tables that are defined in simple table spaces.

Audit policy for table-based SQL auditing

If you want to activate table-based auditing for SQL statements, you need to create an audit policy for the EXECUTE category. Additionally, you need to provide further policy information to specify the object type, object name, and the schema name that you want to audit. The following SYSIBM.SYSAUDITPOLICY columns are relevant to the audit category EXECUTE:

- AUDITPOLICYNAME
- OBJECTSCHEMA
- OBJECTNAME
- OBJECTTYPE
- EXECUTE

For more information about these columns, refer to Table 10-2 on page 340.

Sample scenario

We prepared a sample scenario with two sample tables, an embedded SQL program and a small dynamic SQL workload, to illustrate table-based auditing. The sample scenario contained the following objects:

- Tables DB2R5.AUDEMP and DB2R5.AUDDEPT
- Dynamic SQL

  The dynamic SQL workload shown in Example 10-4 executes three different SQL statements within one unit of work. The SQL statements access the same table. We, therefore, see three IFCID 143 records within one unit of work, one for each distinct SQL DML statement.

Example 10-4 AUD dynamic SQL for auditing

```
INSERT INTO DB2R5.AUDEMP
 VALUES (   '300000' , 'JOSEF' , ' ' , 'KLITSCH' , 'X00' , '1234' ,
'1998-08-29','ITJOB',42,'M','1958-09-13',99.99,99.99,578);
INSERT INTO DB2R5.AUDEMP
 VALUES (   '300001' , 'JOSEF' , ' ' , 'KLITSCH' , 'X00' , '1234' ,
'1998-08-29','ITJOB',42,'M','1958-09-13',99.99,99.99,578);
DELETE FROM DB2R5.AUDEMP WHERE EMPNO >= 300000';
COMMIT  ;
```

- Static SQL COBOL program: AUDSQL

  The COBOL program AUDSQL executes the static SQL statements shown in Example 10-5. The program contains two different select statements, and each statement is executed three times within each unit of work. Therefore, we see two IFCID 144 records for each unit of work, one for each distinct SQL statement.

Example 10-5 AUD SQL static SQL for auditing

```
PERFORM 3 TIMES
   PERFORM 3 TIMES
         EXEC SQL
              SELECT COUNT(*) INTO :V-EMP FROM DB2R5.AUDDEPT
         END-EXEC
         EXEC SQL
              SELECT COUNT(*) INTO :V-EMP FROM DB2R5.AUDEMP
         END-EXEC
   END-PERFORM
   EXEC SQL COMMIT END-EXEC
END-PERFORM.
```

Creating an EXECUTE category audit policy

Within this activity, we executed the SQL statements shown in Example 10-6 to create an audit policy to audit any SQL activity on tables DB2R5.AUD%. We set OBJECTNAME to AUD%, which causes DB2 to use a like predicate when determining the tables to be audited. Example 10-6 inserts an audit policy to audit tables names that begin with AUD% within schema DB2R5.

Example 10-6 Create SQL statement auditing policy

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
(AUDITPOLICYNAME, OBJECTSCHEMA, OBJECTNAME, OBJECTTYPE, EXECUTE)
VALUES('AUDTABLES','DB2R5','''AUD%''','T','A');
```

Starting the audit policy

Under an authority with TRACE privilege, we successfully started the audit policy for table based SQL auditing (see Example 10-7).

Example 10-7 Start SQL statement auditing policy

```
-START TRACE(AUDIT) AUDTPLCY(AUDTABLES) DEST(GTF)
DSNW130I  -DBOB AUDIT TRACE STARTED, ASSIGNED TRACE NUMBER 03
DSNW192I  -DBOB AUDIT POLICY SUMMARY
AUDIT POLICY AUDTABLES STARTED
END AUDIT POLICY SUMMARY
DSN9022I  -DBOB DSNWVCM1 '-START TRACE' NORMAL COMPLETION
```

Displaying audit policy status

If you are not sure an audit policy is started, you can use the DISPLAY TRACE command with the AUDTPLCY parameter to verify that audit policy started. To run the DISPLAY TRACE command, you must have the DISPLAY TRACE privilege, which is included in the system DBADM, the SYSOPR, the SYSCTRL, the SYSADM, and in the SECADM authority. Example 10-8 shows the output of the DISPLAY TRACE command with the AUDTPLCY parameter.

Example 10-8 Display SQL statement audit policy

```
-DIS   TRACE(AUDIT) AUDTPLCY(AUDTABLES) DETAIL(1,2)
DSNW127I  -DBOB CURRENT TRACE ACTIVITY IS -
TNO TYPE   CLASS        DEST QUAL IFCID
03  AUDIT  *            GTF  NO
*********END OF DISPLAY TRACE SUMMARY DATA*********
DSNW143I  -DBOB CURRENT TRACE QUALIFICATIONS ARE -
DSNW152I  -DBOB BEGIN TNO 03 QUALIFICATIONS:
NO QUALIFICATIONS
END TNO 03 QUALIFICATIONS
DSNW185I  -DBOB BEGIN TNO 03 AUDIT POLICIES:
ACTIVE AUDIT POLICY: AUDTABLES
END TNO 03 AUDIT POLICIES
DSNW148I  -DBOB ******END OF DISPLAY TRACE QUALIFICATION DATA******
DSN9022I  -DBOB DSNWVCM1 '-DIS TRACE' NORMAL COMPLETION
```

Stop audit policy

After successful SQL workload execution, we ran the command in Example 10-9 to stop the audit policy. The command requires TRACE or SECADM authority.

Example 10-9 Stop SQL statement audit policy

```
-STOP  TRACE(AUDIT) AUDTPLCY(AUDTABLES)
DSNW131I  -DBOB STOP TRACE SUCCESSFUL FOR TRACE NUMBER(S) 03
DSN9022I  -DBOB DSNWVCM1 '-STOP TRACE' NORMAL COMPLETION
```

Archived

Reports for SQL auditing

In this section, we describe how we used OMEGAMON PE to format the IFCID traces that we collected within our sample scenario.

OMEGAMON PE record trace for static SQL

Figure 10-13 shows the OMEGAMON PE record trace report that we created for our static SQL sample workload. The report shows six IFCID 144 trace records (two for each DB2 unit of work) that DB2 collected because of the SQL workload shown in Example 10-5 on page 351.

```
DB2R5    BATCH    C681E3C79388 'BLANK'          'BLANK'                      'BLANK'
DB2R5    DB2R5R06 TSO         17:39:53.41335706      9   1 361 AUDIT ADMIN  NETWORKID:  USIBMSC
AUDSQL   'BLANK'              N/P                              AUTHORITIES

AUTHORITY TYPE   : SYSADM       AUTHID TYPE : AUTHORIZATION ID
PRIVILEGE CHECKED:         64   PRIVILEGE   : EXECUTE          OBJECT TYPE : APPLICATION PLAN
AUTHORIZATION ID :  DB2R5
SOURCE OBJ NAME  :  AUDSQL

SQL STATEMENT: 'BLANK'
................................................................................
17:39:53.41637954  10   1 144 AUDIT FIRST    'BLANK'
N/P                           READ          NETWORKID:  USIBMSC   LUNAME:  SCPDBOB   LUWSEQ:  1
                                            DATABASE: 285             LOGRBA: X'000000000000'
                                            PAGE SET: 222                     TABLE OBID:   223
                                            STMT ID :                 24682
17:39:53.47192559  11   1 144 AUDIT FIRST    'BLANK'
N/P                           READ          NETWORKID:  USIBMSC   LUNAME:  SCPDBOB   LUWSEQ: 1
                                            DATABASE: 285             LOGRBA: X'000000000000'
                                            PAGE SET: 219                     TABLE OBID:  220
                                            STMT ID :                 24683
17:39:53.50356837  12   1 144 AUDIT FIRST    'BLANK'
N/P                           READ          NETWORKID:  USIBMSC   LUNAME:  SCPDBOB   LUWSEQ: 2
                                            DATABASE: 285             LOGRBA: X'000000000000'
                                            PAGE SET: 222                     TABLE OBID:  223
                                            STMT ID :                 24682
17:39:53.50384456  13   1 144 AUDIT FIRST    'BLANK'
N/P                           READ          NETWORKID:  USIBMSC   LUNAME:  SCPDBOB   LUWSEQ: 2
                                            DATABASE: 285             LOGRBA: X'000000000000'
                                            PAGE SET: 219                     TABLE OBID:  220
                                            STMT ID :                 24683
17:39:53.50402484  14   1 144 AUDIT FIRST    'BLANK'
N/P                           READ          NETWORKID:  USIBMSC   LUNAME:  SCPDBOB   LUWSEQ: 3
                                            DATABASE: 285             LOGRBA: X'000000000000'
                                            PAGE SET: 222                     TABLE OBID:  223
                                            STMT ID :                 24682
17:39:53.50428398  15   1 144 AUDIT FIRST    'BLANK'
N/P                           READ          NETWORKID:  USIBMSC   LUNAME:  SCPDBOB   LUWSEQ: 3
                                            DATABASE: 285             LOGRBA: X'000000000000'
                                            PAGE SET: 219                     TABLE OBID:  220
                                            STMT ID :                 24683
```

Figure 10-13 OMEGAMON PE record trace for static SQL

Archived

OMEGAMON PE record trace for dynamic SQL

Example 10-10 shows the OMEGAMON PE record trace report that we created for our dynamic SQL sample workload. The report shows three IFCID 143 trace records that DB2 collected because of the SQL workload shown in Example 10-4 on page 350. We expected this output because we executed the SQL DML statements within one DB2 unit of work.

Example 10-10 OMEGAMON PE record trace for dynamic SQL

```
DB2R5    BATCH    C68208FEC3A1 'BLANK'          'BLANK'                    'BLANK'
DB2R5    DB2R5X   TSO
DSNTEP10 'BLANK'
     20:26:23.40079720     83   1 143 AUDIT FIRST    NETWORKID:  USIBMSC   LUNAME:  SCPDBOB   LUWSEQ:     1
     N/P                             WRITE        DATABASE: 285             LOGRBA: X'000031D4D1F2'
                                                  PAGE SET: 222                     TABLE OBID:      223
                                                  STMT ID :                  19
     20:26:23.40120014     84   1 143 AUDIT FIRST    'BLANK'
     N/P                             WRITE        NETWORKID:  USIBMSC   LUNAME:  SCPDBOB   LUWSEQ:     1
                                                  DATABASE: 285             LOGRBA: X'000031D4D1F2'
                                                  PAGE SET: 222                     TABLE OBID:      223
                                                  STMT ID :                  20
     20:26:23.40152556     85   1 143 AUDIT FIRST    'BLANK'
     N/P                             WRITE        NETWORKID:  USIBMSC   LUNAME:  SCPDBOB   LUWSEQ:     1
                                                  DATABASE: 285             LOGRBA: X'000031D4D1F2'
                                                  PAGE SET: 222                     TABLE OBID:      223
                                                  STMT ID :                  21
```

10.1.5 Summary

The auditing capability provided by DB2 10 enables you to dynamically activate audit policies without having to alter objects in DB2. DB2 10 also simplifies the management of auditable events by introducing audit policies and by providing the ability to group auditable events into audit categories. Audit policies are stored in the SYSIBM.SYSAUDITPOLICY catalog table, which helps to simplify auditing processes and procedures. Storing the audit policies in this table also makes the audit policy definitions available to all parties involved in administering and managing audit policies throughout a DB2 subsystem or, if applicable, throughout all members of a data sharing group.

The DB2 10 interface for starting and stopping audit policies is based entirely on using audit policies to be stored in the SYSIBM.SYSAUDITPOLICY table. For example, you can start an audit policy using the AUDTPLCY keyword in the START AUDIT TRACE command. You do not need to specify IFCIDs or trace classes. During the audit trace start, DB2 fetches the audit policy from the catalog and determines the IFCIDs that it needs to collect to satisfy the auditing request.

10.2 More granular system authorities and privileges

DB2 10 provides additional system authorities and privileges, as illustrated in Figure 10-14. These authorities and privileges are designed to help business to comply with government regulations and to simplify the management of authorities. DB2 10 introduces the concepts of separation of duties and least privilege to address these needs.

Separation of duties provides the ability for administrative authorities to be divided across individuals without overlapping responsibilities, so that one user does not possess unlimited authority (for example SYSADM).

In DB2 10, the system DBADM authority allows an administrator to manage all databases in a DB2 subsystem. By default, the system DBADM has all the privileges of the DATAACCESS and ACCESSCTRL authorities.

- If you do not want a user with the system DBADM authority to GRANT any explicit privileges, you can specify the WITHOUT ACCESSCTRL clause in the GRANT statement when you grant the authority.
- If you do not want a user with the system DBADM authority to access any user data in the databases, you can specify the WITHOUT DATAACCESS clause in the GRANT statement when you grant the authority. However, even when an authid or role has SYSTEM DBADM WITHOUT DATAACCESS, explicit privileges (such as SELECT) can be still be granted to this authid or role.

The SECADM authority provides the ability to manage access to tables in DB2 while not holding the privilege to create, alter, drop, or access a table. Furthermore, a DATAACCESS authority can access all user tables without being able to manage security or database objects.

You can use these authorities to minimize the need for the SYSADM authority in day-to-day operations by delegating security administration, system related database administration, and database access duties onto the SECADM, system DBADM, and DATAACCESS authorities. To go even further, you can separate security administration from the SYSADM authority so that SYSADM can no longer perform security-related tasks.

The least privilege concept allows an administrator to delegate part of his responsibilities to other users, without providing extraneous privileges. Under this model, database administrators can share administrative duties while ensuring a high level of security.

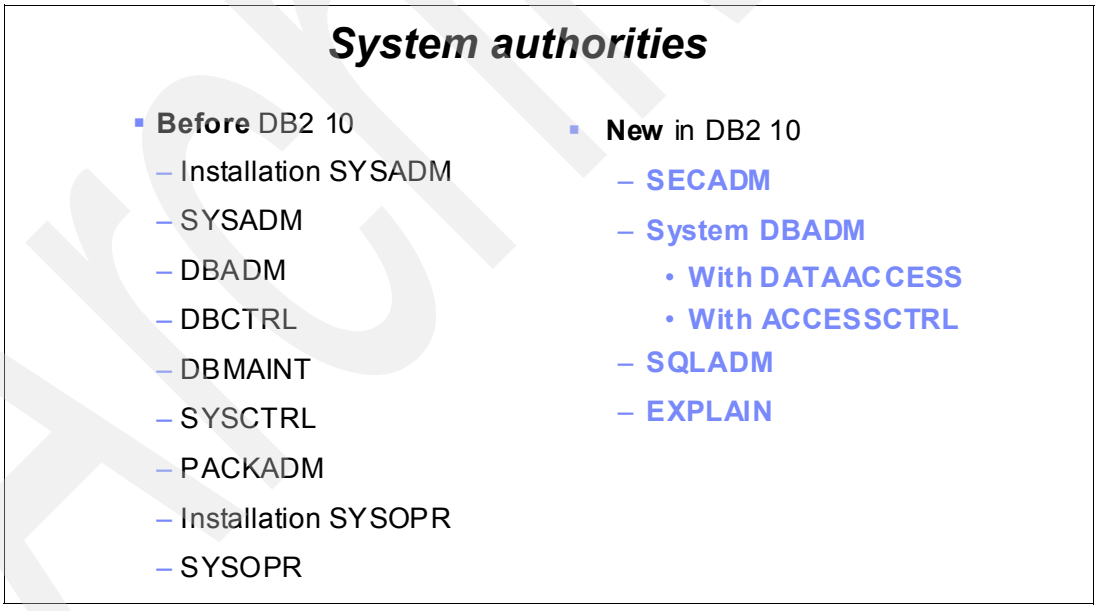


Figure 10-14 System authorities

Archive

10.2.1 Separation of duties

Separation of duties is the ability to distribute administrative authorities to different users without overlapping of responsibilities. DB2 10 implements separation of duties by providing the following auditable system administrative authorities:

- SYSADM can now be set up to no longer manage access to secure objects, such as roles or the ability to grant and revoke authorities or privileges.
- The SECADM authority performs all security-related tasks against a DB2 subsystem, such as managing security-related objects and the ability to grant and revoke authorities and privileges.
- The system DBADM authority manages objects but can be granted without access to user data or the ability to grant and revoke authorities and privileges.
- The DATAACCESS authority to controls database administrator access to user data in DB2.

10.2.2 Least privilege

Least privilege allows an administrator to grant users only the privilege that is required to perform a specific task. DB2 10 introduces privileges to help protect business-sensitive data from users who should only be able to perform administrative tasks that do not require access to user data. In that context DB2 10 introduces the following new privileges:

- The EXPLAIN privilege to issue EXPLAIN, PREPARE, and DESCRIBE statements without requiring the privilege to execute the SQL statements.
- The SQLADM authority to monitor and tune SQL without any additional privileges.
- The ACCESSCTRL authority that allows a SECADM authority to delegate the ability to grant and revoke object privileges and most administrative authorities.

10.2.3 Grant and revoke system privilege changes

Except for the SECADM authority, the administrative authorities are system privileges that can be granted and revoked in DB2. In DB2 10, the interface to grant and revoke system privileges is extended to support these new authorities. Because the SECADM authority is not recorded in the catalog, you cannot grant and revoke that privilege like other system privileges (refer to 10.2.5, “SECADM authority” on page 359 for more information).

Archive

Figure 10-15 shows the syntax of the grant system privilege SQL.

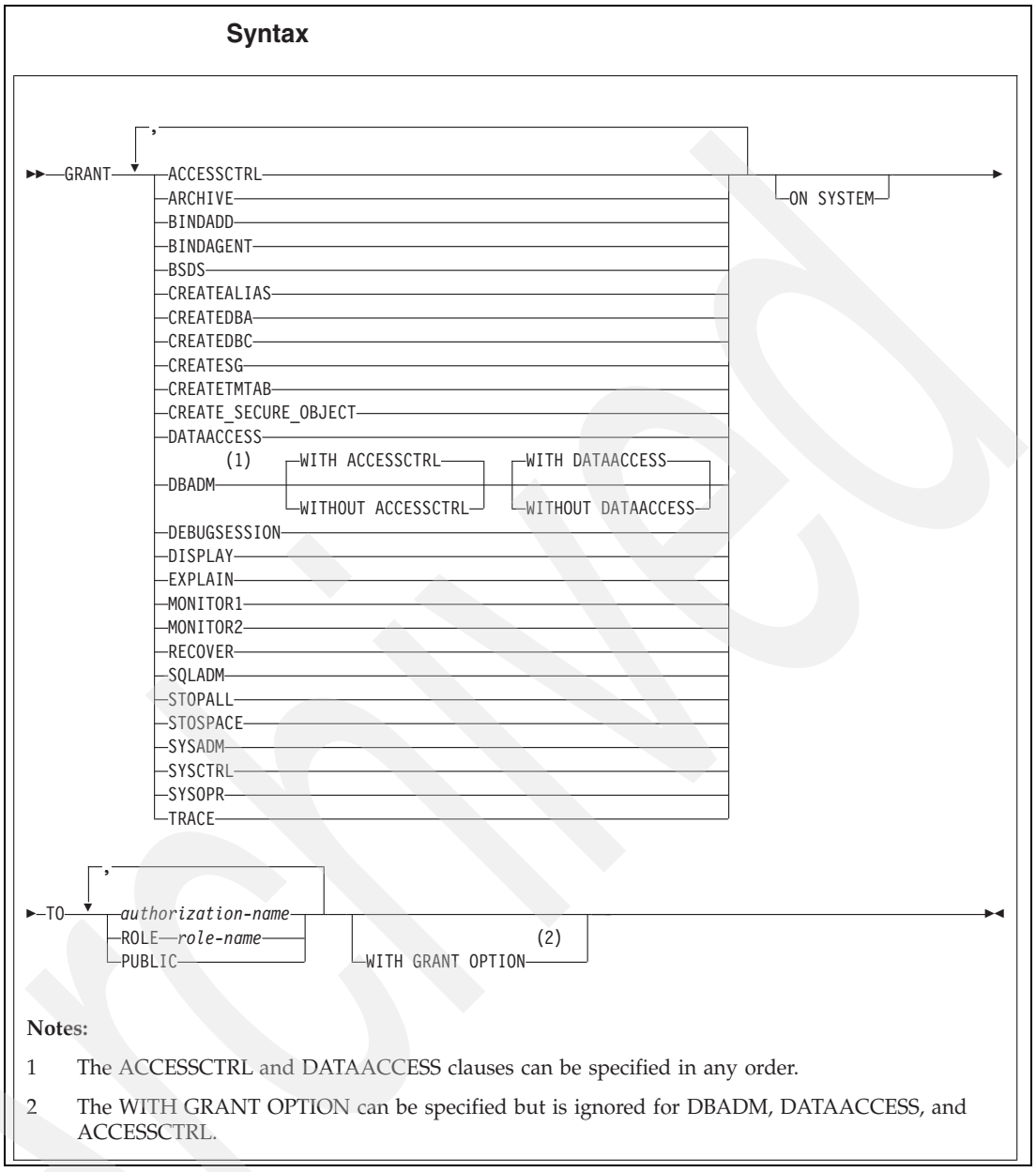


Figure 10-15 SQL syntax grant system

Note that the WITH GRANT OPTION is also ignored for CREATE_SECURE_OBJECT privilege.

Archive

Revoke system privileges

In DB2 10, the SQL interface for revoking system privileges is changed to support the ACCESSCTRL, DATAACCESS, system DBADM, EXPLAIN, and SQLADM authorities. Figure 10-16 shows the syntax of the revoke system privilege SQL.

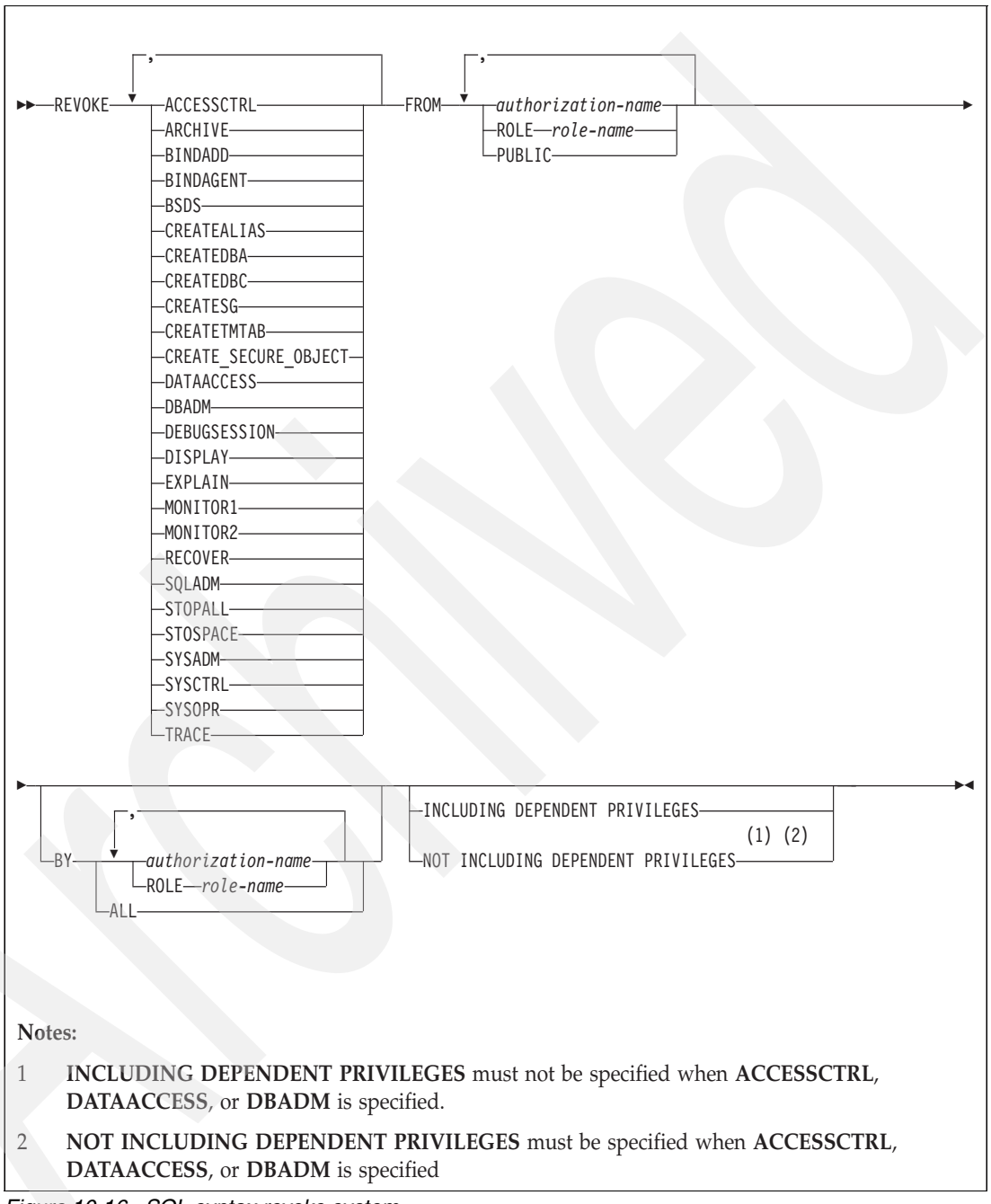


Figure 10-16 SQL syntax revoke system

Archive

10.2.4 Catalog changes

Except for the SECADM authority, these new system administrative authorities are, like any other DB2 system privilege, recorded in the SYSIBM.SYSUSERAUTH catalog table.

The AUTHHOWGOT catalog column tells the authority by which the privilege was granted. In DB2 10, column values for the AUTHOWGOT column indicate whether a privilege was granted by the SECADM or ACCESSCTRL authority.

The SYSIBM.SYSUSERAUTH catalog table

The SYSIBM.SYSUSERAUTH catalog table records the system privileges that are held by users. In DB2 10, columns are added to record the EXPLAIN, SQLADM, system DBADM, DATAACCESS, and ACCESSCTRL system authorities. Table 10-4 provides details about these catalog columns.

Table 10-4 Catalog table changes for SYSIBM.SYSUSERAUTH

Column name	Data type	Description
ACCESSCTRLAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the grantee has system ACCESSCTRL authority: ► blank: Privilege is not held ► Y: Privilege is held without the GRANT option
CREATESECUREAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the GRANTEE can create secured objects (triggers and user-defined functions): ► blank: Privilege is not held ► Y: Privilege is held without the GRANT option
DATAACCESSAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the grantee has system DATAACCESS authority: ► blank: Privilege is not held ► Y: Privilege is held without the GRANT option
EXPLAINAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the grantee has EXPLAIN privilege: ► blank: Privilege is not held ► G: Privilege is held with the GRANT option ► Y: Privilege is held without the GRANT option
SDBADMAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the grantee has system DBADM authority: ► blank: Privilege is not held ► Y: Privilege is held without the GRANT option
SQLADMAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the grantee has SQLADM authority: ► blank: Privilege is not held ► G: Privilege is held with the GRANT option ► Y: Privilege is held without the GRANT option

Additional information: For a full description of the catalog tables referenced at 10.2.4, "Catalog changes" on page 359, refer to *DB2 10 for z/OS SQL Reference,* SC19-2983.

10.2.5 SECADM authority

DB2 10 introduces a security administrative authority, the SECADM authority, that allows you to manage security-related DB2 objects and to control access to all database objects and resources. SECADM has no implicit privilege to access user data or to manage database objects. However, if a table privilege is granted to PUBLIC, everyone gets access, including SECADM user.

The SECADM authority is available in DB2 10 CM. In case of fall back to previous versions, the DSNZPARMs are reassembled, and SECADM is removed.

Privileges held by the SECADM authority

The SECADM authority implicitly holds the grantable privileges shown in Table 10-5.

Table 10-5 Privileges held by SECADM authority

Privilege category	Description
SELECT	Select on all catalog tables
INSERT, UPDATE, DELETE	All updatable catalog tables including SYSIBM.SYSAUDITPOLICY
GRANT, REVOKE (SQL DCL)	► Grant access to and revoke access from all database resources ► Revoke explicit privileges that are granted by SECADM ► Revoke explicit privileges granted by others using the BY clause
Create, alter, drop, comment	► Column mask controls ► Row permission controls
Alter user table to	► Activate and deactivate column level access crontrol ► Activate and deactivate row level acces control
Create, drop, comment	Role
Create, drop, comment, alter	Trusted context
Online DSNZPARM changes	SECADM can perform any online DSNZPARM change
Utilities	► Unload catalog tables only ► DSN1SDMP
Commands	Start, alter, stop, and display trace
CREATE_SECURE_OBJECT	Alter the SECURED or NOT SECURED clause on triggers and user-defined functions

Activating the SECADM authority

In DB2 10 NFM, the SECADM authority is activated either implicitly (if you do not provide the SECADM related DSNZPARMs) or explicitly (if you do provide the SECADM related DSNZPARMs).

DSNZPARMs for the SECADM authority

DB2 10 introduces the DSNZPARMs shown in Table 10-6 to configure and manage the SECADM authority.

Table 10-6 DSNZPARMs for SECADM

DSNZPARM	Description
SECADM1	Authorization ID or role name of first SECADM AUTHID: 1 to 8 characters, starting with an alphabetic character ROLE: Ordinary SQL identifier up to 128 bytes and must not use any of the reserved words ACCESSCTRL, DATAACCESS, DBADM, DBCTRL, DBMAINT, NONE, NULL, PACKADM, PUBLIC, SECADM, or SQLADM. Default if not provided in DSNZPARM: SECADM
SECADM1_TYPE	Specifies whether SECADM1 is an authorization ID or a role Acceptable values: AUTHID, ROLE Default: AUTHID

DSNZPARM	Description
SECADM2	Authorization ID or role name of second SECADM AUTHID: 1 to 8 characters, starting with an alphabetic character ROLE: Ordinary SQL identifier up to 128 bytes and must not use any of the reserved words ACCESSCTRL, DATAACCESS, DBADM, DBCTRL, DBMAINT, NONE, NULL, PACKADM, PUBLIC, SECADM, or SQLADM. Default if not provided in DSNZPARM: SECADM
SECADM2_TYPE	Specifies whether SECADM2 is an authorization ID or a role Acceptable values: AUTHID, ROLE Default: AUTHID
SEPARATE_SECURITY	Security administrator duties (SECADM) to be separated from system administrator duties (SYSADM) Acceptable values: YES or NO Default value: NO

The DSNZPARMs described in Table 10-6 are online changeable by either install SYSADM or the SECADM authority. Any attempt by an authorization ID or role to change any of these online DSNZPARMs fails.

Notes regarding authorities:

- INSTALL SYSADM always has the right to perform security-related tasks.
- If SEPARATE_SECURITY is set to NO, the SYSADM authority implicitly holds the SECADM authority.
- If SEPARATE_SECURITY is set to YES, the SYSADM authority does not hold the privilege to perform security related tasks.
- SECADM1, SYSADM, and install SYSADM can all grant and manage security objects at the same time.

Defaults for SECADM related DSNZPARMs

If you do not specify any of the SECADM DSNZPARMs listed in Table 10-6, DB2 uses their default values. With these defaults, the SECADM authority is not separated from the SYSADM authority, and the SYSADM authority implicitly has SECADM authority and continues to function as in DB2 10 CM or in previous versions of DB2.

At this point (with SEPARATE_SECURITY set to NO), you must implement the SECADM authority authorization ID that is provided by the DSNZPARM defaults, because your SYSADM authority implicitly has SECADM authority and can exercise any privilege that is implicitly held by the SECADM authority.

Setting DSNZPARM SEPARATE_SECURITY to NO

If you set DSNZPARM SEPARATE_SECURITY to NO, the SYSADM authority implicitly has SECADM authority, in which case the SYSADM authority behaves exactly the same way as in DB2 10 CM and in previous versions of DB2 for z/OS.

Setting DSNZPARM SEPARATE_SECURITY to NO is recommended for DB2 for z/OS version migrations. With this DSNZPARM setting, you preserve the status of your existing SYSADM authorities.

Setting DSNZPARM SEPARATE_SECURITY to YES

After you set DSNZPARM SEPARATE_SECURITY to YES, the SECADM authority can perform the tasks listed in Table 10-5 on page 360. SYSADM does not have the privileges to

grant and revoke any of these privileges in DB2. Install SYSADM is not affected by the SEPARATE_SECURITY setting. It can grant and revoke all the time. Only granted SYSADM is affected.

Specify your own SECADM DSNZPARM

You can activate a SECADM authority of your choice by setting at least one of the SECADM1 or SECADM2 DSNZPARMs to a ROLE or an authorization ID of your organization. Table 10-7 shows the SECADM related DSNZPARMs that we set explicitly in our DB2 10 environment.

Table 10-7 SECADM authority DSNZPARMs settings of our sample scenario

DSNZPARM	Parameter value
SECADM1	DB0BSECA
SECADM1_TYPE	AUTHID
SECADM2	DB0BSECA
SECADM2_TYPE	AUTHID
SEPARATE_SECURITY	NO

We set both DSNZPARMs, SECADM1 and SECADM2, to the same value, to RACF group DB0BSECA. In the command output provided in Figure 10-17, user DB2R53 is connected to that RACF group. As a result, user DB2R53 has the capability to perform DB2 work that requires the privileges of the SECADM authority. For example, DB2R53 can perform DSNZPARM online changes of SECADM related DSNZPARMs and can perform any of the tasks listed in Table 10-5 on page 360.

```
LG DB0BSECA
INFORMATION FOR GROUP DB0BSECA
    SUPERIOR GROUP=SYS1         OWNER=RC63        CREATED=10.215
    INSTALLATION DATA=SECADM RACF GROUP FOR DB2 SYSTEM DB0B
    NO MODEL DATA SET
    TERMUACC
    NO SUBGROUPS
    USER(S)=      ACCESS=      ACCESS COUNT=      UNIVERSAL ACCESS=
      DB2R53        USE           000000               NONE
         CONNECT ATTRIBUTES=NONE
         REVOKE DATE=NONE                 RESUME DATE=NONE
```

Figure 10-17 user DB2R53 is connected to RACF group DB0BSECA

For the SECADM authority to be exercised by user DB2R53, user DB2R53 must set its CURRENT SQLID special register to DB0BSECA. Otherwise, user DB2R53 cannot grant and revoke privileges, and any attempt to grant or revoke privileges under an SQLID other than DB0BSECA fails with SQLCODE -551.

Revoking the SECADM authority

Similar to install SYSADM, the SECADM authority is not recorded in the DB2 catalog and cannot be granted or revoked. Instead, SECADM can be modified only through an online DSNZPARM change that must be performed either by the install SYSADM or the SECADM authority. Removing the SECADM authority from an authorization ID or role through an online DSNZPARM change does not revoke any dependent privileges that are granted by SECADM.

Revoking privileges granted by others

If you use the SECADM authority to revoke privileges granted by others, you must specify the BY clause in the revoke statement as shown in Example 10-11.

Example 10-11 SECAM to revoke privileges granted by others

```
REVOKE DATAACCESS ON SYSTEM FROM DBOBDA BY DBOBSECA
NOT INCLUDING DEPENDENT PRIVILEGES ;
```

RACF/DB2 Access Control Authorization Exit users

You can define the SECADM authority in RACF for use with the RACF/DB2 Access Control Authorization Exit. For a list of RACF profiles that are available to manage the DB2 10 security authorities through RACF, refer to 10.2.11, “Using RACF profiles to manage DB2 10 authorities” on page 377.

Auditing the SECADM authority

You can use a category SECMAINT audit policy to keep track of grants and revokes performed by SECADM authorities. In our DB2 10 environment, we used the SQL statement shown in Example 10-12 to create an audit policy to audit all grants and revokes performed by SECADM authorities.

Example 10-12 SECMAINT audit policy - grant - revoke auditing

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
   (AUDITPOLICYNAME, SECMAINT) VALUES('AUDSECMAINT','A');
```

We then started the audit policy and used the SECADM authority authorization ID DB0BSECA to grant the DATAACCESS privilege to authorization ID DB0BDA (see Example 10-13).

Example 10-13 SECMAINT audit policy - grant DATAACESS SQL

```
SET CURRENT SQLID = 'DBOBSECA';
GRANT  DATAACCESS ON SYSTEM TO   DBOBDA;
```

As a result, an IFCID 141 audit trace record with an audit reason of SECADM was created by DB2. The OMEGAMON PE formatted audit trace for this trace record is shown in Figure 10-18.

```
TIMESTAMP   TYPE                                  DETAIL
----------- -------- ------------------------------------------------------------------
18:32:11.91 AUTHCNTL GRANTOR: DBOBSECA   OWNER TYPE: PRIM/SECOND AUTHID
                                         REASON: SECADM                 SQLCODE:     0
                     OBJECT TYPE: USERAUTH
                     TEXT: GRANT DATAACCESS ON SYSTEM TO DBOBDA
```

Figure 10-18 OMEGAMON PE category SECMAINT audit report

10.2.6 System DBADM authority

DB2 10 introduces the system DBADM database administrative authority, which allows you to perform administrative tasks throughout all databases of a DB2 system and to issue commands for a DB2 subsystem without being able to access user data and without the ability to grant and revoke privileges.

The system DBADM is available in DB2 10 NFM.

Privileges held by system DBADM

The system DBADM authority implicitly holds the grantable privileges shown in Table 10-8.

Table 10-8 System DBADM privileges

Privilege category	Description
System privileges	BINDADD BINDAGENT CREATEALIAS CREATEDBA CREATEDBC CREATEMTAB DISPLAY MONITOR1 MONITOR2 TRACE EXPLAIN SQLADM
On all collections	CREATEIN
All user databases	CREATETAB CREATETS DISPLAYDB DROP IMAGCOPY RECOVERDB STARTDB STOPDB
On all user tables except for tables with row permissions or column masks defined	ALTER INDEX REFERENCES TRIGGER TRUNCATE
On all packages	BIND, COPY
On all plans	BIND
On all schemes	CREATEIN ALTERIN DROPIN
On all sequences	ALTER
On all distinct types	ALTER
Use privileges	TABLESPACES
On all catalog tables	SELECT, UNLOAD
On all updatable catalog tables except for table SYSIBM.SYSAUDITPOLICY	INSERT UPDATE DELETE

Archived

Privilege category	Description
Utilities	► Unload on catalog tables only ► DSN1SDMP to start, stop and capture monitor traces
User-defined routines	Execute privilege on all system-defined routines (functions and procedures)

Management of trusted contexts, roles, row permissions, column masks, and altering or defining triggers and UDFs with the SECURED or NOT SECURED clause is entirely in the responsibility of the SECADM authority. SECADM exclusively manages to protect business sensitive information, which is why System DBADM cannot manage any of these objects.

The system DBADM authority does not have any of the ARCHIVE, BSDS, CREATESG, and STOSPACE system privileges. Therefore, you need to grant these privileges separately if an authorization ID with system DBADM needs to perform tasks that require these privileges.

The system DBADM authority can set the plan or package bind owner to any value, provided that DSNZPARM SEPARATE_SECURITY is set to NO. However, the package owner specified by system DBADM must have authorization to execute all statements that are embedded in the package.

Granting and revoking the system DBADM privilege

The system DBADM authority can be granted or revoked only by an authorization ID or role with SECADM authority. Revoking this authority does not revoke dependent privileges.

Under SECADM authority, we used the SQL statements shown in Example 10-14 to grant and revoke the system DBADM privilege for the authorization ID DB0BSDBA. By default, the DATAACCESS and ACCESSCTRL authorities are granted when the system DBADM authority is granted.

Example 10-14 Grant or revoke system DBADM privilege

```
SET CURRENT SQLID = 'DBOBSECA';
GRANT  DBADM ON SYSTEM TO    DBOBSDBA;
REVOKE DATAACCESS ON SYSTEM FROM DBOBSDBA BY DBOBSECA
NOT INCLUDING DEPENDENT PRIVILEGES ;
```

If you do not want the system DBADM authority to have access to data and the ability to grant and revoke privileges, you have to explicitly exclude the DATAACCESS and ACCESSCTRL privileges from the grant system DBADM SQL statement as shown in Example 10-15.

Example 10-15 Grant system DBADM WITHOUT DATAACCESS WITHOUT ACCESSCTRL

```
GRANT DBADM WITHOUT DATAACCESS WITHOUT ACCESSCTRL ON SYSTEM TO DBOBSDBA;
```

Note that WITHOUT DATAACESS is not applicable in a persistent way to all objects. An ID WITHOUT DATAACCESS can still be granted EXPLICIT privileges.

When you revoke the system DBADM privilege, you must specify the NOT INCLUDING DEPENDENT PRIVILEGES clause. Otherwise, the revoke statement fails with SQLCODE -104. If system DBADM is granted WITH DATAACCESS and WITH ACCESSCTRL, then DATAACCESS and ACCESSCTRL must be revoked explicitly.

Creating objects

The system DBADM authority can create databases, tables, aliases, sequences, table spaces, and distinct types without requiring any additional privileges. System DBADM can also create objects such as tables, views, indexes, and aliases for someone else. However, to create functions, indexes, triggers, MQTs, procedures, or views, the system DBADM authority might require additional privileges as described in Table 10-9.

Table 10-9 Additional privileges required by system DBADM

DB2 object	Description
Function using a table as input parameter	If the function uses a table as parameter. one of the following privileges: ► Select privilege on any table that is an imput parameter to the function ► DATAACCESS authority
Sourced function	If the function is sourced on another function: ► Execute privilege for the source function ► DATAACCESS authority
External function or procedure required to run in a WLM APPLENV	If the function or procedure runs in a WLM application environment (WLM APPLENV) ► Authority to create functions or procedures in that WLM APPLENV
JAVA function or procedure	If a .jar file name is specified in the external name clause, one of the following privileges: ► Usage privilege on the .jar file ► DATAACCESS authority
Index on expression	If the index is created on an expression, one of the following privileges: ► Execute privilege on any UDF that is involved in the index expression ► DATAACCESS authority
Trigger	If the trigger accesses other DB2 objects, one of the following privileges: ► The privilege that satisfies the authorization requirement for the SQL statement contained in the trigger ► DATAACCESS authority
View	One of the following privileges: ► Select privileges on the table or view referenced by the view ► DATAACCESS authority

Altering and dropping objects

The system DBADM authority can alter and drop user objects without requiring object ownership or the privilege to alter or drop the object. However, the system DBADM cannot alter or drop any of the security objects that are exclusively managed by the SECADM authority. These security objects include:

- Alter user tables to activate or deactivate row level access control
- Alter triggers to modify the SECURED or NOT SECURED clause
- Alter functions to modify SECURED or NOT SECURED clause

RACF/DB2 Access Control Authorization Exit users

You can define the system DBADM authority in RACF for use with the RACF/DB2 Access Control Authorization Exit. For a list of RACF profiles that are available to manage the DB2 10 security authorities through RACF, refer to 10.2.11, “Using RACF profiles to manage DB2 10 authorities” on page 377.

Auditing the system DBADM authority

You can use a category DBADMIN audit policy to keep track of activities that are performed by system DBADM authorities. In our DB2 10 environment, we used the SQL statement shown in Example 10-16 to create an audit policy to audit all database administrative tasks performed by system DBADM authorities. We set the DBADMIN column to “*” to audit all authorities.

Example 10-16 DBADMIN audit policy - system DBADM auditing

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
   (AUDITPOLICYNAME, DBADMIN) VALUES('AUDDBADMIN','*');
```

We then started the audit policy and used the system DBADM authority authorization ID DB0BSDBA to create a plan table in schema DB0BEXPLAIN (see Example 10-17).

Example 10-17 DBADMIN audit policy - create table by system DBADM

```
SET CURRENT SQLID = 'DB0BSDBA';
CREATE TABLE DB0BEXPLAIN.PLAN_TABLE LIKE DB2R5.PLAN_TABLE;
```

As a result, DB2 created an IFCID 361 audit trace record for authorization type SYSDBADM. Figure 10-19 shows the OMEGAMON PE formatted audit trace for this trace record.

```
AUTHCNTL AUTH TYPE:          SYSDBADM
         PRIV CHECKED:       CREATE TABLE                    OBJECT TYPE: DATABASE
         AUTHID:             DB0BSDBA
         SOURCE OBJECT
          QUALIFIER:         SYSIBM
          NAME:              DSNDB04
         TARGET OBJECT
          QUALIFIER:         DB0BEXPLAIN
          NAME:              PLAN_TABLE
         TEXT:               CREATE TABLE DB0BEXPLAIN.PLAN_TABLE LIKE DB2R5.PLAN_TABLE
```

Figure 10-19 OMEGAMON PE auth type SYSDBADM audit report

Additional information: For more information about row level and column mask control, refer to 10.5, “Support for row and column access control” on page 384.

10.2.7 DATAACCESS authority

DB2 10 introduces the DATAACCESS authority, which allows you to access data in all user tables, views, and materialized query tables within a DB2 subsystem. A DATAACCESS authority furthermore can execute all plans, packages, functions, and procedures.

The DATAACCESS authority is available in DB2 10 NFM.

Privileges held by DATAACCESS

The DATAACCESS authority implicitly has the grantable privileges shown in Table 10-10.

Table 10-10 DATAACCESS implicitly held grantable privileges

Privilege category	Description
System privilege	DEBUGSESSION
All user tables, views, MQTs	SELECT INSERT UPDATE DELETE TRUNCATE
All plans, packages, routines	EXECUTE
All databases	RECOVERDB REORG REPAIR LOAD
All jar files	USAGE
All sequences	USAGE
All distinct types	USAGE
All catalog tables	SELECT
All updatable catalog tables except for SYSIBM.SYSAUDITPOLICIES	INSERT UPDATE DELETE

Granting and revoking the DATAACCESS authority

DATAACCESS authority can be granted or revoked only by an authorization ID or role with SECADM authority. Revoking this authority does not revoke dependent privileges. The WITH GRANT option is not supported with DATAACCESS and is ignored when used in grant statements. Under SECADM authority, we used the SQL statements shown in Example 10-18 to grant and revoke the DATAACCESS authority for the authorization ID DB0BDA.

Example 10-18 Grant or revoke the DATAACCESS privilege

```
SET CURRENT SQLID = 'DB0BSECA';
GRANT  DATAACCESS ON SYSTEM TO   DB0BDA;
REVOKE DATAACCESS ON SYSTEM FROM DB0BDA BY DB0BSECA
NOT INCLUDING DEPENDENT PRIVILEGES ;
```

When you revoke the DATAACCESS privilege, you must specify the NOT INCLUDING DEPENDENT PRIVILEGES clause. Otherwise, the revoke statement fails with SQLCODE -104.

RACF/DB2 Access Control Authorization Exit users

You can define the DATAACCESS authority in RACF for use with the RACF/DB2 Access Control Authorization Exit. For a list of RACF profiles that are available for managing the DB2 10 security authorities through RACF, refer to 10.2.11, “Using RACF profiles to manage DB2 10 authorities” on page 377.

Auditing the DATAACCESS authority

You can use a category DBADMIN audit policy to keep track of activities that are performed by DATAACCESS authorities. In our DB2 10 environment, we used the SQL statement shown in Example 10-19 to create an audit policy to audit all database administrative tasks performed by DATAACCESS authorities. We set the DBADMIN column to T to audit only DATAACCESS authorities.

Example 10-19 DBADMIN audit policy: DATAACCESS auditing

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
   (AUDITPOLICYNAME, DBADMIN) VALUES('AUDDBADMIN_T','T');
```

We then started the audit policy and used the DATAACCESS authority authorization ID DB0BDA to query plan table DB2R5.PLAN_TABLE (Example 10-20).

Example 10-20 DBADMIN audit policy - query a table by DATAACCESS authority

```
SET CURRENT SQLID = 'DB0BDA';
SELECT * FROM DB2R5.PLAN_TABLE ;
```

As a result, DB2 created an IFCID 361 audit trace record for authorization type DATAACCESS. Figure 10-20 shows the OMEGAMON PE formatted audit trace for this trace record.

```
 TYPE                                DETAIL
-------- ------------------------------------------------------------------------------
AUTHCNTL AUTH TYPE:          DATAACCS
         PRIV CHECKED:       SELECT                          OBJECT TYPE:       TAB/VIEW
         AUTHID:             DB0BDA
         SOURCE OBJECT
          QUALIFIER:         DB2R5
          NAME:              PLAN_TABLE
         TARGET OBJECT
          QUALIFIER:         N/P
          NAME:              N/P
         TEXT:               SELECT * FROM DB2R5.PLAN_TABLE
```

Figure 10-20 OMEGAMON PE auth type DATAACCESS audit report

10.2.8 ACCESSCTRL authority

DB2 10 introduces the ACCESSCTRL authority, which allows you to grant and revoke explicit privileges to AUTHIDs and ROLEs. The ACCESSCTRL authority itself is not holding any SQL DML privilege that allow access to user tables.

The ACCESSCTRL authority is available in DB2 10 NFM.

Archived

Privileges held by ACCESSCTRL

ACCESSCTRL holds the grantable privileges shown in Table 10-11.

Table 10-11 ACCESSCTRL implicitly held grantable privileges

Privilege category	Description
All catalog tables	SELECT UNLOAD
All catalog tables	INSERT, UPDATE, DELETE on all updatable catalog tables, except for SYSIBM.SYSAUDITPOLICIES
Grant, revoke	► Grant all grantable privileges except for system DBADM, DATAACCESS, ACCESSCTRL and CREATE_SECURE_OBJECT ► Revoke privileges granted by others using the BY clause, except for system DBADM, DATAACCESS, ACCESSCTRL and CREATE_SECURE_OBJECT

Granting and revoking the ACCESSCTRL authority

The ACCESSCTRL authority can be granted or revoked only by the SECADM authority. Revoking ACCESSCTRL does not revoke privileges and authorities granted by ACCESSCTRL. The WITH GRANT option is not supported with ACCESSCTRL and is ignored when used in grant statements. In our DB2 10 environment, we used the SQL statements shown in Example 10-21 to grant and revoke the ACCESSCTRL privilege for the authorization ID DB0BAC.

Example 10-21 Grant and revoke the ACCESSCTRL privilege

```
GRANT  ACCESSCTRL ON SYSTEM   TO DB0BAC;
REVOKE ACCESSCTRL ON SYSTEM FROM DB0BAC BY DB0BSECA
       NOT INCLUDING DEPENDENT PRIVILEGES;
```

When you revoke the ACCESSCTRL privilege, you must specify the NOT INCLUDING DEPENDENT PRIVILEGES clause. Otherwise, the revoke statement fails with SQLCODE -104.

Revoking privileges granted by others

If you use the ACCESSCTRL authority to revoke privileges granted by others you must specify the BY clause in the revoke statement as shown in Example 10-11.

Example 10-22 SECADM to revoke privileges granted by others

```
REVOKE DATAACCESS ON SYSTEM FROM DB0BDA BY DB0BSECA
NOT INCLUDING DEPENDENT PRIVILEGES ;
```

If you do not specify the BY clause, the revoke statement fails with SQLCODE -556.

RACF/DB2 Access Control Authorization Exit users

You can define the ACCESSCTRL authority in RACF for use with the RACF/DB2 Access Control Authorization Exit. For a list of RACF profiles that are available to manage the DB2 10 security authorities through RACF, refer to 10.2.11, “Using RACF profiles to manage DB2 10 authorities” on page 377.

Auditing the ACCESSCTRL authority

You can use a category DBADMIN audit policy to keep track of activities that are performed by ACCESSCTRL authorities. In our DB2 10 environment, we used the SQL statement shown in Example 10-23 to create an audit policy to audit all database administrative tasks performed by ACCESSCTRL authorities. We set the DBADMIN column to G to audit only ACCESSCTRL authorities.

Example 10-23 DBADMIN audit policy: ACCESSCTRL auditing

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
   (AUDITPOLICYNAME, DBADMIN) VALUES('AUDDBADMIN_G','G');
```

We then started the audit policy and used the ACCESSCTRL authority authorization ID DB0BAC to grant the select privilege on table DB2R5.PLAN_TABLE to public (refer to Example 10-24).

Example 10-24 DBADMIN audit policy: Grant privilege by ACCESSCTRL authority

```
SET CURRENT SQLID = 'DB0BAC';
GRANT SELECT ON TABLE DB2R5.PLAN_TABLE TO PUBLIC;
```

As a result, DB2 created an IFCID 361 audit trace record for authorization type ACCESSCTRL.

Figure 10-21 shows the OMEGAMON PE formatted audit trace for this trace record.

```
 TYPE                              DETAIL
-------- -------------------------------------------------------------------------------
AUTHCNTL AUTH TYPE:          ACCSCTRL
         PRIV CHECKED:       SELECT                          OBJECT TYPE:       TAB/VIEW
         AUTHID:             DB0BAC
         SOURCE OBJECT
          QUALIFIER:         DB2R5
          NAME:              PLAN_TABLE
         TARGET OBJECT
          QUALIFIER:         DB0BAC
          NAME:              N/P
         TEXT:               GRANT SELECT ON TABLE DB2R5.PLAN_TABLE TO PUBLIC
```

Figure 10-21 OMEGAMON PE auth type ACCESSCTRL audit report

10.2.9 Authorities for SQL tuning

DB2 10 introduces the EXPLAIN and SQLADM authorities. These authorities have no access to user data. They have only the privileges that are required for SQL query tuning activities, such as explain, prepare, and describe table.

Using these authorities, you can enable application developers and SQL tuning specialists to perform SQL tuning even in production environments, without exposing user data. This function is supported by your favorite SQL tuning tool. The privileges that are provided by the SQLADM authority are a super set of the privileges that are provided by the EXPLAIN privilege (see Figure 10-22).

Archived

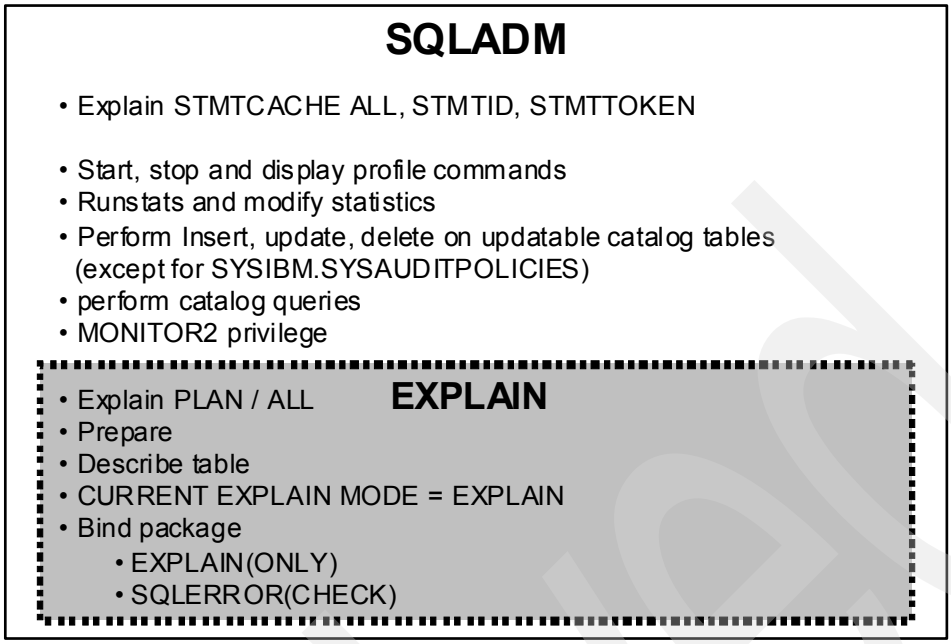


Figure 10-22 SQLADM authority and EXPLAIN privilege

EXPLAIN privilege

Using the EXPLAIN privilege, you can perform SQL explain, prepare, and describe table statements without having the privilege to execute the explainable SQL statement that is to be explained, prepared, or described.

Availability

The EXPLAIN privilege is available in DB2 10 NFM.

Privileges held by the EXPLAIN privilege

With the EXPLAIN privilege, you can perform the tasks described in Table 10-12 without having to require access to user data.

Table 10-12 Privileges of EXPLAIN privilege

Privilege	Description
Explain	SQL explain with option PLAN and ALL
Prepare	Prepare statement without binding an executable statement
Describe table	Obtaining table structure information
Bind	Bind packages with options ► EXPLAIN ONLY explains SQL statements contained in DBRM. cannot be used with rebind. ► SQLERROR(CHECK) performs all syntax and semantics checks on the SQL statements contained in a DBRM
CURRENT EXPLAIN MODE=EXPLAIN	Explain dynamic SQL statements executing under special register, CURRENT EXPLAIN MODE = EXPLAIN

Granting and revoking the EXPLAIN privilege

The EXPLAIN privilege can be granted or revoked by the SECADM authority or ACCESSCTRL authority. EXPLAIN can be revoked by the EXPLAIN privilege, if the EXPLAIN privilege was granted by an authorization ID or role with EXPLAIN privilege and GRANT option. In our DB2 10 environment, we used the SQL statements shown in Example 10-25 to grant and revoke the EXPLAIN privilege for user DB2R52. In this revoke example, DB2 does not perform a cascading revoke of EXPLAIN privileges granted by authorization ID DB2R52.

Example 10-25 Grant, revoke the explain privilege

```
GRANT  EXPLAIN ON SYSTEM TO   DB2R52 WITH GRANT OPTION;
REVOKE EXPLAIN ON SYSTEM FROM DB2R52 BY DBOBSECA
       NOT INCLUDING DEPENDENT PRIVILEGES;
```

Other privileges required by the EXPLAIN privilege

To execute the explain SQL statement the authorization ID or role that you assigned the EXPLAIN privilege, you also need plan or package execution authorization, which is required to execute EXPLAIN privilege for dynamic SQL. For example, you might want to grant that authorization ID the execute privilege on plan DSNTEP2 for mainframe-based dynamic SQL access or on package collection NULLID for DRDA-based dynamic SQL access.

The authorization ID or role that you assign EXPLAIN privilege also needs all privileges on the set of EXPLAIN tables for that authorization ID or role. For example, if you grant only the EXPLAIN privilege to authorization ID DB2R52 and subsequently create the EXPLAIN tables in schema DB2R52, authorization ID DB2R52 does not become the EXPLAIN table owner. In that case, you must grant all privileges on all EXPLAIN tables in schema DB2R52 explicitly to authorization ID DB2R52 so that DB2R52 can further process EXPLAIN tables.

CURRENT EXPLAIN MODE not to be used with EXPLAIN privilege

The EXPLAIN privilege has only the privileges described in Table 10-12. Any attempt to execute an SQL DML statement other than explain, prepare, or describe table fails. For example, if you as an EXPLAIN privilege use the CURRENT EXPLAIN MODE special register to run an SQL query in EXPLAIN mode, your query fails with SQLCODE -522, because the EXPLAIN privilege does not hold the EXPLAINED MONITORED privilege (see SQL error message in Figure 10-23 for details).

```
---------+---------+---------+---------+---------+---------+---------+--------
 set current explain mode = EXPLAIN;
---------+---------+---------+---------+---------+---------+---------+--------
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+--------
 select * from dsn_statemnt_table;
---------+---------+---------+---------+---------+---------+---------+--------
DSNT408I SQLCODE = -552, ERROR:  DB2R51 DOES NOT HAVE THE PRIVILEGE TO PERFORM
         OPERATION EXPLAIN MONITORED STATEMENTS
---------+---------+---------+---------+---------+---------+---------+--------
```

Figure 10-23 EXPLAIN privilege failure with CURRENT EXPLAIN MODE special register

If you have only the privileges that are implicitly held by the EXPLAIN privilege, you need to use the SQL EXPLAIN statement to successfully explain the SQL query (see Example 10-26).

Example 10-26 EXPLAIN

```
EXPLAIN ALL SET QUERYNO=578
  FOR
   SELECT * FROM DSN_STATEMNT_TABLE;
```

New bind options EXPLAIN(ONLY) and SQLERROR(CHECK)

DB2 10 introduces the EXPLAIN(ONLY) and SQLERROR(CHECK) package bind options. These two bind package options can be used only in package BIND commands. They are unavailable to the BIND PLAN and REBIND package commands.

The EXPLAIN privilege holds the implicit privilege to bind packages with the EXPLAIN(ONLY) and the SQLERROR(CHECK) bind options. The EXPLAIN(ONLY) bind package option performs an SQL EXPLAIN for each SQL statement that is contained in the DBRM. The SQLERROR(CHECK) bind package option performs a syntax check on SQL statements that are contained in the DBRM.

RACF/DB2 Access Control Authorization Exit users

You can define the EXPLAIN privilege in RACF for use with the RACF/DB2 Access Control Authorization Exit. For a list of RACF profiles that are available to manage the DB2 10 security authorities through RACF, refer to 10.2.11, “Using RACF profiles to manage DB2 10 authorities” on page 377.

Auditing the EXPLAIN privilege

The audit category DBADMIN does not provide auditing for the EXPLAIN privilege, because auditing is performed on authorities, not privileges. The EXPLAIN privilege does not allow access to sensitive information.

IFCID 361 is cut for all successful accesses by a user if the trace is started using the IFCID keyword.

SQLADM authority

The SQLADM implicitly holds the privileges of the EXPLAIN privilege. Additionally, the SQLADM authority allows you to issue start, stop, and display profile commands, to execute RUNSTATS, and to modify statistics utilities. It also includes the MONITOR2 privileges, which allow SQLADM to start, read, and stop the instrumentation facility interface (IFI) monitor traces.

Availability

The SQLADM authority is available in DB2 10 NFM.

Privileges held by the SQLADM authority

With the SQLADM authority, you can perform the tasks that are described in Figure 10-22 on page 372 without requiring access to user data.

Granting and revoking the SQLADM authority

An authorization ID or role with SECADM or ACCESSCTRL authority can grant or revoke the SQLADM privilege. In our DB2 10 environment, we used the SQL statements shown in Example 10-27 to grant and revoke the SQLADM privilege for user DB2R51. In this example, DB2 performs a cascading revoke of SQLADM privileges granted by authorization ID DB2R51.

Example 10-27 Grant, revoke SQLADM privilege

```
GRANT  SQLADM ON SYSTEM TO   DB2R51 WITH GRANT OPTION;
REVOKE SQLADM ON SYSTEM FROM DB2R51 BY DBOBSECA
    INCLUDING DEPENDENT PRIVILEGES;
```

CURRENT EXPLAIN MODE special register

With the SQLADM system privilege, you can set the CURRENT EXPLAIN MODE special register to EXPLAIN to dynamically explain a workload of SQL statements. As shown in Figure 10-24, you can dynamically explain a series of SQL statements using the CURRENT EXPLAIN MODE special register.

```
set current explain mode = EXPLAIN
DB20000I  The SQL command completed successfully.
select 1 from sysibm.sysdummy1
SQL0217W  The statement was not executed as only Explain information requests
are being processed.  SQLSTATE=01604
set current explain mode = NO
DB20000I  The SQL command completed successfully.
SELECT PROGNAME,CREATOR,TNAME,ACCESSTYPE,STMTTOKEN FROM PLAN_TABLE
DB20000I  The SQL command completed successfully.
PROGNAME CREATOR  TNAME              ACCESSTYPE STMTTOKEN
-------- -------- ------------------ ---------- ----------
SQLC2H20 SYSIBM   SYSDUMMY1          R          (NULL)
SQLC2H20 SYSIBM   SYSDUMMY1          R          (NULL)
```

Figure 10-24 SQLADM authority to use CURRENT EXPLAIN MODE special register

As indicated by SQL0217W, the query requested only explain information. As a result, no executable statement was prepared and stored in the dynamic statement cache. For this reason, the plan table has no statement token information for the SQL queries shown in Figure 10-24.

RACF/DB2 Access Control Authorization Exit users

You can define the SQLADM authority in RACF for use with the RACF/DB2 Access Control Authorization Exit. For a list of RACF profiles that are available to manage the DB2 10 security authorities through RACF, refer to 10.2.11, “Using RACF profiles to manage DB2 10 authorities” on page 377.

Archived

Auditing the SQLADM authority

You can use the category DBADMIN audit policy to keep track of activities performed by SQLADM authorities. In our DB2 10 environment, we used the SQL statement shown in Example 10-28 to create an audit policy to audit all database administrative tasks performed by SQLADM authorities. We set the DBADMIN column to K to audit only SQLADM authorities.

Example 10-28 DBADMIN audit policy: SQLADM auditing

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
   (AUDITPOLICYNAME, DBADMIN) VALUES('AUDDBADMIN_K','K')
```

We then started the audit policy and used the SQLADM authority authorization ID DB2R51 to query table SYSIBM.SYSVOLUMES (refer to Example 10-29).

Example 10-29 DBADMIN audit policy: Query table by SQLADM authority

```
SELECT * FROM SYSIBM.SYSVOLUMES
```

As a result, DB2 created an IFCID 361 audit trace record for authorization type SQLADM. Figure 10-25 shows the OMEGAMON PE formatted audit trace for this trace record.

```
 TYPE                                DETAIL
-------- -------------------------------------------------------------------------------
AUTHCNTL AUTH TYPE:          SQLADM
         PRIV CHECKED:       SELECT                          OBJECT TYPE:       TAB/VIEW
         AUTHID:             DB2R51
         SOURCE OBJECT
          QUALIFIER:         SYSIBM
          NAME:              SYSVOLUMES
         TARGET OBJECT
          QUALIFIER:         N/P
          NAME:              N/P
         TEXT:               SELECT * FROM SYSIBM.SYSVOLUMES
```

Figure 10-25 OMEGAMON PE auth type SQLADM audit report

10.2.10 The CREATE_SECURE_OBJECT system privilege

DB2 10 introduces the CREATE_SECURE_OBJECT system privilege, which is required to secure triggers or scalar functions that are used on row permission or column mask enforced tables. For details about row and column mask enforced tables, refer to 10.5, “Support for row and column access control” on page 384.

If you use triggers or functions on tables that are row permission or column mask enforced, your trigger or function needs to be marked as secure.

Archived

10.2.11 Using RACF profiles to manage DB2 10 authorities

If you use the DB2 RACF Access Control Module to enforce security policy, you can use the RACF profiles listed in Table 10-13 to manage the DB2 10 administrative authorities and privileges.

Table 10-13 DB2 RACF profiles for DB2 10 new authorities

Authority name	RACF profile	RACF class
SECADM	DB2-ssid.SECADM	DSNADM
ACCESSCTRL	DB2-ssid.ACCESSCTRL	DSNADM
System DBADM	DB2-ssid.SYSDBADM	DSNADM
DATAACCESS	DB2-ssid.DATAACCESS	DSNADM
SQLADM	DB2-ssid.SQLADM	MDSNSM or GDSNSM
EXPLAIN	DB2-ssid.EXPLAIN	MDSNSM or GDSNSM

Additional information: For more details about DB2 RACF profiles, refer to *DB2 10 for z/OS RACF Access Control Module Guide,* SC19-2982.

10.2.12 Separating SECADM authority from SYSADM and SYSCTRL authority

DB2 10 introduces DSNZPARM SEPARATE_SECURITY, which you can use to separate the security authority from system and database administration authority. You can set this DSNZPARM to YES or NO. The default value is NO.

DSNZPARM SEPARATE_SECURITY is updatable using the online DSNZPARM capability. The online DSNZPARM change can be performed only by users with installation SYSADM or SECADM authority.

Security administration authorities not separated from SYSADM

If you specify NO (the default) for DSNZPARM SEPARATE_SECURITY, the install SYSADM and SYSADM authorities implicitly have the SECADM authority and can continue to manage all security objects and to grant and revoke privileges granted by others. Additionally, the SYSADM authority manages security objects, including the row-level and column-level access control security objects available with DB2 10.

Users with SYSCTRL authority implicitly have ACCESSCTRL authority and can continue to perform security-related tasks with no authority to access user data. Additionally, SYSCTRL can manage roles and can set the BIND OWNER to any value, provided that the specified owner qualifies for the data access privilege that is required by the SQL DML statements contained in the package.

Archive

Figure 10-26 illustrates the privileges implicitly held by the SYSADM authorities when DB2 is configured with DSNZPARM SEPARATE_SECURITY set to NO. In this scenario, SYSADM can execute all privileges, including those of the SECADM and ACCESSCTRL authorities.

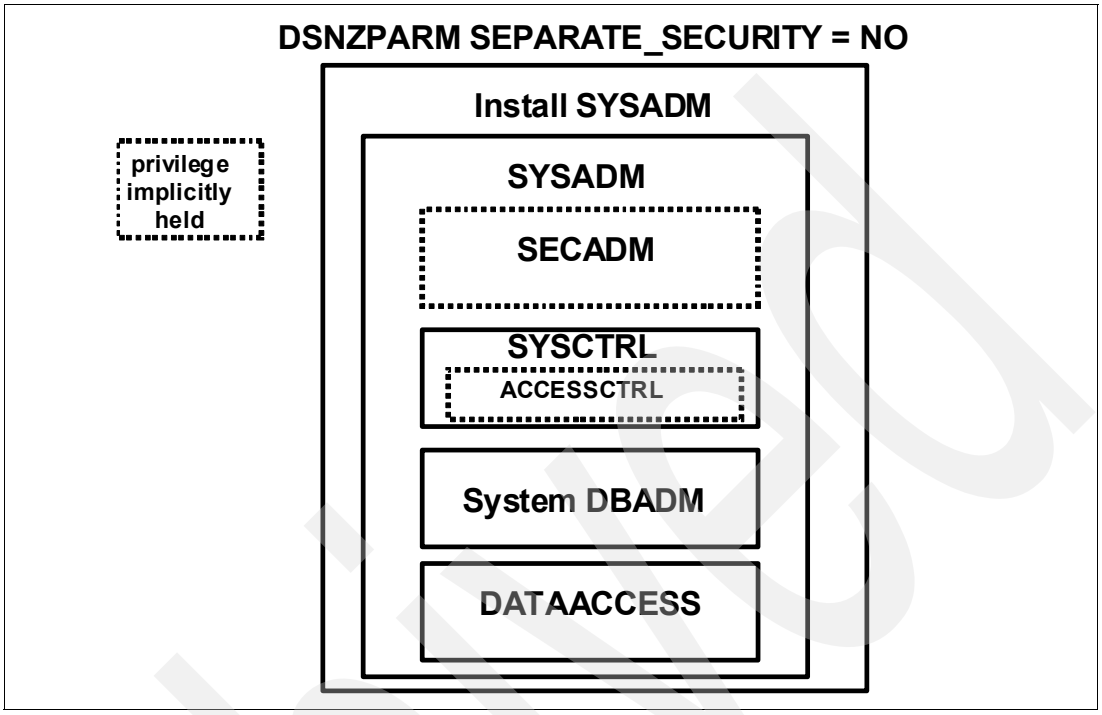


Figure 10-26 SECADM and ACCESSCTRL not separated from SYSADM

Security administration authorities separated from SYSADM

Install SYSADM is not affected by the SEPARATE_SECURITY setting. Install SYSADM can always grant, revoke, and manage security objects.

After you set the SEPARATE_SECURITY system parameter to YES, the SYSADM authority cannot manage security objects or grant and revoke privileges. The SYSCTRL authority does not have the ability to manage roles either. From this point, the SECADM authorization grants and revokes privileges and manages these objects instead. SYSADM or SYSCTRL cannot perform grants to revoke privileges granted by others. Existing privileges granted by SYSADM and SYSCTRL are left intact. You need a user with ACCESSCTRL or SECADM authority to revoke any of these privileges.

Archive

Figure 10-27 illustrates the separation of security administrative duties from the SYSADM authorities when DB2 is configured with DSNZPARM SEPARATE_SECURITY set to YES. In this scenario, the security administration authority SECADM is separated from SYSADM. The SYSADM authority (including install SYSADM) loses the power to execute the privileges held by the SECADM and ACCESSCTRL authorities.

Important: The SYSADM authority can continue to grant and revoke privileges on tables for which it is the table owner. If you do not desire that behavior, you can use the DB2 CATMAINT utility to change the table owner to a different authorization ID or role name.

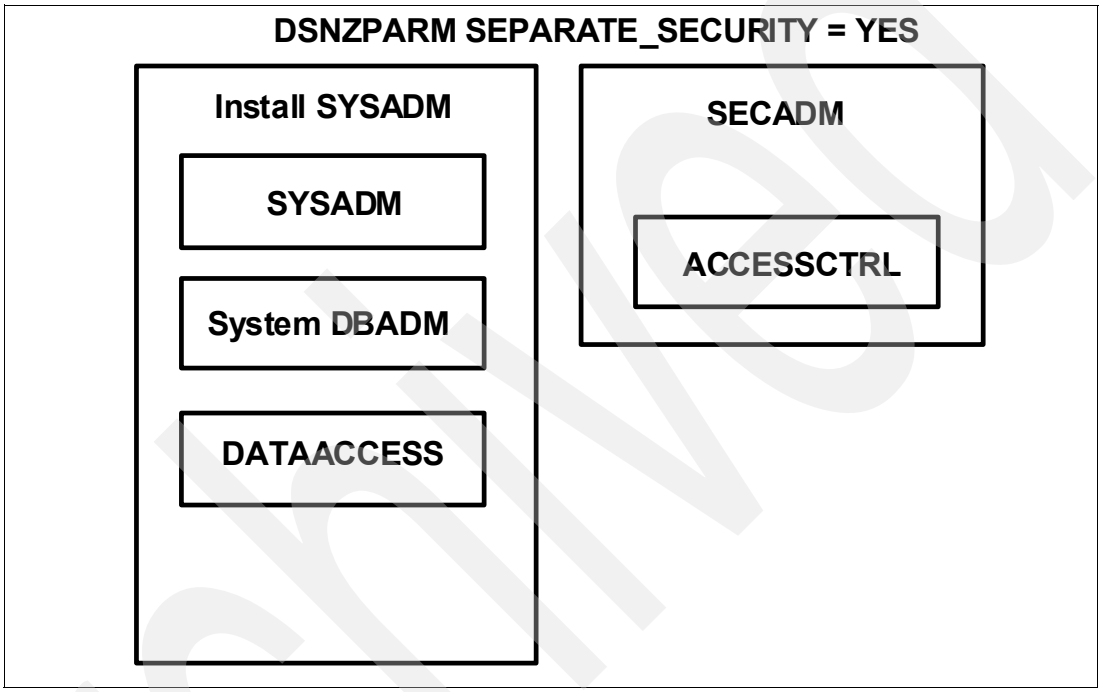


Figure 10-27 SECADM and ACCESSCTRL separated from SYSADM

Important: With system parameter SEPARATE_SECURITY set to YES, the following conditions are true:

- The SYSADM authority cannot set the CURRENT SQLID special register to an ID that is not included in the list of secondary authorization IDs (RACF groups).
- The SYSADM, SYSCTRL and system DBADM authorities cannot set BIND OWNER to an ID that is not included in the list of secondary authorization IDs.

If you plan to separate the SECADM authority from the SYSADM authority by setting DSNZPARM SEPARATE_SECURITY to YES, you must consider these runtime behavior changes. For example, if your application runs under the SYSADM authority and is used to set the CURRENT SQLID special register to an arbitrary value, your application might not work correctly. To avoid similar situations, we strongly recommend to use the DB2 10 policy-based audit facility to monitor and analyze carefully the use of SYSADM authorities in your environment prior to setting SEPARATE_SECURITY DSNZPARM to YES.

Archive

10.2.13 Minimize need for SYSADM authorities

An authorization ID or role with SYSADM authority can access data from any table in the entire DB2 subsystem. To minimize the risk of exposing sensitive data to too many people, you can plan to reduce the number of authorization IDs or roles that have SYSADM authority. In DB2 10, the SYSADM authority is required only for certain system administrative tasks (provided the other administrative authorities are appropriately in use) and only during a short window of time.

To limit the number of users with SYSADM authority, you can separate the privileges of the SYSADM authority and migrate them to other administrative authorities, which allows you to minimize the need for granting the SYSADM authority. For example, you can use this function to perform the following tasks:

- The INSTALL SYSADM authority grants SYSADM authority to a RACF group or role.
  - Users eligible to have SYSADM authority are connected to the SYSADM RACF group just for the time the SYSADM authority is needed to perform the task that requires SYSADM authority.
  - The SYSADM trusted context is enabled just for the time the SYSADM authority is needed to perform the task that requires SYSADM authority
- DSNZPARM SECADM1 and SECADM2 assign SECADM authority to security administrators who perform security administration and manage access control.
- The SECADM authority grants ACCESSCTRL authority to database administrators who control access to DB2.
- The SECADM authority grants system DBADM authority to database administrators who only manage objects.
- The SECADM authority grants DATAACCESS authority to database administrators who only access data.
- The SECADM or ACCESSCTRL authority grant SQLADM authority to performance analysts who are responsible for analyzing DB2 performance.
- The SQLADM authority grants the EXPLAIN privilege to application architects and developers who need to explain SQL statements or collect metadata information about the SQL statements.
- The SECADM or ACCESSCTRL authority grant the SYSOPR authority, the ARCHIVE, BSDS, CREATESG, and STOSPACE privileges to system administrators for performing system administrative tasks.
- The SECADM authority uses the policy-based auditing to monitor the usage of the administrative authorities to monitor the usage of administrative authorities.

You can furthermore protect data from SQL access through SYSADM authorities by implementing row access control on tables that contain business sensitive information. For details about row access controls, refer to 10.5, “Support for row and column access control” on page 384.

10.3 System-defined routines

In DB2 10, any function or procedure that is created or altered by the install SYSADM authority is marked as *system defined* (the SYSIBM.SYSROUTINES.SYSTEM_DEFINED catalog column is set to S). Furthermore, the DBADM and SQLADM authorities system holds the implicit execute privilege on any routine that is marked as system-defined and on any

package executed within such system-defined routines. In this section, we discuss the installation of system-defined routines and the addition of user-defined routines.

We also discuss DB2-supplied routines in 9.9, “DB2-supplied stored procedures” on page 329 and 12.13, “Simplified installation and configuration of DB2-supplied routines” on page 523.

10.3.1 Installing DB2-supplied system-defined routines

When you install and configure the DB2 for z/OS supplied routines, you run install job DSNTRIN under install SYSADM authority, which marks any routine installed by DSNTRIN as system defined. As a result, these routines can be executed by the system DBADM and SQLADM authorities, which fits well with popular SQL tuning tools. For example, IBM Optim™ Query Tuner requires many of the DB2-supplied stored procedures to be available and accessible by an authority that focuses on SQL tuning activities.

The following DB2-supplied system-defined routines are commonly used:

- Stored procedures for administration
- JDBC metadata stored procedures that are used by the JDBC driver when accessing the DB2 catalog
- WLM_REFRESH and WLM_SET_CLIENT_INFO
- Stored procedures for XLM schema administration
- Real Time Statistics stored procedure DSNACCOX
- SOAP user-defined functions
- MQ user-defined functions
- The DB2 utility DSNUTILS, DSNUTILU stored procedures
- DSNWZP stored procedure for retrieving DB2 DSNZPARM and DSNHDECP parameter settings

Additional information: For a complete list of DB2-supplied stored procedures, refer to *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974. For details about DB2-stored procedures for administration, refer to *DB2 10 for z/OS Administration Guide,* SC19-2968.

10.3.2 Define your own system-defined routines

If you want to mark one of your own routines as system-defined to make it implicitly executable for authorization IDs and roles with system DBADM and SQLADM authority, you need an install SYSADM authority to create or to alter that routine. For example, your install SYSADM authorization ID can perform a dummy alter on that routine (for example, alter one of the routine parameters to its current setting).

10.3.3 Mark user-provided SQL table function as system defined

In our DB2 10 environment, we granted user DB2R52 to have only SQLADM authority. The SQLADM authority had the implicit execute privilege on system-defined routines. When DB2R52 tried to execute the GET_SECURITY_ENFORCED_TABLES table UDF to check whether any row or column access controls are activated on the customer table, a SQLCODE

217 is issued as shown in Figure 10-28. The table UDF at this point is not marked as system defined.

```
---------+---------+---------+---------+---------+---------+---------+---------
SELECT * FROM TABLE(DB2R5.GET_SECURITY_ENFORCED_TABLES(
'DB2R5','CUSTOMER'
)) AS A;
---------+---------+---------+---------+---------+---------+---------+---------
TBCREATOR   TBNAME            TYPE    NAME
---------+---------+---------+---------+---------+---------+---------+---------
DSNT404I SQLCODE = 217, WARNING:  THE STATEMENT WAS NOT EXECUTED AS ONLY
         EXPLAIN INFORMATION REQUESTS ARE BEING PROCESSED
---------+---------+---------+---------+---------+---------+---------+---------
```

Figure 10-28 SQLADM Execution failure non-system-defined routine

Under install SYSADM, we performed a dummy alter function statement to mark the table UDF as system defined, as shown in Example 10-30, which set the corresponding SYSIBM.SYSROUTINES.SYSTEM_DEFINED catalog column to S.

Example 10-30 SQLADM marking UDF as system defined through dummy alter

```
SET CURRENT SQLID = 'DBOBINST';
ALTER FUNCTION DB2R5.GET_SECURITY_ENFORCED_TABLES
 (XTBCREATOR VARCHAR(128),XTBNAME    VARCHAR(128) ) RESTRICT
 READS SQL DATA;
```

Now that the table UDF is marked as system defined, user DB2R52 can execute the GET_SECURITY_ENFORCED_TABLES table UDF successfully (see Figure 10-29).

```
---------+---------+---------+---------+---------+---------+---------+---------
 SELECT * FROM TABLE(DB2R5.GET_SECURITY_ENFORCED_TABLES(
 'DB2R5','CUSTOMER'
 )) AS A;
---------+---------+---------+---------+---------+---------+---------+---------
TBCREATOR   TBNAME            TYPE    NAME
---------+---------+---------+---------+---------+---------+---------+---------
DB2R5       CUSTOMER          COLUMN  INCOME_BRANCH
DB2R5       CUSTOMER          ROW     SYS_DEFAULT_ROW_PERMISSION__CUSTOMER
DB2R5       CUSTOMER          ROW     RA01_CUSTOMERS
```

Figure 10-29 SQLADM run system-defined user provided UDF

Important: Do not use the install SYSADM authority to create or alter routines that should not be marked as system defined. If you in your day-to-day operations use the install SYSADM authority to create or alter routines that access business sensitive data, you unintentionally expose these routines to be executed by the SQLADM and system DBADM authorities.

10.4 The REVOKE dependent privilege clause

If you revoke privileges in DB2 9, cascading revokes dependent privileges. In some situations, this behavior is not desirable, especially in situations in which you want to keep privileges that are granted by an authority that you plan to revoke. For example, you might need to revoke a SYSADM authority, and you do not want to lose privileges that are granted by that authority.

DB2 10 addresses this issue by adding a dependent privilege clause to the revoke statement that allows you to control whether dependent privileges go through cascade revoke processing. The depending privileges clause is available in DB2 10 NFM.

10.4.1 Revoke statement syntax

Figure 10-30 shows the DB2 10 SQL revoke syntax diagram.

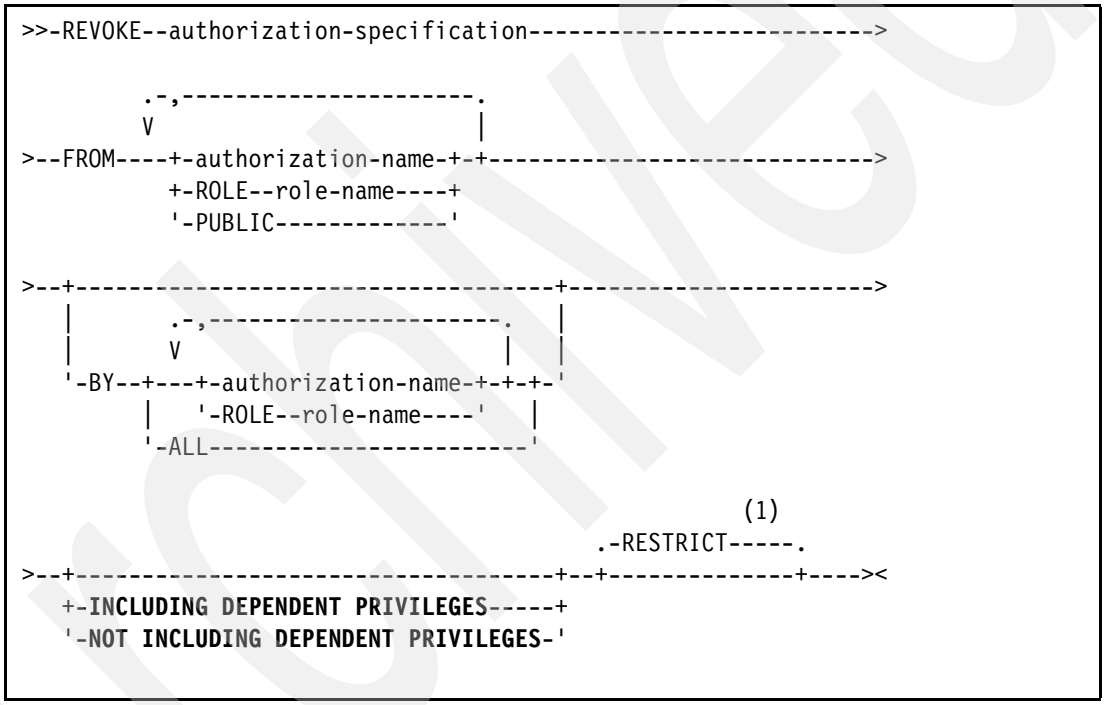


Figure 10-30 Revoke syntax diagram

DB2 10 supports the new dependent privileges clause for all SQL revoke statements.

10.4.2 Revoke dependent privileges system default

The dependent privileges clause on the SQL REVOKE statement is controlled by the DSNZPARM REVOKE_DEP_PRIV. The system default for the revoke dependent privileges clause depends on the authority being revoked and the setting of DSNZPARM REVOKE_DEP_PRIVILEGES. Possible system defaults for the REVOKE DEPENDING PRIVILEGES clause are listed in Table 10-14.

Archived

Table 10-14 Default behavior for REVOKE_DEPENDING_PRIVILEGES

Privilege to be revoked ACCESSCTRL,DATAACCESS or system DBADM	DSNZPARM setting for REVOKE_DEP_PRIVILEGES	REVOKE DEPENDING PRIVILEGES default
YES	NO/YES/SQLSTMT	NO
NO	NO	NO
NO	YES	YES
NO	SQLSTMT	YES

When you revoke any of the ACCESSCTRL, DATAACCESS, or system DBADM administrative authorities, you must specify the NOT INCLUDING DEPENDENT PRIVILEGES clause. Otherwise, the revoke statement fails with SQLCODE -104. This mechanism protects you from unintentional cascading revokes, which can potentially harm the availability of the application system.

10.5 Support for row and column access control

DB2 10 introduces a method of implementing *row and column access control* as an additional layer of security that you can use to complement the privileges model and to enable DB2 to take part in your efforts to comply with government regulations for security and privacy. You use row and column access control to control SQL access to your tables at the row level, column level, or both, based on your security policy.

After you enforce row- or column-level access control for a table, any SQL DML statement that attempts to access that table is subject to the row and column access control rules that you define. During table access, DB2 transparently applies these rules to every user, including the table owner and the install SYSADM, SYSADM, SECADM, system DBADM, and DBADM authorities, which can help to close remaining security loopholes. For example, you can use the new row and column access controls to transparently hide table rows from users with SYSADM authority or to mask confidential table column information.

DB2 enforces row and column access control at table level, transparent to any kind of SQL DML operation or applications that access that table through SQL DML. Existing views do not need to be aware of row or column access controls because the access rules are enforced transparently on its underlying tables and table columns. When a table with row or column level control is accessed through SQL, all users of a table are affected, regardless of how they access the table (through an application, through ad-hoc query tools, through report generation tools, or other access) or what authority or privileges they hold.

Row and column access control is available in DB2 10 NFM.

10.5.1 Authorization

Only the SECADM authority has the privilege to manage row and column access controls, row permissions and column masks. That privilege cannot be granted to others. The management tasks to be performed exclusively by SECADM include:

- Alter a table to activate or deactivate row access control
- Alter a table to activate or deactivate column access control
- Create, alter, and drop row permissions objects
- Create, alter, and drop column mask objects

The SECADM authority does not need to have object or execute privileges on tables, views, or routines that are referenced by row permissions and column mask objects.

10.5.2 New terminology

Before we discuss the new DB2 10 access control features in more detail, we clarify the new terminology for these new controls and objects here:

- *Access control* refers to the method that DB2 10 introduces to implement access control at row and at column level.
- *Row access control* is the DB2 security mechanism that uses SQL to control access to a table at row level. When you activate row access control by altering the table DB2 defines a default row permission with a predicate of 1=0 to prevent any access to that table (because the predicate 1=0 will never be true).
- *Column access control* is the DB2 security mechanism that uses SQL to control access to a table at column level. You activate column access control by altering the table.
- A *row permission* is a table based database object that describes the row filtering rule DB2 implicitly applies to all table rows for every user whenever the table is accessed using SQL DML. You create a row permission by the new CREATE PERMISSION DDL. As shown Figure 10-31, DB2 enforces a row permission only if the row permission is explicitly enabled and row access control for the table is activated.

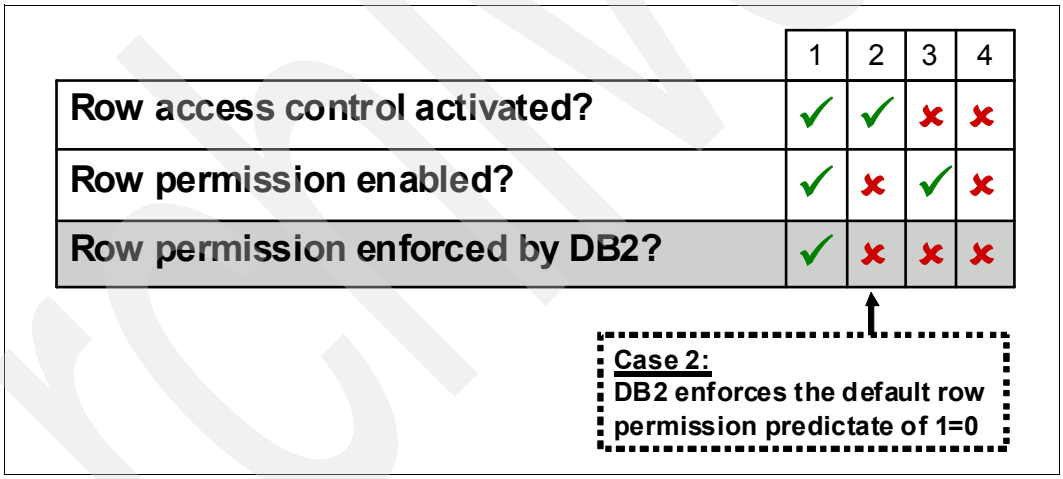


Figure 10-31 Row permission enforcement

Archive

- A *column mask* is a table based database object that describes the column masking rule DB2 implicitly applies to a table column for every user whenever the table column is referenced using SQL DML. You create a column mask by the new CREATE MASK DDL. As shown in Figure 10-32, DB2 enforces a column mask only if the column mask is explicitly enabled and column access control for the table is activated.

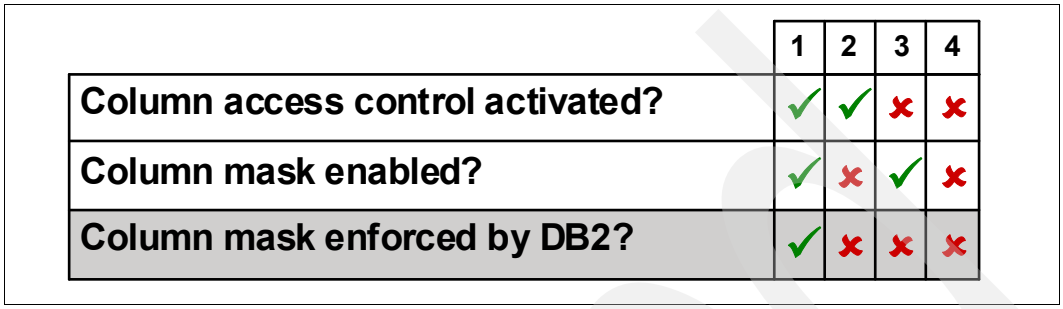


Figure 10-32 Column mask enforcement

10.5.3 Object types for row and column based policy definition

DB2 row access and column access control is similar to a table-based switch that you can set to activate or deactivate the enforcement of row and column security rules for a table. Except for the default row access control predicate (1=0), activating row and column access control itself does not define any of the row and column security policies that you want to enforce for individual users, groups or roles.

In DB2 10, you can use the following object types to define table row and table column policy rules:

- Row permission object
- Column mask object

These object types are managed through SQL DCL operations that are performed exclusively by the SECADM authority. A row permission object is a row filter that DB2 implicitly applies when the table is accessed through SQL DML. A column mask is similar to a scalar function that returns a masked value for a column. If enforced for the column, the column mask is implicitly applied by DB2 when that column is referenced in an SQL DML operation.

You can create and enable these objects any time, even if row access and column access controls for the table are not enabled. However, DB2 enforces these object types only if you explicitly enable them and if you alter the table to activate row access control (prerequisite to enforce row permissions) or column access control (prerequisite to enforce column masks) for the table. The order in which you activate controls or enable these object types is irrelevant. DB2 enforces row permission and column mask objects as soon as you enable these objects and their corresponding access controls. For more information about when DB2 enforces row permissions and column masks, refer to the illustrations provided in Figure 10-31 and in Figure 10-32.

10.5.4 SQL DDL for managing new access controls and objects

DB2 10 implements the following SQL DDL changes to support the creation and management of the new access controls and objects:

- Row access control

  In DB2 10, the alter table DDL statement is extended to support the activation and deactivation of row access control, as shown in Figure 10-33. DB2 provides no support to activate or deactivate row access control in the create table statement.

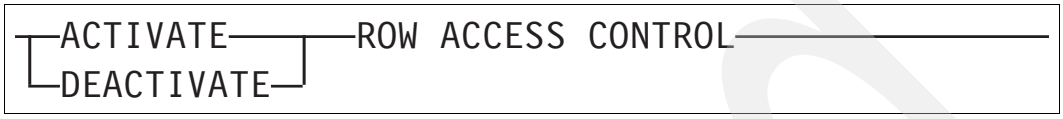


Figure 10-33 Alter table row access control

- Column access control

  In DB2 10, the alter table DDL statement is extended to support the activation and deactivation of column access control, as shown in Figure 10-34. DB2 provides no support to activate or deactivate column access control in the create table statement.

Figure 10-34 Alter table column access control

- Row permission

  DB2 10 provides an SQL DDL interface to create and alter row permission objects. Figure 10-35 shows the SQL DDL statement to create row permission objects.

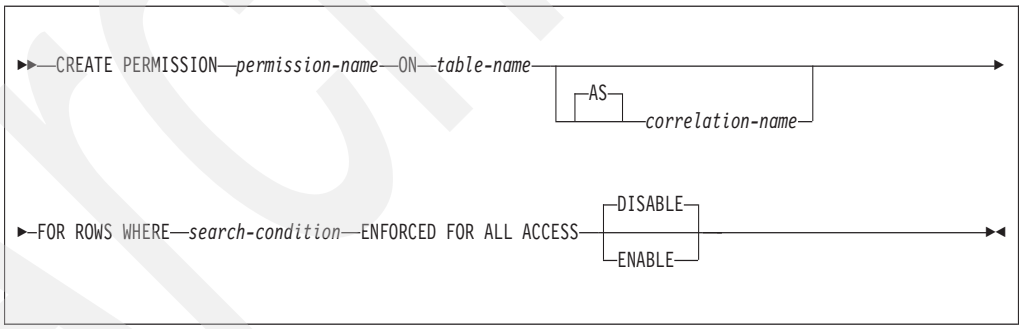


Figure 10-35 CREATE PERMISSION SQL DDL

Figure 10-36 shows the SQL DDL statement to alter a row permission.

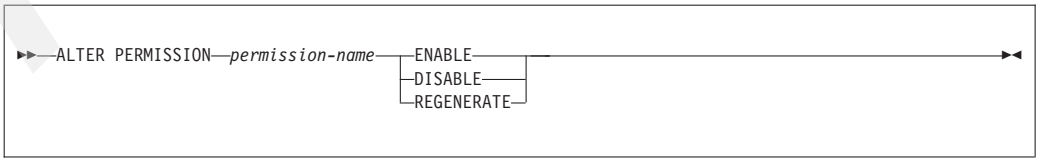


Figure 10-36 ALTER PERMISSION DDL

Archive

- Column mask

  DB2 10 provides an SQL DDL interface to create and alter column mask objects. Figure 10-37 shows the SQL DDL to create a mask object.

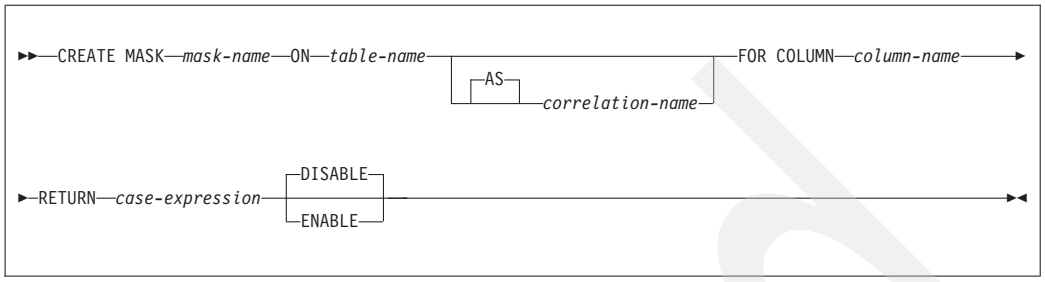


Figure 10-37 CREATE MASK SQL DDL

Figure 10-38 shows the SQL DDL statement to alter a mask.

Figure 10-38 ALTER MASK DDL

You can also DROP a column mask.

Similar CREATE, ALTER, and DROP statements apply to row permissions.

10.5.5 Built-in functions

DB2 provides the following scalar built-in functions to support row and column access control:

- VERIFY_GROUP_FOR_USER
- VERIFY_TRUSTED_CONTEXT_ROLE_FOR_USER
- VERIFY_ROLE_FOR_USER

You can use these built-in functions to verify from a row permission or a column mask whether the SQL user is connected to a RACF group, whether the user runs within a given trusted context, and whether the user runs under a given role. These built-in functions can help you to control row permissions and column masks based on RACF groups, trusted contexts and roles.

Although these built-in functions are introduced to support row and column access control, you can use them outside of row permission and column mask objects as well.

For more information about these new built-in functions, refer to *DB2 10 for z/OS SQL Reference,* SC19-2983.

Archived

10.5.6 Catalog tables and row and column access control

DB2 10 introduces a catalog table and implements changes to existing catalog tables to support row and column access control. The new SYSIBM.SYSCONTROLS table stores row permissions and column masks. In addition, the following existing catalog tables are changed:

- The SYSIBM.SYSDEPENDENCIES table reflects column mask dependencies
- The SYSIBM.SYSOBJROLEDEP table reflects row and column mask dependencies
- The SYSIBM.SYSTABLES table reflects row and column access control
- The SYSIBM.SYSTRIGGERS table supports the new secure attribute
- The SYSIBM.SYSCOLUMNS table reflects column mask existence
- The SYSIBM.SYSROUTINES table supports the new secure attribute
- The SYSIBM.SYSUSERAUTH table reflects new system authority SECURE_AUTH

For more detailed information about the DB2 catalog table changes for access control, refer to 10.5.16, “Catalog changes for access control” on page 406.

When you alter a table to activate row or column access control, create a row permission or column mask, or alter an existing row permission or column mask, DB2 reflects changes that are caused by any of these activities in its catalog, as illustrated in Figure 10-39.

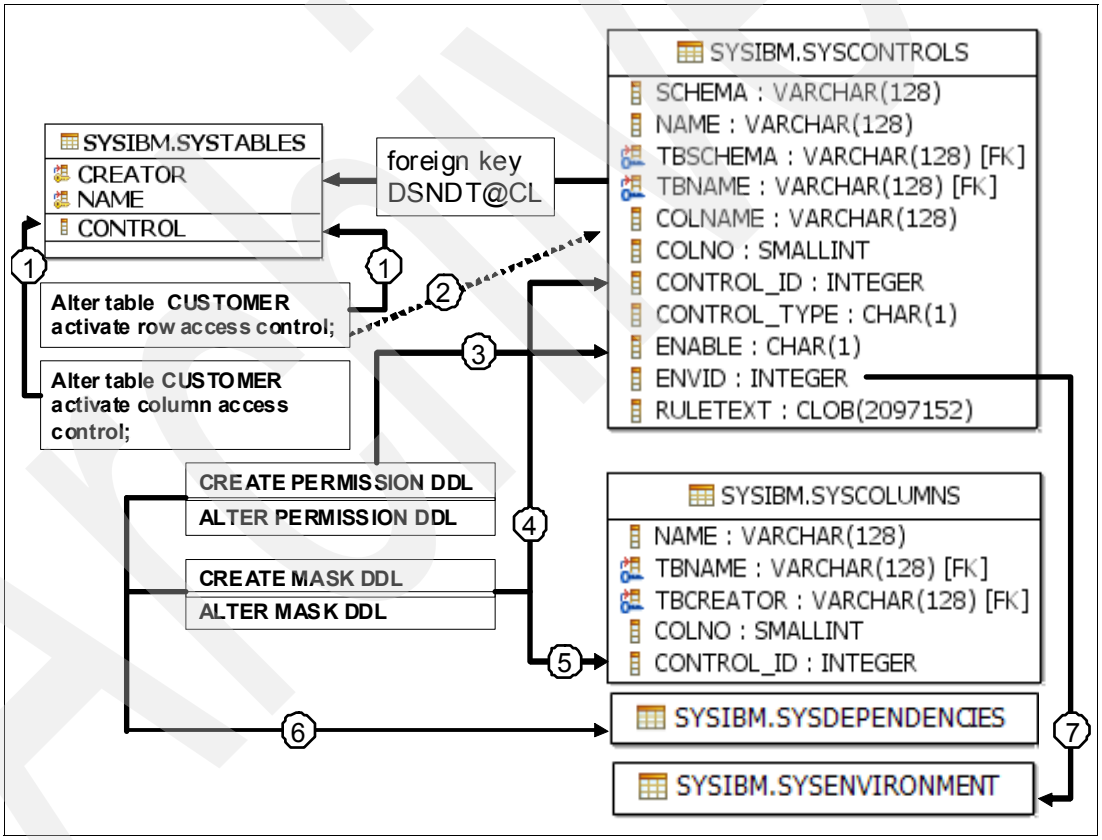


Figure 10-39 Catalog tables and access control

The scenario illustrated in Figure 10-39 shows the catalog changes that DB2 performs when you issue any of the SQL DCL statements that are provided for row and column access control. The steps in this process are as follows:

1. An authorization ID with SECADM authority alters the customer table to activate row and column access control. The ALTER commands cause DB2 to update the SYSIBM.SYSTABLES.CONTROL catalog columns to reflect the activation of row and column access control in SYSIBM.SYSTABLES.
2. Because of the row access control activation, DB2 inserts a default row permission with the default predicate of 1=0 into the SYSIBM.SYSCONTROLS catalog table to restrict all access to the table. The default row permission is deleted from the SYSIBM.SYSCONTROLS table when you deactivate row access control.
3. An authorization ID with SECADM authority creates a row permission, which causes DB2 to insert a row that contains the row policy information into the SYSIBM.SYSCONTROLS catalog table. The SYSIBM.SYSCONTROLS.ENABLE column indicates whether the row permission is enabled or disabled. The SECADM authorization ID then issues an alter permission statement to enable the row permission, which causes DB2 to subsequently update the ENABLE column in SYSIBM.SYSCONTROLS.
4. An authorization ID with SECADM authority creates a column mask, which causes DB2 to insert a row that contains the column masking rule into the SYSIBM.SYSCONTROLS catalog table. The SYSIBM.SYSCONTROLS.ENABLE column indicates whether the column mask is enabled or disabled. The SECADM authorization ID then issues an alter mask statement to enable the column mask, which causes DB2 to subsequently update the ENABLE column in SYSIBM.SYSCONTROLS.
5. Because of the create mask DDL statement, DB2 assigns a unique CONTROL_ID for the row mask definition and stores that CONTROL_ID information in the SYSIBM.SYSCONTROLS and SYSIBM.SYSCOLUMNS catalog tables to indicate the existence of a column mask in the SYSIBM.SYSCOLUMNS table.
6. DB2 reflects any column dependency caused by a column mask or row permission in the SYSIBM.SYSDEPENDENCIES catalog table.
7. The SYSIBM.SYSCONTROLS.ENVID column value uniquely identifies the environment variable settings that DB2 used when it created the row permission or column mask policy row in SYSIBM.SYSCONTROLS. You can use the ENVID value to query the SYSIBM.SYSENVIRONMENT catalog table to determine the environment variable settings that are used for the row permission or column mask.

10.5.7 Sample customer table used in examples

The examples that we provide in this chapter to illustrate row and column access control use the customer table data shown in Figure 10-40.

```
SELECT * FROM DB2R5.CUSTOMER
---------+---------+---------+---------+---------+---------+---------+---------
ACCOUNT              NAME        PHONE           INCOME  BRANCH
---------+---------+---------+---------+---------+---------+---------+---------
1111-2222-3333-4444  Alice       111-1111         22000  A
2222-3333-4444-5555  Bob         222-2222         71000  B
3333-4444-5555-6666  Louis       333-3333        123000  B
4444-5555-6666-7777  David       444-4444        172000  C
```

Figure 10-40 Sample data customer table

10.5.8 Row access control

After you activate row access control on a table, all table rows are skipped during SQL processing, unless one or more row permissions exist that allow SQL access, for example by a user, group, or role.

In our customer table scenario, we queried the customer table when no row access control was enforced on the table. Therefore, the SQL query illustrated in Figure 10-40 returned all table rows. We then executed the alter table SQL statement shown in Example 10-31 to enforce row access control.

Example 10-31 Activate row access control on customer table

```
SET CURRENT SQLID = 'DBOBSECA';  --> SECADM racf group
ALTER TABLE DB2R5.CUSTOMER
     ACTIVATE ROW    ACCESS CONTROL ;
```

The ACTIVATE ROW ACCESS CONTROL clause imposes a default row permission predicate of the form 1=0 to all references to data in the customer table. With one single clause, you restrict all access to the data in the customer table because the implied search condition can never be true.

Upon successful row access control activation we queried the SYSIBM.SYSCONTROLS catalog table to list existing row permissions for the customer table. The activation of row access control for the customer table caused DB2 to insert a default row policy for the table. As shown in Figure 10-41, DB2 defined that row permission with a default name of SYS_DEFAULT_ROW_PERMISSION__CUSTOMER. DB2 enables a default row permission for all references to the table, including SELECT, INSERT, UPDATE, and DELETE SQL DML operations.

```
SELECT
   SUBSTR(NAME,01,36)       AS "Row Permission NAME"
 , SUBSTR(TBSCHEMA,01,08)   AS TBSCHEMA
 , SUBSTR(TBNAME,01,08)     AS TBNAME
 , SUBSTR(RULETEXT,01,08)   AS RULETEXT
  , ENFORCED, IMPLICIT, ENABLE
FROM SYSIBM.SYSCONTROLS
WHERE CONTROL_TYPE = 'R' --> indicates row permission
 AND  OWNER = 'SYSIBM'   --> indicates default row permission
 AND  IMPLICIT = 'Y'     --> implicitly created for row access control
 AND (TBSCHEMA,TBNAME) = ('DB2R5','CUSTOMER')
 with ur;
---------+---------+---------+---------+---------+---------+---------+---------+---------+----
Row Permission NAME                  TBSCHEMA  TBNAME    RULETEXT  ENFORCED  IMPLICIT  ENABLE
---------+---------+---------+---------+---------+---------+---------+---------+---------+----
SYS_DEFAULT_ROW_PERMISSION__CUSTOMER  DB2R5     CUSTOMER  1=0       A         Y         Y
```

Figure 10-41 Default row permission stored in SYSIBM.SYSCONTROLS

At this point, only row access control is activated on the customer table. As yet, no column mask exists. Only the DB2 generated default row permission is enforced on the customer table. When you query the customer table, DB2 implicitly applies the default row permission with a predicate of 1=0, which causes DB2 to return no rows as illustrated in Figure 10-42.

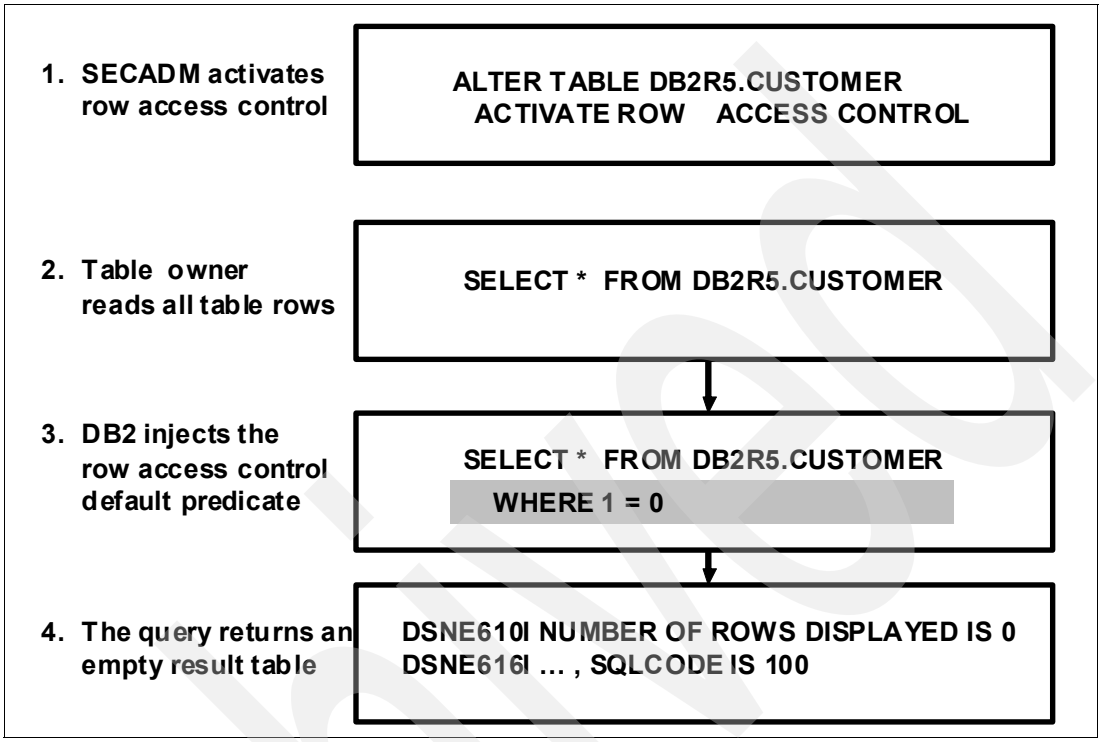


Figure 10-42 Query row access control with default predicate 1=0

With just the default row permission activated, DB2 returns no rows for any read access, regardless of the SQL statement or the authority of the statement authorization ID, except for the owner who can still read the data. We now run the query shown in Figure 10-40 on page 390 again. As shown in Figure 10-43, no rows are returned by DB2 although the query was run under the install SYSADM authority, which is the highest authority in DB2 for z/OS.

```
---------+---------+---------+---------+---------+---------+---------+---------
 SELECT * FROM DB2R5.CUSTOMER WITH UR;
---------+---------+---------+---------+---------+---------+---------+---------
ACCOUNT             NAME        PHONE          INCOME  BRANCH
---------+---------+---------+---------+---------+---------+---------+---------
DSNE610I NUMBER OF ROWS DISPLAYED IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

Figure 10-43 Impact of default predicate on query result

The default row permission created by DB2 is not limited to control row access for only SQL SELECT statements. Rather, it applies to all SQL DML operations such as SELECT, INSERT, UPDATE, DELETE, and MERGE.

According to the rules for row permissions that are defined with the ENFORCED FOR ALL ACCESS ENABLE clause, you can DELETE, UPDATE, or INSERT only rows that satisfy the row permission predicate. Because the predicate of the default permission is set to 1=0, none

of the rows can ever be updated, inserted, or deleted. This behavior enforces the rule that you can insert, update, or delete only rows that you can read.

10.5.9 Row permissions

With DB2 10, you use the CREATE PERMISSION statement to create a row permission object. A row permission is a database object that expresses a row access control rule for a specific table. It contains the row access rule as an SQL search condition that describes the condition under which rows of a table can be accessed.

For example, you can use the built-in functions that we discuss in 10.5.5, “Built-in functions” on page 388 to specify for which RACF group, trusted context, or role row access is permitted, and you can use additional filters that are typically allowed in search conditions in a WHERE clause of a subselect.

In our customer table scenario, we activated row access control on the customer table, which imposed a default row permission with a predicate of 1=0 that restricted all table access. In the CREATE PERMISSION statement shown in Example 10-32, we created a row permission object that implements the following policy rules:

- Every database user has CURRENT SQLID special register set to one of the following RACF groups:

DB0B#A	RACF group eligible to process rows of branch A
DB0B#B	RACF group eligible to process rows of branch B
DB0B#C	RACF group eligible to process rows of branch C
DB0B##	RACF group eligible to process all table rows

- The row permission implements row filters to restrict row access to each RACF groups access eligibility.

Example 10-32 Row permission object RA01_CUSTOMER SQL DDL

```
CREATE PERMISSION RA01_CUSTOMERS ON DB2R5.CUSTOMERS
FOR ROWS WHERE
 ((CURRENT SQLID = 'DB0B#A' AND BRANCH = 'A') OR
  (CURRENT SQLID = 'DB0B#B' AND BRANCH = 'B') OR
  (CURRENT SQLID = 'DB0B#C' AND BRANCH = 'C') OR
  (CURRENT SQLID = 'DB0B##')
 ) ENFORCED FOR ALL ACCESS
ENABLE
```

After we created the row permission shown in Example 10-32, the SYSIBM.SYSCONTROLS table contained the following row policies for the customer table:

- Default row permission SYS_DEFAULT_ROW_PERMISSION__CUSTOMER with a default predicate of 1=0
- Row permission RA01_CUSTOMERS with the predicate contained in the search condition of the CREATE PERMISSION shown in Example 10-32

For each row permission, DB2 records column dependencies in the SYSIBM.SYSDEPENDENCIES catalog table. Figure 10-44 shows the column dependency that DB2 recorded for our row permission object.

```
SELECT
   SUBSTR(BNAME    ,01,08) AS BNAME
 , SUBSTR(BSCHEMA  ,01,08) AS BSCHEMA
 , SUBSTR(BCOLNAME ,01,08) AS BCOLNAME
 , SUBSTR(DNAME    ,01,16) AS DNAME
FROM SYSIBM.SYSDEPENDENCIES
  WHERE DNAME = 'RA01_CUSTOMERS' WITH UR;
---------+---------+---------+---------+---------+---------+---------+---------
BNAME     BSCHEMA   BCOLNAME  DNAME
---------+---------+---------+---------+---------+---------+---------+---------
CUSTOMER  DB2R5     BRANCH    RA01_CUSTOMERS
```

Figure 10-44 SYSIBM.SYSDEPENDENCIES RA01_CUSTOMER

Under the SQLID of DB0B#B, we then run a query to select all rows from the customer table. DB2 in this example logically combines the predicates of the default permission with the predicate of the RA01_CUSTOMERS permission as illustrated in Figure 10-45.

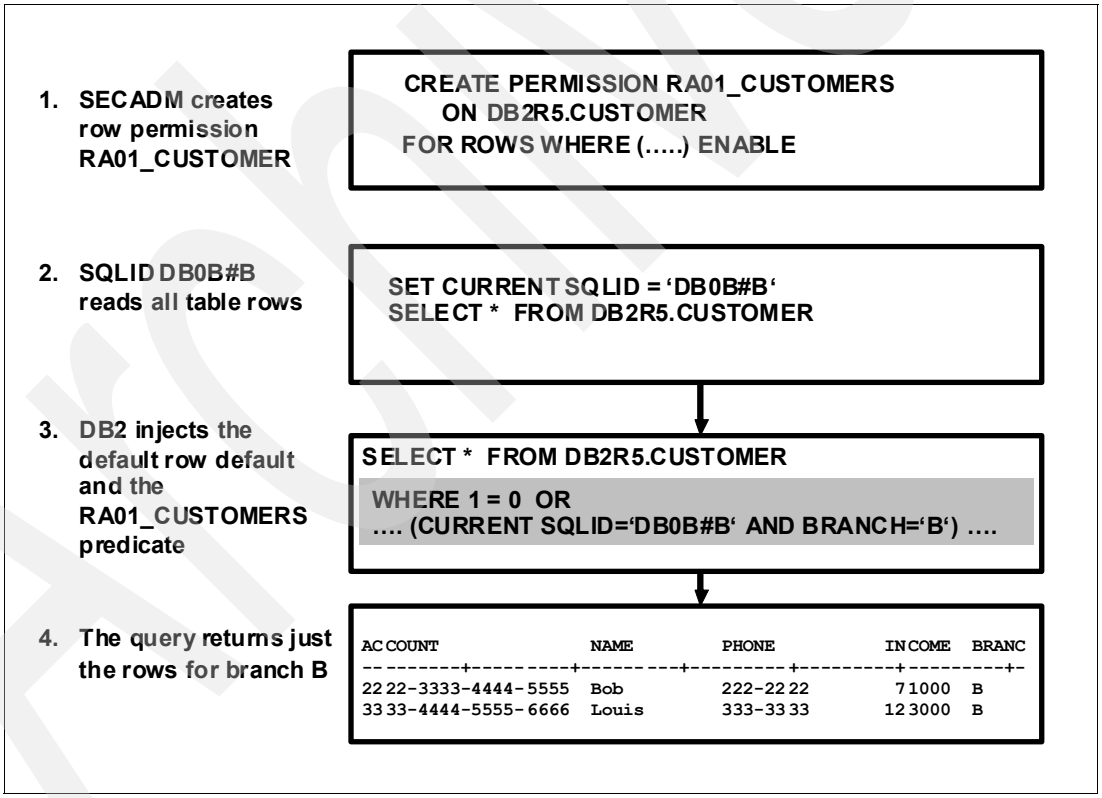


Figure 10-45 Query permission RA01_CUSTOMER without query predicate

When you access a table for which row access control is enforced, DB2 takes any existing row permission into consideration. If multiple row permissions are defined for the table, DB2 logically combines each of them with OR operators, along with the default row permission, and consolidates them into one row access control search condition. This search condition

acts as a filter to the table before any user-specified operations, such as predicates, ordering, or other operations, are processed.

Now, we extend the query shown in Figure 10-45 to select only rows where the income is higher than 71000 as illustrated in Figure 10-46.

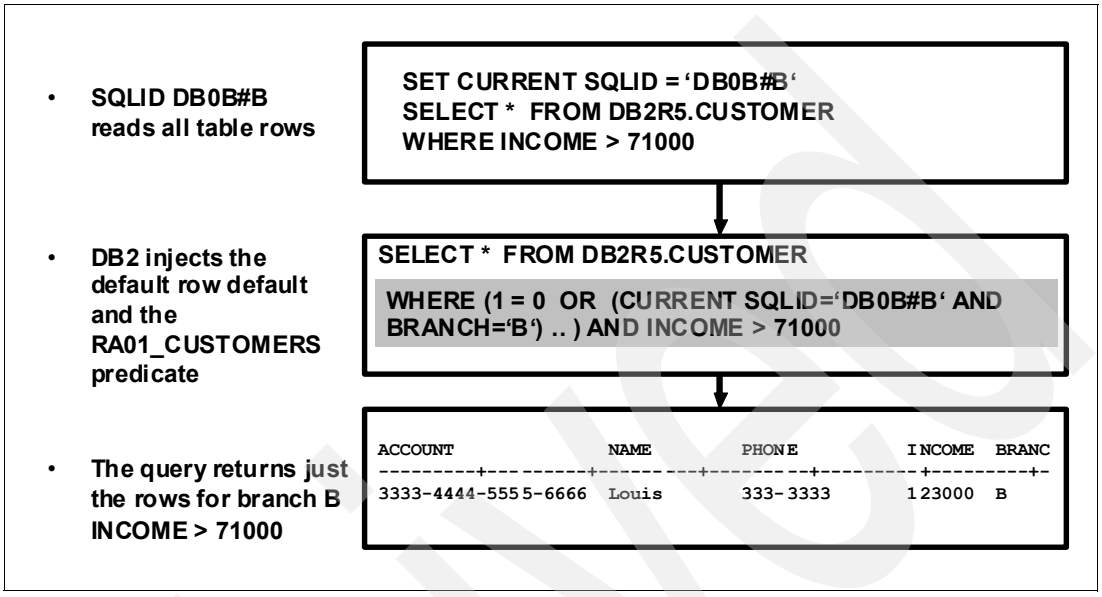


Figure 10-46 Query permission RA01_CUSTOMER and with query predicate

For details about the CREATE PERMISSION SQL statement, refer to *DB2 10 for z/OS SQL Reference,* SC19-2983.

10.5.10 Column access control

After you activate column access control on a table, any SQL DML attempts on that table are subject to column masks that you define and activate on that table. In DB2 10, you define column access control rules through a new object type using column mask objects.

A column mask works much like a scalar function that is implicitly invoked by DB2 when the table column for which you have defined and enabled the column mask is referenced by an SQL DML operation. This approach is transparent to any SQL DML operation and, therefore, integrates easily with existing applications. We discuss this object type in more detail at "Column masks" on page 396.

Now, let us get back to our customer table scenario. We execute the alter table SQL statement shown in Example 10-33 to enforce column access control on the customer table.

Example 10-33 Activate column access control on customer table

```
SET CURRENT SQLID = 'DBOBSECA';  --> SECADM racf group
ALTER TABLE DB2R5.CUSTOMER
   ACTIVATE   COLUMN ACCESS CONTROL ;
```

So far no column mask is defined on any column for the customer table. Therefore, activating column access control on the customer table has no impact yet on column values that are returned by an SQL query.

Before we move on to discuss column masks, we deactivate the row access control on the customer table, because we need RACF group DB0B#A to access all rows on the table when we illustrate the creation of column masks. Example 10-34 shows our alter table statement.

Example 10-34 Deactivate row access control on customer table

```
SET CURRENT SQLID = 'DB0BSECA';  --> SECADM racf group
ALTER TABLE DB2R5.CUSTOMER
   DEACTIVATE ROW ACCESS CONTROL
```

10.5.11 Column masks

With DB2 10, you can use the CREATE MASK statement to create a column mask object. A column mask is a database object that uses a CASE expression to return a masked column value in place of the column value. The column masking is performed in the SQL CASE expression. In the CASE expression WHEN clause, you have many choices on how to conditionally mask a column value. For example, you can use SPECIAL REGISTERS, SESSION VARIABLES, RACF groups, roles, and trusted context names, or you can invoke your own scalar UDFs. You can create multiple column masks per table, one for each table column.

In our customer table scenario, we defined a column mask that assumes the following parameters for the implementation of the column mask:

- The SQLID special register is always set to one of these RACF groups by the application
- The income column value is not to be shown in the report if the branch column value is not within a RACF groups responsibility. (for example, DB0B#A is responsible for branch A but not for branches B and C)

Example 10-35 shows our column mask definition for enforcing that column access policy.

Example 10-35 Column mask INCOME_BRANCH

```
CREATE MASK INCOME_BRANCH ON DB2R5.CUSTOMER    1
  FOR COLUMN INCOME RETURN
   CASE
     WHEN (VERIFY_GROUP_FOR_USER
       (SESSION_USER,'DB0B#A','DB0B#B','DB0B#C') = 1) THEN  2
       CASE
        WHEN (SUBSTR(CURRENT SQLID,6,1) = BRANCH ) THEN INCOME  3
        ELSE NULL    4
       END
    ELSE NULL
   END
 ENABLE; 5
```

The column mask shown in Example 10-35 implements the column access policy requirements through the following column mask attributes:

1. The column mask name is INCOME_BRANCH and is for table column DB2R5.CUSTOMER.INCOME.
2. The column mask returns the income column value when the current authorization ID is connected to one of the RACF groups DB0B#A, DB0B#B, DB0B#C.
3. The last character of the CURRENT SQLID special register matches the branch column value,

4. Otherwise, the column mask returns a NULL value.
5. The column mask is enabled within the create mask statement.

When you define a column mask, DB2 records column dependencies in the SYSIBM.SYSDEPENDENCIES catalog table. As shown in Figure 10-47, DB2 recorded a dependency for the INCOME and the BRANCH column in that catalog table.

```
---------+---------+---------+---------+---------+---------+---------+---------
SELECT
   SUBSTR(BNAME    ,01,08) AS BNAME
 , SUBSTR(BSCHEMA  ,01,08) AS BSCHEMA
 , SUBSTR(BCOLNAME ,01,08) AS BCOLNAME
 , SUBSTR(DNAME    ,01,16) AS DNAME
FROM SYSIBM.SYSDEPENDENCIES
  WHERE DNAME = 'INCOME_BRANCH' WITH UR;
---------+---------+---------+---------+---------+---------+---------+---------
BNAME     BSCHEMA   BCOLNAME  DNAME
---------+---------+---------+---------+---------+---------+---------+---------
CUSTOMER  DB2R5     INCOME    INCOME_BRANCH
CUSTOMER  DB2R5     BRANCH    INCOME_BRANCH
```

Figure 10-47 SYSIBM.SYSDEPENDENCIES rows for INCOME_BRANCH column mask

Example 10-35 defines and enables a column mask on the INCOME table column. Because column access control is already activated for the table, DB2 applies the policy that is defined on the INCOME_BRANCH column mask when a SQL DML operation references the DB2R5.CUSTOMER.INCOME column. Figure 10-48 illustrates the process that applies the column mask on the INCOME column like a virtual scalar function.

Archived

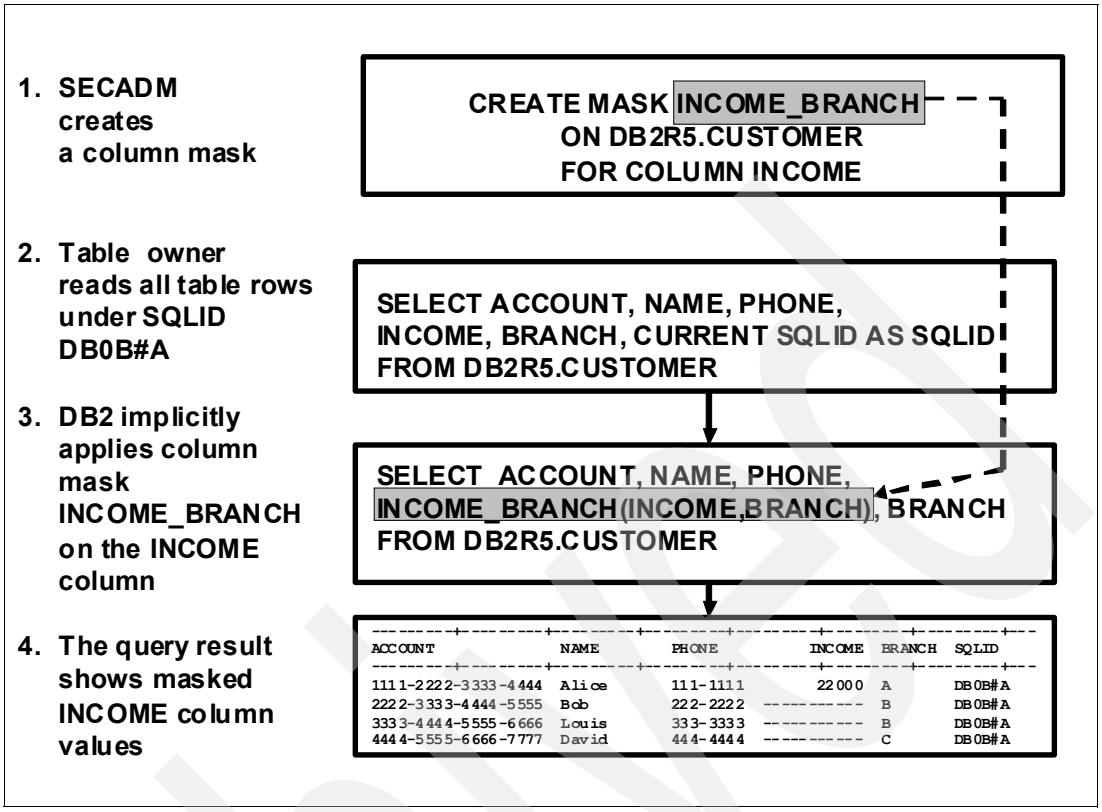


Figure 10-48 Query with column access control with column mask applied

10.5.12 Row permissions and column masks in interaction

During query evaluation DB2 considers row access control for all table references before it processes query specifications, such as operators, predicates, function invocations, and ordering. A row permission acts as a table row filter, and DB2 allows you to access only the table rows that qualify for row selection after the row filter is applied.

DB2 considers column access control for all output columns of the outermost select list after it processes query specifications, such as predicates, GROUP BY, DISTINCT, and ORDER BY. If the select list contains a column that is embedded in an expression, DB2 applies the column mask to that column before it performs expression evaluation.

In the examples that we used in 10.5.8, “Row access control” on page 391 and 10.5.11, “Column masks” on page 396, we illustrated separately the use of the row permission and the column mask objects on the customer table. In this step, we demonstrate what happens when you enforce row permissions and column masks together on the customer table and how they interact with a predicate that you define in your SQL statement. For this step, we perform the following tasks on the customer table:

- Create row permission RA01_CUSTOMER as shown in Example 10-32 on page 393,
- Create column mask INCOME_BRANCH as shown in Example 10-35 on page 396
- Activate row access control as shown in Example 10-31 on page 391
- Activate column access control as shown in Example 10-33 on page 395

Upon task completion, the SYSIBM.SYSCONTROLS table shows the row and column access policy information illustrated in Figure 10-49.

```
---------+---------+---------+---------+---------+---------+---------+----
Row Permission NAME                    TBSCHEMA  TBNAME    ENFORCED  ENABLE
---------+---------+---------+---------+---------+---------+---------+----
INCOME_BRANCH                          DB2R5     CUSTOMER  A         Y
RA01_CUSTOMERS                         DB2R5     CUSTOMER  A         Y
SYS_DEFAULT_ROW_PERMISSION__CUSTOMER   DB2R5     CUSTOMER  A         Y
```

Figure 10-49 Permission and mask policies activated for query sample

After we activated these controls and objects, we selected all rows from the customer table under SQLID DB0B#B. Figure 10-50 shows the query result. The query returned the expected result and included only rows for branch B as determined by the row permission policy.

```
---------+---------+---------+---------+---------+---------+---------+---------
 SELECT * FROM DB2R5.CUSTOMER;
---------+---------+---------+---------+---------+---------+---------+---------
ACCOUNT             NAME       PHONE          INCOME  BRANCH
---------+---------+---------+---------+---------+---------+---------+---------
2222-3333-4444-5555  Bob        222-2222        71000  B
3333-4444-5555-6666  Louis      333-3333       123000  B
```

Figure 10-50 Customer table query for SQLID DB0B#B

We extended the query to return only rows with a salary higher than 80000 as shown in Figure 10-51. DB2 applied the query predicate to the rows that were returned based on the row permission filter.

```
 SELECT * FROM DB2R5.CUSTOMER
  WHERE income > 80000;
---------+---------+---------+---------+---------+---------+---------+---------
ACCOUNT             NAME       PHONE          INCOME  BRANCH
---------+---------+---------+---------+---------+---------+---------+---------
3333-4444-5555-6666  Louis      333-3333       123000  B
```

Figure 10-51 Customer table query for SQLID DB0B#B salary > 80000

Archived

We then used SQLID DB0B#A to access table rows of branch B. As shown in Figure 10-52, the query returned no rows. The statement was successful and returned an SQLCODE of 100, without any warning or error issued by DB2. In this case, the row permission did not return any qualifying row.

```
 SET CURRENT SQLID = 'DB0B#A';
---------+---------+---------+---------+---------+---------+---------+---------
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------
 SELECT * FROM DB2R5.CUSTOMER
  WHERE BRANCH = 'B'  ;
---------+---------+---------+---------+---------+---------+---------+---------
ACCOUNT              NAME        PHONE          INCOME  BRANCH
---------+---------+---------+---------+---------+---------+---------+---------
DSNE610I NUMBER OF ROWS DISPLAYED IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

Figure 10-52 Customer table query for SQLID DB0B#C selecting rows of branch C

Finally, we ran an SQL DML statement under SQLID DB0B#C to update rows of branch B. The update returned an SQLCODE 100 because no row qualifies for row retrieval because SQLID DB0B#C does not qualify to see rows of branch B. DB2 makes sure that SQLID DB0B#C cannot update rows it cannot read. DB2 does not issue an error or warning message.

10.5.13 Application design considerations

With DB2 row and column access control, you can take security logic out of the application logic and place it in row permissions and column masks, making the security rules available to the wider range of applications across the enterprise. With this approach, DB2 transparently provides access protection at the table and column level that is integrated in the database layer. Any application that access data through SQL DML is transparently subject to row permission and column mask control, ensuring data protection regardless of the interface that you use to access data from SQL DML.

10.5.14 Operational considerations

We list additional considerations related to security when dealing with other operational structures here.

Views defined using the WITH CHECK option

A view with the WITH CHECK OPTION clause cannot directly or indirectly be based on a table that is enforced by row or column access control.

Materialized query tables

DB2 10 does not support row and column access control for materialized query tables.

Scalar functions

By default, DB2 marks a UDF as insecure for row and column access control. The UDF's secure attribute is stored in the SECURE column of the SYSIBM.SYSROUTINES catalog table. If the argument of a UDF references a column mask enforced column, the UDF is

always required to be secure, regardless of whether the UDF is referenced in a SELECT list for output or in a predicate.

You can set the secure attribute of an UDF either during UDF creation or by altering the UDF. Setting the secure attribute for the UDF requires SECADM authority or the CREATE_SECURE_OBJECT privilege.

DB2 10 introduces the DSN_FUNCTION_TABLE change shown in Table 10-15 to identify insecure UDFs that are used on row or column access control enforced tables.

Table 10-15 DSN_FUNCTION_TABLE change for row and column access control

Column name	Format	Description
SECURE	CHAR(1) NOT NULL	Whether the UDF is secure ► Y it is secure ► N it is not secure

Triggers on row or column mask enforced tables

A trigger is usually used to enforce database integrity. Thus, a trigger body must have access to data that is not filtered by the row or column access control rules. If you limit the rows that can be accessed from within a trigger, you can compromise database integrity.

If you use a trigger on row and column access enforced tables, your trigger must be defined or altered to be secure. To alter or define a trigger with the secure attribute, you must have either SECADM authority or the CREATE_SECURE_OBJECT privilege. You can alter existing triggers using the alter trigger syntax shown in Figure 10-53.

```
>>-ALTER TRIGGER--trigger-name--+-SECURED-----+----------------><
                               '-NOT SECURED-'
```

Figure 10-53 Alter trigger syntax

Row and column access control and multilevel security

Row access control and multilevel security are mutually exclusive. If you have row access control activated for a table, you cannot alter the table to add a security label column.If your table has a security label column, you cannot activate row access control for that table. You can activate column access control, and you can enable a column mask for a table that has a security label column.

For information about multilevel security, see *Securing DB2 and Implementing MLS on z/OS*, SG24-6480.

Utilities

Most of the utilities, except for the cross-loader function of the LOAD utility and the unload sample program DSNTIAUL, are not using SQL DML. Therefore, row and column access control is not enforced for those utilities.

10.5.15 Performance considerations

Row and column access control introduces an additional layer of security that you can use to complement your security model. When activated on tables, it requires more query processing at execution time to determine rows that are accessible and values within accessible rows that are returned. DB2 considers predicates from row permissions in the access path selection.

In case you face SQL performance issue, you must be able to confirm that the problem was not introduced by row or column access control that is enforced by DB2 on a table that suddenly suffers from bad performance.

Alternatively, the SECADM authority must be aware of the performance impact that a new row permission or column mask definition might have on existing DB2 applications prior to activating row permission and column mask objects in the production environment. For example, enforcing row access or column access control on a single table can potentially invalidate hundreds of packages in DB2. The SECADM authority needs to work closely with an SQLADM authority to assure query performance in the production environments complies with existing service level agreements.

To assist in this task, DB2 provides the following options, which we discuss further in the sections that follow:

- Explain table information
- Dynamic SQL information
- Row permission or column mask impact on package invalidations

Explain table information

When you explain an SQL statement that references a row or column access control enforced table, the plan table shows any table that is referenced by that query at the time the optimizer selected the access path for that SQL statement. This information includes references to tables that are referenced in row permission or column mask objects and not shown in the explained SQL statement. This action allows for row permission predicates and column masks to be included in query analysis and visualization by tools such as IBM Optim Query Tuner.

For illustration purposes, we enforced the row permission shown in Example 10-36 on the customer table in our scenario. The row permission references the table SYSIBM.SYSDUMMY1.

Example 10-36 RA02_CUSTOMERS row permission referencing SYSIBM.SYSDUMMY1

```
CREATE PERMISSION RA02_CUSTOMERS ON DB2R5.CUSTOMER
FOR ROWS WHERE
 ((CURRENT SQLID = 'DBOB#A' AND BRANCH IN
   (SELECT 'A' FROM SYSIBM.SYSDUMMY1) )
 ) ENFORCED FOR ALL ACCESS
ENABLE
```

PLAN_TABLE information

Under user DB2R5, we used the SQL statement in Example 10-37 to explain a query that selects all rows of the customer table.

Example 10-37 RA02_CUSTOMERS explain a query accessing the customer table

```
EXPLAIN ALL SET QUERYNO=578 FOR
   SELECT * FROM DB2R5.CUSTOMER;
```

After the EXPLAIN, we ran a query on PLAN_TABLE to verify the tables that DB2 chose for access path selection. As shown in Figure 10-54, the EXPLAIN provides information about all the tables that are involved in access path selection, even SYSIBM.SYSDUMMY1, which is shown in the SQL query.

```
 SELECT
   SUBSTR(CREATOR,01,08) AS CREATOR
 , SUBSTR(TNAME,01,08) AS TNAME
 , ACCESSTYPE
   FROM DB2R5.PLAN_TABLE
 WHERE QUERYNO = 578;
---------+---------+---------+---------+---------+---------+---------+---------
CREATOR   TNAME     ACCESSTYPE
---------+---------+---------+---------+---------+---------+---------+---------
SYSIBM    SYSDUMMY1 R
DB2R5     CUSTOMER  R
```

Figure 10-54 RA02_CUSTOMERS EXPLAIN output

DSN_PREDICAT_TABLE information

DB2 10 introduces the DSN_PREDICAT_TABLE change shown in Table 10-16 to support additional information about row permission filters and column mask information in SQL explain.

Table 10-16 DSN_PREDICAT_TABLE changes

Column name	Format	Description
ORIGIN	CHAR(1) NOT NULL WITH DEFAULT	Predicate originated from: ► blank - DB2 generated ► C - column mask ► R - row permission ► U - user specified
CLAUSE	CHAR(8) NOT NULL	Clause where the predicate exists: ► SELECT - select clause

In return for the explained SQL statement shown in Example 10-37 on page 402, DB2 externalizes additional information that is related to row permission and column mask objects into the DSN_PREDICAT_TABLE.

Archived

The extract of the query output in Figure 10-55 shows the predicates DB2 injected into the query for the row permission that exists on our customer table. You need this information if you have to reconstruct the full predicate including those injected by DB2 because of row permission and column mask objects.

```
---------+---------+---------+---------+---------+---------+---------+--
SELECT CLAUSE,ORIGIN,TEXT
FROM DB2R5.DSN_PREDICAT_TABLE
WHERE ORIGIN IN ('C','R')
AND QUERYNO = 578;
---------+---------+---------+---------+---------+---------+---------+--
CLAUSE    ORIGIN  TEXT
---------+---------+---------+---------+---------+---------+---------+--
WHERE     R       CURRENT SQLID='DBOB#A'
WHERE     R       DB2R5.CUSTOMER.BRANCH IN (SELECT 'A'  FROM   SYSDUMMY1
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+--
```

Figure 10-55 RA02_CUSTOMERS DSN_PREDICAT_TABLE query output

DSN_STRUCT_TABLE information

DB2 10 introduces the DSN_STRUCT_TABLE change shown in Table 10-16 to support additional information about row permission filters and column mask information in SQL explain.

Table 10-17 DSN_STRUCT_TABLE

Column name	Format	Description
ORIGIN	CHAR(1) NOT NULL WITH DEFAULT	Query block originated from: ► blank - DB2 generated ► C- column mask ► R - row permisson ► U - user specified

In return for the explained SQL statement shown in Example 10-37 on page 402, DB2 externalizes additional information related to row permission and column mask objects into DSN_STRUCT_TABLE. DSN_STRUCT_TABLE provides information such as number of rows returned by the data manager (DM) and the relational data system (RDS) component on query block level and are typically used for detailed access path analysis (for example, IBM Optim Query Tuner).

An extract of the DSN_STRUCT_TABLE query output in Figure 10-56 provides a snapshot of that information provided for illustration.

```
 SELECT TIMES, ROWCOUNT, ORIGIN
   FROM DB2R5.DSN_STRUCT_TABLE
  WHERE ORIGIN = 'R' AND QUERYNO = 578;
---------+---------+---------+---------+---------+---------+---------+---------
                 TIMES        ROWCOUNT  ORIGIN
---------+---------+---------+---------+---------+---------+---------+---------
+0.10000000000000000E+05        10000    R
```

Figure 10-56 TRA02_CUSTOMERS DSN_STRUCT_TABLE query output

Dynamic SQL information

You can collect information about prepared dynamic SQL statements through audit trace IFCID 145. DB2 10 changes the IFCID 145 to support auditing the entire SQL statement text and to indicate row permission and column mask object usage during access path selection process.

You can collect the IFCID 145 trace by starting the audit policy category EXECUTE on the table you want to audit. Refer to 10.1.4, “Policy-based SQL statement auditing for tables” on page 349 for more information about using that audit category.

OMEGAMON PE adds support to format the IFCID 145 trace record, as shown in Figure 10-57.

```
|-------------------------------------------------------------------------------------------------------
|                              AUDIT INFO PREPARED STATEMENT
|
|LOCATION NAME: DBOB                  PKG COLLCT ID: DSNTEP2         PROGRAM NAME : DSNTEP2
|TIME       : 10/27/13 23:28:32.852385   TYPE       : SELECT - QUERY   ISOLATION    : CS
|HOST OPTIONS : X'0400000000000000'        SQL CODE    :         0       STMT #       : 1829
|STMT ID      :                   50       DBID/OBID #  :          1       MASK/PERM #  : 2
|TOTAL LEN STM:       28            LEN STM AO   :       28
|
|LOCATION NAME (LONG):  DBOB
|PKG COLLCT ID (LONG):  DSNTEP2
|PROGRAM NAME (LONG) :  DSNTEP2
|.......................................................................................
|                                TABLE OBID(S) AND DBID(S)
|
|DATABASE: 353    TABLE OBID:       3
|.......................................................................................
|                              ROW AND COLUMN ACCESS CONTROLS
|
|SCHEMA NAME:  DBOBSECA
|OBJECT NAME:  INCOME_BRANCH
|
|SCHEMA NAME:  DBOBSECA
|OBJECT NAME:  RAO2_CUSTOMERS
|.......................................................................................
|                           SQL STATEMENT TEXT (MAY BE TRUNCATED)
|
|SELECT * FROM DB2R5.CUSTOMER
|-------------------------------------------------------------------------------------------------------
```

Figure 10-57 OMEGAMON PE formatted audit report for audit category EXECUTE

Row permission or column mask impact on package invalidations

When a row permission or column mask becomes active for a table, all packages and dynamic statement cache entries that reference that table are invalidated. In a highly frequented production environment, you want to activate row permission or column mask objects in a controlled manner, knowing exactly the impact that the change will have and being prepared to back out the change if it impacts application performance negatively. Prior to activating the change, you want to know the packages and dynamic statement cache entries that the change will invalidate. As with any other SQL change, your row permission and column mask object change needs to go through stress and integration testing to assure application performance will not suffer.

Archived

To determine the packages that will be invalidated when the row permission becomes active for a table, you can query the SYSIBM.SYSPACKDEP table as shown in Figure 10-58.

```
 select
  strip(bqualifier concat '.' concat bname) as "Table Name",
  strip(dcollid concat '.' concat dname   ) as "Collid.PackageID"
  from sysibm.syspackdep
  where
  (bqualifier,bname) = ('DB2R5','AUDEMP')
  with ur
---------+---------+---------+---------+---------+---------+---------+---------
Table Name                     Collid.PackageID
---------+---------+---------+---------+---------+---------+---------+---------
DB2R5.AUDEMP                   AUDSQL.AUDSQL
```

Figure 10-58 Query SYSIBM.SYSPACKDEP

To assess the impact on statements stored in the dynamic statement cache, you can externalize the dynamic statement cache information to your local DSN_STATEMENT_CACHE_TABLE table and subsequently analyze the SQL statements stored in that table. For analysis, you can run submit an SQL query to look for your table name in the SQL_TEXT column. Figure 10-59 shows a sample of such a query.

```
EXPLAIN STMTCACHE ALL;
---------+---------+---------+---------+---------+---------+---------+---------
SELECT
 STMT_TEXT
 FROM DB2R5.DSN_STATEMENT_CACHE_TABLE
 WHERE STMT_TEXT LIKE '%DB2R5.AUDEMP%'
 ;
---------+---------+---------+---------+---------+---------+---------+---------
STMT_TEXT
---------+---------+---------+---------+---------+---------+---------+---------
INSERT INTO DB2R5.AUDEMP VALUES ( '300000' , 'JOSEF' , ' ' , 'KLITSCH' , 'X00'
SELECT * FROM DB2R5.AUDEMP
DELETE FROM DB2R5.AUDEMP WHERE EMPNO = '300000'
INSERT INTO DB2R5.AUDEMP VALUES ( '300001' , 'JOSEF' , ' ' , 'KLITSCH' , 'X00'
DELETE FROM DB2R5.AUDEMP WHERE EMPNO = '300001'
```

Figure 10-59 Query DSN_STATEMENT_CACHE_TABLE

10.5.16 Catalog changes for access control

To support row and column access control, DB2 10 introduces changes to the DB2 catalog that we describe in this section.

SYSIBM.SYSCOLUMNS

DB2 10 adds the column shown in Table 10-18 to the SYSIBM.COLUMNS catalog table to support column access control.

Table 10-18 Access control changes for SYSIBM.SYSCOLUMNS

Column name	Format	Description
CONTROL_ID	INTEGER NOT NULL WITH DEFAULT	DB2 identifier of the column access control mask that is defined for this column. The value is zero (0) if no column access control mask is defined for the column.

SYSIBM.SYSCONTROLS

DB2 10 introduces the SYSIBM.SYSCONTROLS catalog table to support row and column mask objects. The table contains one row for each row permission and column mask. Table 10-19 shows the structure of the table.

Table 10-19 Access control new table SYSIBM.SYSCONTROLS

Column name	Format	Description
SCHEMA	VARCHAR(128) NOT NULL	Schema of the row permission or column mask
NAME	VARCHAR(128) NOT NULL	Name of the row permission or column mask
OWNER	VARCHAR(128) NOT NULL	Owner of the row permission or column mask
OWNERTYPE	CHAR(1) NOT NULL	Indicates the type of the owner: ► Blank: An authorization ID ► L: Role
TBSCHEMA	VARCHAR(128) NOT NULL	Schema of the table for which the row permission or column mask is defined
TBNAME	VARCHAR(128) NOT NULL	Name of the table for which the row permission or column mask is defined
TBCORRELATION	VARCHAR(128) NOT NULL	If specified, the correlation name of the table for which the row permission or column mask is defined. Otherwise, the value is an empty string.
COLNAME	VARCHAR(128) NOT NULL	Column name for which the column mask is defined. If the value is left blank, this is a row permission.
COLNO	SMALLINT NOT NULL	Column number for which the column mask is defined. The value is zero (0) if this is a row permission.
CONTROL_ID	INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY	DB2 access control ID

Archived

Column name	Format	Description
CONTROL_TYPE	CHAR(1) NOT NULL	Indicates the type of the access control object: ► R: Row permission ► M: Column mask
ENFORCED	CHAR(1) NOT NULL	Indicates the type of the access enforced by the row permission. Column mask always has a value of A. ► A: All access
IMPLICIT	CHAR(1) NOT NULL	Indicates whether the row permission was implicitly created: ► N: The row permission was explicitly created or this is a column mask ► Y: The row permission was implicitly created
ENABLE	CHAR(1) NOT NULL	Indicates whether the row permission or the column mask is enabled for access control: ► N: Not enabled ► Y: Enabled
STATUS	CHAR(1) NOT NULL	Indicates the status of the row permission or column mask definition: ► Blank: The definition of the row permission or column mask is complete. ► R: An error occurred when an attempt was made to regenerate the row permission or column mask
CREATEDTS	TIMESTAMP NOT NULL	The timestamp when the row permission or column mask was created
RELCREATED	CHAR(1) NOT NULL	The release of DB2 in which the row permission or column mask was created
ALTEREDTS	TIMESTAMP NOT NULL	The timestamp when the row permission or column mask was last changed
REMARKS	VARCHAR(762) NOT NULL	A character string provided by using the COMMENT ON statement
IBMREQD	CHAR(1) NOT NULL	A value of Y indicates that the row came from the basic machine-readable material (MRM) tape
ENVID	INTEGER NOT NULL	DB2 identifier of the environment
ROWID	ROWID	Row identifier to support LOB columns in the table
RULETEXT	CLOB(2MB) NOT NULL	The source text of the search condition or expression portion of the CREATE PERMISSION or CREATE MASK statement
DESCRIPTOR	BLOB(2MB) NOT NULL	DB2 description of the row permission or column mask

Archive

SYSIBM.SYSDEPENDENCIES

DB2 10 adds column values to the columns shown in Table 10-20 to the SYSIBM.SYSDEPENDENCIES catalog table to support row and column access control.

Table 10-20 Access control changes for SYSIBM.SYSDEPENDENCIES

Column name	Format	Description
BTYPE	CHAR(1) NOT NULL	Type of the object that is identified by BNAME, BSCHEMA, and BCOLNAME ► F: Function ► E: Distinct type ► C: Column ► T: Table ► M: Materialized query table ► V: View ► A: Alias ► S: Synonym
DTYPE	CHAR(1) NOT NULL	Type of the object that is identified by DNAME, DSCHEMA, and DCOLNAME: ► I: Index ► X: Row permission ► Y: Column mask

SYSIBM.SYSOBJROLEDEP

DB2 10 adds the column values to the column shown in Table 10-21 to the SYSIBM.SYSOBJROLEDEP catalog table to support row and column access control.

Table 10-21 Access control changes to SYSIBM.SYSOBJROLEDEP

Column name	Format	Description
DTYPE	CHAR(1) NOT NULL	Type of the dependent object in column DNAME: ► X: Row permission ► Y: Column mask

SYSIBM.SYSROUTINES

DB2 10 adds the column shown in Table 10-22 to the SYSIBM.SYSROUTINES catalog table to support column and row access control.

Table 10-22 Access control changes to SYSIBM.SYSROUTINES

Column name	Format	Description
SECURE	CHAR(1) NOT NULL WITH DEFAULT 'N'	Indicates whether the routine is secured: ► Y: The routine is secured ► N: The routine is not secured

Archived

SYSIBM.SYSTABLES

DB2 10 adds the column shown in Table 10-23 to the SYSIBM.SYSTABLES catalog table to support row and column access control.

Table 10-23 Access control changes to SYSIBM.SYSTABLES

Column name	Format	Description
CONTROL	CHAR(1) NOT NULL WITH DEFAULT	Indicates whether access to the table is enforced using row or column access control: ► Blank: No access control enforcement, this is the DB2 provided default ► R: Row access control enforced ► C: Column access control enforced ► B: Both row and column access control enforced

SYSIBM.SYSTRIGGERS

DB2 10 adds the columns shown in Table 10-24 to the SYSIBM.SYSTRIGGERS catalog table to support row and column access control.

Table 10-24 Access control changes to SYSIBM.SYSTRIGGERS

Column name	Format	Description
SECURE	CHAR(1) NOT NULL WITH DEFAULT 'N'	Indicates the trigger is secured or not: ► Y: The trigger is secured ► N: The trigger is not secured
ALTEREDTS	TIMESTAMP NOT NULL	Time the last ALTER statement was executed for this trigger

SYSIBM.SYSUSERAUTH

DB2 10 adds the columns shown in Table 10-25 to the SYSIBM.SYSUSERAUTH catalog table to support row and column access control.

Table 10-25 Access control changes to SYSIBM.SYSUSERAUTH

Column name	Format	Description
AUTHHOWGOT	CHAR(1) NOT NULL	Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor. ► E: SECADM ► G: ACCESSCTRL
CREATESECUREAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the GRANTEE can create secured objects (triggers and UDFs): ► Blank: Privilege is not held ► Y: Privilege is held without the GRANT option

Archive

10.5.17 Added and changed IFCIDs

To support row and column access control DB2 10 introduces changes to the IFCIDs shown in Table 10-26.

Table 10-26 Changed IFCIDs for row and column access control

IFCID	Description
62	Start statement execution. Changed to reflect new SQL statements for row permission and column mask administration
63	Bind of dynamic SQL. Changed to reflect new SQL statements for row permission and column mask administration
140	Records authorization failures. Changed to reflect create secure object activities.
145	Audit trace record for prepare SQL. Provides information about row permission and column mask names that are implicitly used by DB2 for row filtering and column masking.
271 (Added)	An added trace IFCID record that is written by DB2 when a row permission or column mask is created, dropped, or altered.
361	Audit administrative authorities. Provide access control information about drop table, truncate table, utility exemption, and create secure object activities.

10.6 Support for z/OS security features

In this section, we discuss the following z/OS security features:

- z/OS Security Server password phrase
- z/OS Security Server identity propagation

10.6.1 z/OS Security Server password phrase

A traditional z/OS password can be up to eight characters in length, and the characters that you can use in the password are somewhat limited. For security purpose, such textual passwords are considered “weak” passwords because they are easier to guess or to crack. Thus, you must change these types of passwords frequently and obey strict password rules, which can make them hard to remember.

With a *password phrase*, you can use a memorable phrase and use that to authenticate your identity with the database. You can use uppercase letters, lowercase letters, special characters, or even spaces. Due to the complexity of a password phrase (up to 100 characters in z/OS), it is almost impossible to guess and unlikely for some unauthorized person to hack the password. However, at the same time, it should be easier for the user to remember.

DB2 for z/OS operates in a distributed environment in which it has to support password phrases when connecting to remote database servers or when acting as a database server. To meet that challenge, DB2 10 adds support for password phrases to support password information of up to 100 characters in length when acting as server or client. DB2 assumes the password information provided for authentication to be a password phrase if the password information contains more than eight characters.

Archived

In this section, we use the following scenarios to illustrate the use of password phrases in DB2 for z/OS:

- Using SPUFI to connect to a remote location
- Using a JDBC type 4 application to connect to DB2 for z/OS
- Using an embedded static SQL connect statement from a COBOL program

Prerequisites

Figure 10-60 illustrates the demonstration environment that we used to run the sample scenarios.

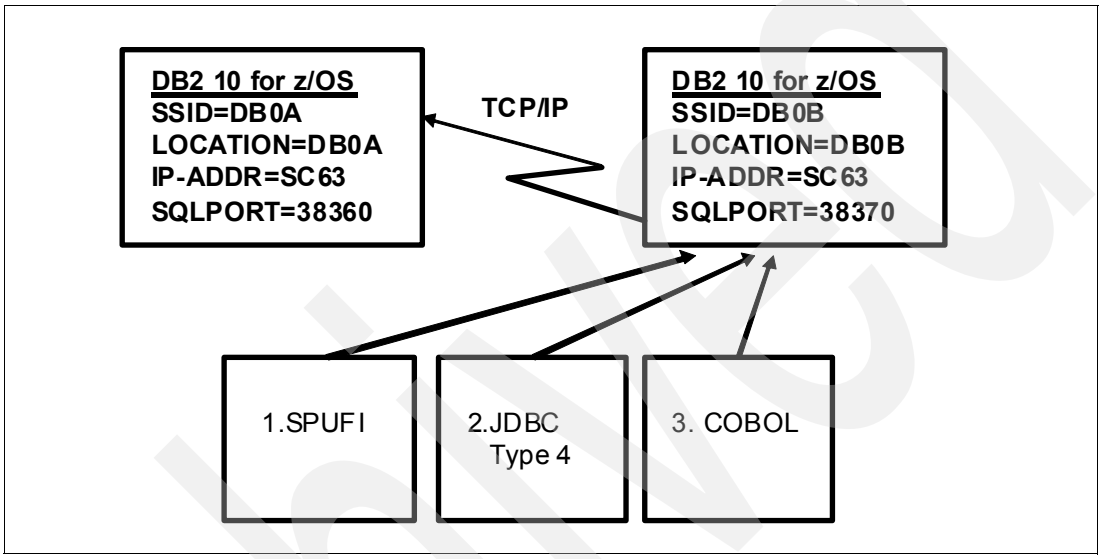


Figure 10-60 DRDA Environment for password phrase scenario

We performed the following tasks to prepare our environment for the scenarios that authenticate with DB2 using a password phrase:

- Create RACF user with password phrase:

  ```
  ADDUSER DB2R56 DFLTGRP(SYS1) OWNER(DB2R5) PHRASE('INITIAL PASSWORD')
  ```

  The user profile is owned by the system administrator's user ID, DB2R5, and is a member of the existing default group SYS1.

- Change the initial password phrase using the DB2 command line processor as shown in Example 10-38.

  Example 10-38 Change the password phrase using DB2 CLP

  ```
  connect to db0b user db2r56 using 'INITIAL PASSWORD'
         new 'So Long, and Thanks for all the Fish'
     confirm 'So Long, and Thanks for all the Fish';
  ```

- Configure the communications database to set up connectivity between DB2 systems DB0B and DB0A. In DB2 system DB0B, we ran the SQL insert statements shown in Example 10-39.

  Example 10-39 Configure DB2 communications database

  ```
  INSERT INTO SYSIBM.USERNAMES
    ( TYPE, AUTHID, LINKNAME, NEWAUTHID, PASSWORD)
   VALUES ('O',' ' ,'DBOA',' ',' ') ;
  ```

```
INSERT INTO SYSIBM.LOCATIONS (LOCATION,LINKNAME,PORT)
 VALUES ('DB0A','DB0A','38360');
INSERT INTO SYSIBM.IPNAMES
( LINKNAME ,SECURITY_OUT,USERNAMES,IPADDR  )
 VALUES('DB0A','P','O','9.12.6.70');
```

- We configured DB2 system DB0A to expect a password passed by remote clients. In DB0A, we configured DSNZPARM TCPALVER = SERVER_ENCRYPT.
- Install the DSNLEUSR stored procedure.

  DSNLEUSR is installed by installation job DSNTIJRT. The DSNLEUSR stored procedure requires ICSF to be configured and operational. DSNLEUSR uses the ICSF CSNBCKM and CSNBENC APIs to perform password and new authorization ID encryption.

Using SPUFI to connect to a remote location

You can use DB2I SPUFI to connect to a remote location. The SQL connect statement that is used by SPUFI connects to a remote location without specifying user ID and password in the SQL connect statement. This connection causes problems in situations in which the remote DB2 server is set up to expect the password to be passed along with the user ID, which is the case for the remote DB2 system, DB0A, in our scenario.

As a solution to this issue, DB2 supports DRDA outbound user ID and password translation. The translation rules are stored in the SYSIBM.USERNAMES table. For DRDA outbound translation, you must insert a row into the USERNAMES table that specifies the originating user ID, the new user ID, and the password of the new user ID. Based on this information, DB2 performs outbound translation when a client connects to that remote location without providing user ID and password.

In DB2 10, the CDB table SYSIBM.USERNAMES now supports password phrases (see Table 10-27).

Table 10-27 SYSIBM.USERNAMES change

Column name	Format	Description
PASSWORD	VARCHAR(255) NOT NULL WITH DEFAULT	Password to accompany an outbound request, CCSID 1208

Additionally, DB2 10 changes the DSNLEUSR stored procedure to support passwords of up to 100 characters in length. DB2 uses a VARCHAR(255) format to cater for additional column space that might be required in case an EBCDIC encoded password phrase is expanded due to the conversion into Unicode (UTF-8) encoded characters.

To protect the user password from misuse you use the DSNLEUSR stored procedure to insert the outbound translation rule into table SYSIBM.USERNAMES. Prior to inserting the row DSNLEUSR performs encryption on the NEWAUTHID and PASSWORD information to protect the user credentials from misuse. During outbound translation DB2 decrypts the NEWAUTHID and PASSWORD information respectively. We used the DB2 UNIX System Services Command Line Processor (CLP) command shown in Example 10-40 to invoke the DSNLEUSR stored procedure to perform the insert into the SYSIBM.USERNAMES table.

Example 10-40 DSNLEUSR stored procedure invocation

```
CALL SYSPROC.DSNLEUSR
 ('O','DB2R5','DB0A','DB2R56',
 'So Long, and Thanks for all the Fish' ,?,?)
```

As shown in Figure 10-61, the NEWAUTHID and PASSWORD information in SYSIBM.USERNAMES is encrypted.

```
---------+---------+---------+---------+---------+---------+---------+---------
SELECT
 SUBSTR(NEWAUTHID,1,38) AS NEWAUTHID,SUBSTR(PASSWORD,1,38) AS PASSWORD
FROM SYSIBM.USERNAMES
 WHERE AUTHID='DB2R5' AND LINKNAME='DB0A'
---------+---------+---------+---------+---------+---------+---------+---------
NEWAUTHID                              PASSWORD
---------+---------+---------+---------+---------+---------+---------+---------
..6b05d2225d6bbe0383c92894             ..8e0c40c74a9469e692da59730aea8be0f20
```

Figure 10-61 SYSIBM.USERNAMES row DSNLEUSR encrypted

After we provide the outbound translation credentials in the SYSIBM.USERNAMES table, we use SPUFI to attach DB0B to the DB2 system as user DB2R5. Then, we use a three-part table name to connect to the remote DB2 for z/OS system, DB0A, to return the current server and the user special register that is used by the remote location for verification. The query result shown in Figure 10-62 shows a successful connection to the DB2 system DB0A for user DB2R56 with that user's password phrase.

```
 SELECT CURRENT SERVER, USER  FROM DB0A.SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+---------+---------
DB0A            DB2R56
```

Figure 10-62 SPUFI and outbound translation with password phrase

If you want to track outbound translation that is performed by the DB2 for z/OS client, you can collect the IFCID 169 audit trace record. For our usage scenario, we captured the IFCID 169 trace record shown in Figure 10-63. The collected information shows that DB2 subsystem DB0B successfully translated user ID DB2R5 into DB2R56, decrypted the password phrase, and passed the password phrase stored in the SYSIBM.USERNAMES catalog table to the remote location DB0A.

```
 TYPE                                  DETAIL
-------- ----------------------------------------------
AUTHCHG  TYPE:                 OUTBOUND DDF TRANSLATION
         PREVIOUS AUTHID:      DB2R5
         NEW AUTHID:           DB2R56
         RESPOND LOCATION:     DB0A
```

Figure 10-63 OMEGAMON PE RECTRACE report on DRDA outbound translation

Using a JDBC type 4 application to connect to DB2 for z/OS

Within this scenario, we use the DB2-provided TestJDBC.java application to connect to the DB2 system DB0B through JDBC type 4. The TestJDBC application receives the JDBC connection attributes from the command line, allowing for type 2 and type 4 connection testing. Upon successful database connection, the TestJDBC application reads a couple of rows from SYSIBM.SYSTABLES and outputs the obtained table schema and table name to the stdout destination and terminates. To provide evidence of the DB2R56 user ID being used, we additionally selected the USER special register when reading the

SYSIBM.SYSTABLES table. We invoked the TestJDBC application through the shell script shown in Example 10-41.

Example 10-41 Shell script for invoking the TestJDBC application

```
java TestJDBC jdbc:db2://9.12.6.70:38370/DBOB DB2R56 \
"So Long, and Thanks for all the Fish"
```

As shown in Example 10-41, we provided all connection attributes, including the password phrase that is embedded in double quotation marks in the TestJDBC interface. The TestJDBC application returned the output shown in Figure 10-64. The CURRENT USER information confirms that DB2R56 is connected to DB2 using his password phrase information. Because user DB2R56 has no textual password, the password phrase was used.

```
Loading DB2 JDBC Driver: com.ibm.db2.jcc.DB2Driver
  successful driver load, version 3.59.83

Establishing Connection to URL: jdbc:db2://9.12.6.70:38370/DBOB
  successful connect

Acquiring DatabaseMetaData
  successful... product version: DSN10015
Creating Statement
..............
About to fetch from ResultSet, maxRows=10
    CREATOR: <DB2R5> NAME: <AUDDEPT>  CURRENT_USER: <DB2R56>
    CREATOR: <DB2R5> NAME: <AUDEMP>  CURRENT_USER: <DB2R56>
    CREATOR: <DB2R5> NAME: <CUSTOMER>  CURRENT_USER: <DB2R56>
successful processing of ResultSet
============================
SUCCESS... TestJDBC complete
============================
```

Figure 10-64 TestJDBC output

Obtaining the test application: You can find the DB2-provided TestJDBC.java application in the /usr/lpp/db2a10_jdbc/samples directory. For details about how to use the TestJDBC application, refer to the inline documentation provided inside the JAVA program.

SQL connect statement from a COBOL program

Within this scenario, we demonstrate how to use a password phrase to connect to a remote DB2 location from a COBOL program. As shown in Example 10-42, there is a character string that represents the password phrase. You still have to provide a user ID and the password-related information in host variables prior to connecting to the remote DB2 location. Even the host variable for the password information already supports the maximum password phrase length of 100 characters.

Example 10-42 COBOL logic for connecting to a remote location

```
Procedure Division.
 A00-CONTROL SECTION.
 A0010.
     SET MSG1 TO TRUE.
     MOVE 'So Long, and Thanks for all the Fish' TO PWPHRASE.
```

```
     MOVE 'DB2R56' TO AUTHID.
     MOVE 'DB0A'   TO LOCATION.
     EXEC SQL
          CONNECT TO :LOCATION USER :AUTHID USING :PWPHRASE
     END-EXEC.
     PERFORM U00-Handle-Result.
     SET MSG2 TO TRUE.
     EXEC SQL SET :CUSER = USER END-EXEC.
     PERFORM U00-Handle-Result.
 A0099.
     goback.
```

While we ran the program in Example 10-42, we collected the information shown in Figure 10-65 in the remote DB2 for z/OS system. This information shows that user DB2R56 connected to the remote DB2 subsystem DB0A successfully using password phrase information.

```
-dis thd(batch) detail
 DSNV401I  -DB0A DISPLAY THREAD REPORT FOLLOWS -
 DSNV402I  -DB0A ACTIVE THREADS -
 NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
 BATCH    RA *     1 DB2R5R06     DB2R56   PWPHRCON 00A4   192
  V441-ACCOUNTING=ACCNT#
  V445-USIBMSC.SCPDB0B.C67AB6AE38CE=192 ACCESSING DATA FOR
   (  1)::9.12.6.70
  V447--INDEX SESSID          A ST TIME
  V448--(  1) 38360:1183      W R2 1023620434684
 DISPLAY ACTIVE REPORT COMPLETE
```

Figure 10-65 DISPLAY THREAD taken during COBOL SQL connect scenario

10.6.2 z/OS Security Server identity propagation

The idea of user identity mapping (also known as *user identity propagation*) across integrated servers that do not share user registries is already known in DB2 for z/OS. Starting with DB2 9 for z/OS, you can use the z/OS Security Server V1R8 provided SAF user mapping plug-in service and implement the DB2 support for Enterprise Identity Mapping (EIM). However, due to its complexity, the implementation in DB2 is time consuming and costly.

For example, for the EIM implementation in DB2, you must configure the z/OS LDAP server, set up RACF for LDAP, set up the EIM domain controller, and add the SAF user mapping plug-in load library to the z/OS linklist.

In z/OS Security Server V1R11, the EIM function is fully integrated into the RACF database. As a result, you no longer have to provide the LDAP server and EIM domain controller infrastructure that is required for the DB2 EIM implementation. This function provides a more integrated approach to manage user IDs across the enterprise. In z/OS Security Server V1R11, this feature is described as *z/OS identity propagation*.

z/OS identity propagation provides a consistent method of mapping distributed user identities into RACF user IDs by means of distributed identity filters. A distributed identity filter is a mapping association between a RACF user ID and one or more distributed user identities, because they are known to distributed application servers and are defined in distributed user registries. In other words, a distributed identity filter consists of one or more components of a

distributed user's name and the name of the registry where the user is defined. To administer these distributed identity filters, use the RACF RACMAP command to associate (or map) a distributed user identity with a RACF user ID.

During DB2 remote connection authentication processing, when RACF accepts information about the identities of distributed users from DB2 (using RACROUTE REQUEST=VERIFY macro request), RACF attempts to derive the RACF user ID.

For more information about how to use a distributed-identity name filter to map distributed identities to a RACF user ID, refer to *z/OS V1R11.0 Security Server RACF Security Administrator's Guide,* SA22-7683.

Important: Be aware that the SAF user identity mapping plug-in service will not be supported in the future release of DB2 for z/OS. Therefore, we encourage you to make use of the new z/OS identity propagation in DB2 10 for z/OS.

Implementing and testing a distributed identity filter

In the sample scenario shown in Figure 10-66, we demonstrate the tasks that you need to perform in DB2 and in RACF to prepare a distributed identity user for trusted context authorization ID switching. First, we discuss the prerequisites needed. Then, we explain how to define a distributed identity filter in RACF using the RACMAP command and how to define the DB2 trusted context and the database role in DB2 that are to be used for the distributed identity user. We then demonstrate how the TrustedContextDB2zOS application connects to DB2 through a DB2 trusted context and how our distributed identity user, DataJoe, can be used for trusted context authorization ID switching.

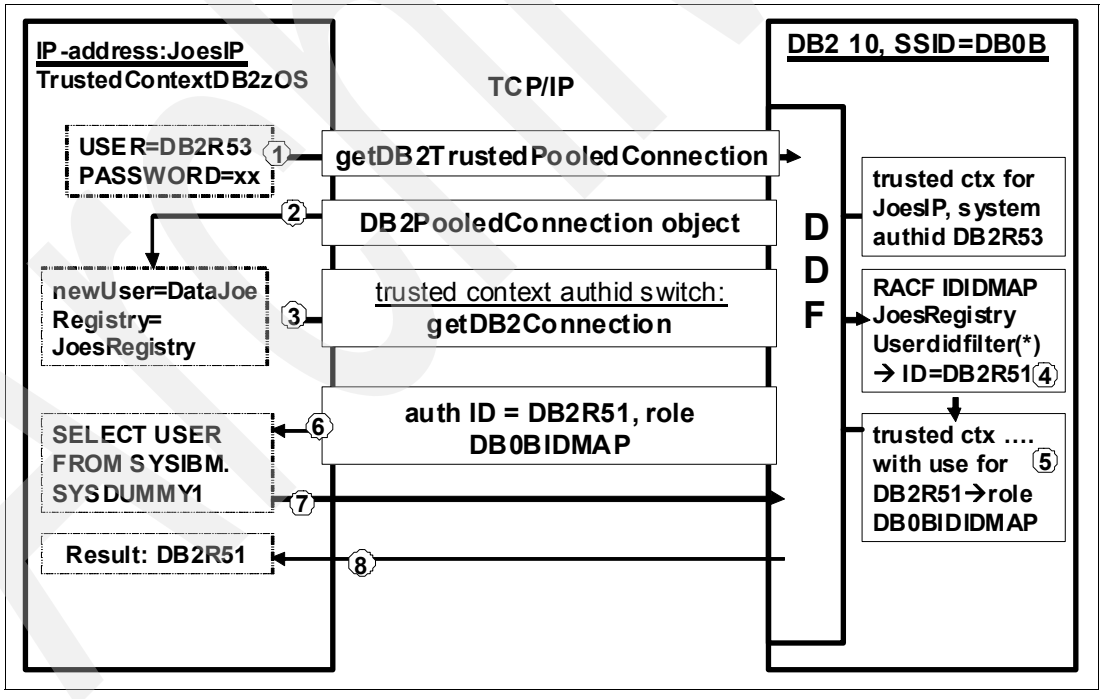


Figure 10-66 z/OS identity propagation TrustedContextDB2zOS flow

Prerequisites

Before you implement distributed identity filters, the z/OS environment must satisfy the following requirements:

- Ensure that DB2 10 is running in a z/OS V1R11 environment.
- Define the RACMAP command as authorized TSO command, which requires a change of the IKJTSOxx member in the z/OS PARMLIB data set. You need to contact your z/OS systems programmer if you need to perform this change. If you are authorized to do so, you can issue the PARMLIB LIST(AUTHCMD) TSO command to verify your current TSO AUTHCMD settings.
- Activate and enable RACF general resource IDIDMAP for RACLIST processing. We ran the commands shown in Example 10-43.

Example 10-43 Activate RACF class IDIDMAP

```
SETROPTS CLASSACT(IDIDMAP) RACLIST(IDIDMAP)
SETROPTS RACLIST(IDIDMAP) REFRESH
```

Creating the identity mapping in RACF

To create the required identity mapping in RACF, we performed the following tasks:

1. Run the RACMAP command as shown in Example 10-44 to create the identity mapping for JoessRegistry. We specified a wildcard for the USERDIDFILTER parameters to allow for any distributed identity user of the distributed identity JoesRegistry to be mapped by RACF to user ID DB2R51.

Example 10-44 RACMAP MAP command to add identity mapping

```
RACMAP ID(DB2R51) MAP USERDIDFILTER(NAME('*')) +
REGISTRY(NAME('JoesRegistry')) +
WITHLABEL('DB2 10 identity propagation')
```

2. Refresh the IDIDMAP class profile as shown in Example 10-45.

Example 10-45 Refresh IDIDMAP class profiles

```
SETROPTS RACLIST(IDIDMAP) REFRESH
```

3. Review the newly created distributed identity filter by running the RACMAP LISTMAP command. Figure 10-67 shows the RACMAP LISTMAP command output.

```
RACMAP ID(DB2R51) LISTMAP

Mapping information for user DB2R51:

  Label: DB2 10 identity propagation
  Distributed Identity User Name Filter:
    >*<
  Registry Name:
    >JoesRegistry<
```

Figure 10-67 RACMAP LISTMAP command output

Creating the trusted context in DB2

You can use z/OS identity propagation only through a DB2 trusted context. In our environment, we used the SQL DDL statement shown in Example 10-46 for the trusted context definition.

Example 10-46 Trusted context definition for identify propagation

```
CREATE  ROLE DB0BIDIDMAP ; 1
GRANT   SELECT ON DB2R5.CUSTOMER TO ROLE DB0BIDIDMAP; 2
CREATE TRUSTED CONTEXT CTXIDID    3
        BASED UPON CONNECTION USING SYSTEM AUTHID DB2R53 4
        ATTRIBUTES (ADDRESS '9.112.46.111') 5
        DEFAULT ROLE DB0BEP              6
        WITHOUT ROLE AS OBJECT OWNER
        ENABLE
        WITH USE FOR DB2R51 ROLE DB0BIDIDMAP ; 7
```

The SQL DDL shown in Example 10-46 creates a database role and a trusted context as follows:

1. The database role DB0BIDIDMAP is used by the TrustedContextDB2zOS application when performing SQL work under distributed identity user DataJoe.
2. Grant a simple select authorization to role DB0BIDIDMAP.
3. Create a trusted context with a name of CTXIDID.
4. The trusted context system authorization ID is DB2R53. We use this authorization ID in our JAVA application when we ask DB2 to create a trusted context based database connection by invoking the getDB2TrustedPooledConnection method.
5. DB2 creates the trusted context based connection only when the connection request comes from authorization ID DB2R53 an when that authorization ID issued the connection request from an IP host with an IP address of JoesIP.
6. The default role for that trusted context is DB0BEP. You see authorization ID DB2R53 is assigned to use role DB0BIDIDMAP.
7. The trusted context WITH FOR clause specifies the name of the RACF user DB2R51, which is assigned through the identity filter that we defined in “Creating the identity mapping in RACF” on page 418.

Implementing the TrustedContextDB2zOS application

Our TrustedContextDB2zOS application is partially modelled on the TrustedContext Java application that ships with DB2 LUW provided samples. In its core, the TrustedContextDB2zOS application uses the following methods:

- getDB2TrustedPooledConnection

  This method creates the trusted context based DB2 connection through JDBC type 4. It passes the connection user ID (DB2R53) and its password into DB2. For illustration purposes, we provide the code part showing the getDB2TrustedPooledConnection in Figure 10-68.

  At the time the JAVA program invokes the getDB2TrustedPooledConnection method all required data source properties like the ServerName, the PortNumber, the DatabaseName and the DriverType are initialized successfully with the parameter values that are required to successfully connect to DB2 subsystem DB0B through the trusted context CTXIDID.

Archived

```
String user  = new String("DB2R53");
String password  = new String("one2many");
com.ibm.db2.jcc.DB2ConnectionPoolDataSource ds1 =
    new com.ibm.db2.jcc.DB2ConnectionPoolDataSource();
ds1.setServerName("wtsc63.itso.ibm.com");
ds1.setPortNumber(38370);
ds1.setDatabaseName("DBOB");
ds1.setDriverType (4);
java.util.Properties properties = new java.util.Properties();
try
{objects = ds1.getDB2TrustedPooledConnection(
        user,password, properties);}
catch(Exception ex)
```

Figure 10-68 getDB2TrustedPooledConnection

- getDB2Connection

  This method uses the connection that we created in Figure 10-68 and asks DB2 to perform a trusted context authorization ID switch on that trusted context connection. As illustrated in Figure 10-69, the getDB2Connection method passes the distributed identity user (DataJoe) and the distributed registry name (JoesRegistry) together with the old user ID (DB2R53) into DB2. Upon a successful trusted context authorization ID switch, the JAVA program issues a SET CURRENT SQL statement to trigger SQL processing in DB2.

```
String user1 = new String("DataJoe");
String user1_pwd = new String(" ");
String registry = new String("JoesRegistry");
try{
     con = pooledCon.getDB2Connection(
     cookie,user1, user1_pwd, registry,
     userSecTkn, originalUser, properties);
     stmt = con.createStatement();
     stmt.execute("SET CURRENT SQLID = USER");}
```

Figure 10-69 getDB2Connection

Additional information: For more information about the getTrustedPooledConnection and getDB2Connection Java methods, refer to the information center at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.apdv.java.doc/doc/c0023797.html

Running the TrustedContextDB2zOS application

In this section, we discuss the TrustedContextDB2zOS application flow that is illustrated in Figure 10-66 on page 417. Referring to that illustration, the processing steps of the TrustedContextDB2zOS application are as follows:

1. The application invokes the getDB2TrustedPooledConnection method to create a trusted context based DB2 connection. The connection authorization ID (DB2R53), the password and the IP address of the IP host our application runs on are passed into DB2 for authentication. DB2 authenticates the user and uses the authorization ID along with the IP-address to locate the trusted context that DB2 uses for trusted context creation. In our scenario DB2 locates the trusted context for the system authorization ID DB2R53 and the

IP-address JoesIP (refer to “Creating the trusted context in DB2” on page 419 for trusted context details).

2. DB2 successfully creates the trusted connection and a connection object is returned to our TrustedContextzOS application. At this point, we took the documentation shown in Figure 10-70 to confirm the successful creation. As shown in the display thread output, the authorization ID DB2R53 is used for connection creation. Furthermore the connection uses the trusted context default role of DB0BEP as defined in trusted context CTXIDID to perform work in DB2.

```
-dis thd(*) type(inactive) detail
 DSNV401I  -DB0B DISPLAY THREAD REPORT FOLLOWS -
DSNV424I  -DB0B INACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   R2       0 db2jcc_appli DB2R53   DISTSERV 0094    64
V485-TRUSTED CONTEXT=CTXIDID,
     SYSTEM AUTHID=DB2R53,
     ROLE=DB0BEP
 V437-WORKSTATION=wtsc63.itso.ibm.co, USERID=DB2R53,
      APPLICATION NAME=db2jcc_application
 V441-ACCOUNTING=JCC03590wtsc63.itso.ibm.co
 V445-G90C0646.G43D.C67C368E32D9=64 ACCESSING DATA FOR
  (  1)::9.12.6.70
 V447--INDEX SESSID           A ST TIME
 V448--(  1) 38370:1085       W R2 1023801181659
DISPLAY INACTIVE REPORT COMPLETE
DSN9022I  -DB0B DSNVDT '-DIS THD' NORMAL COMPLETION
```

Figure 10-70 Display thread output right after trusted context pool connection

3. Our application uses the distributed identity user (DataJoe) and registry (JoesRegistry) to invoke the getDB2Connection method to perform a trusted context authorization ID switch on the existing trusted context connection in DB2.
4. DB2 uses the distributed identity user and registry information to issue a RACROUTE REQUEST=VERIFY macro to request RACF to return the RACF authorization ID that is associated with the distributed identity. RACF in turn locates the matching IDIDMAP class profile and returns the RACF authorization ID (DB2R51) that is associated with it to DB2.
5. DB2 applies the matching trusted context WITH USE FOR clause and assigns the database role DB0BIDIDMAP in return for user DB2R51.

Archived

6. For confirmation we took the documentation shown in Figure 10-71. As you can see DB2 successfully switched the trusted context authorization ID to user DB2R51 and assigned the database role DB0BIDIDMAP. When you compare the DB2 logical unit of work ID (LUW) token provided by DB2 message DSNV445 with the corresponding message DSNV445 shown in Figure 10-70, both messages have the same DB2 LUW ID token representing the same physical connection in DB2.

```
-dis thd(*) type(inactive) detail
 DSNV401I  -DB0B DISPLAY THREAD REPORT FOLLOWS -
DSNV424I  -DB0B INACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   R2       0 db2jcc_appli DB2R51   DISTSERV 0094    64
V485-TRUSTED CONTEXT=CTXIDID,
     SYSTEM AUTHID=DB2R53,
     ROLE=DB0BIDIDMAP
 V437-WORKSTATION=wtsc63.itso.ibm.co, USERID=DB2R53,
      APPLICATION NAME=db2jcc_application
 V441-ACCOUNTING=JCC03590wtsc63.itso.ibm.co
 V445-G90C0646.G43D.C67C368E32D9=64 ACCESSING DATA FOR
  (  1)::9.12.6.70
 V447--INDEX SESSID          A ST TIME
 V448--(  1) 38370:1085      W R2 1023801225355
DISPLAY INACTIVE REPORT COMPLETE
DSN9022I  -DB0B DSNVDT '-DIS THD' NORMAL COMPLETION
```

Figure 10-71 Display thread output right after trusted context switch

7. Under authorization ID DB2R51 the application issues a SET CURRENT SQLID = USER statement to prove that issuing SQL principally works when executed under the database role DB0BIDIDMAP.
8. DB2 returns the content of the NEW SQLID special register to the application which is confirmed by the audit report shown in Figure 10-72.

```
DB2R51   db2jcc_a DRDA        05:55:23.55 AUTHCHG  TYPE:              SET CURRENT SQLID
 DATAJOE  ppli     C67C3ED5C0F5                     PREVIOUS SQLID:    DB2R51
 DISTSERV SERVER                                    NEW SQLID:         DB2R51
 REQLOC  :::9.12.6.70
 ENDUSER :DB2R53
 WSNAME  :wtsc63.itso.ibm.co
 TRANSACT:db2jcc_application
```

Figure 10-72 Audit report trusted context SET CURRENT SQLID

Trusted context auditing

Before we started the application, we prepared and started an audit policy of category type VALIDATE. You can use the audit policy category VALIDATE to track activities that are related to trusted context processing in DB2. For more information about how to create and start and audit policy in DB2, refer to 10.1, “Policy-based audit capability” on page 338. For creating audit reports, we used OMEGAMON XE for DB2 Performance Expert. In addition to the runtime documentation explained in “Running the TrustedContextDB2zOS application” on page 420, we created audit reports that provide detailed information about trusted context authentication processing for the auditing events that we describe in the sections that follow.

Establish trusted context

The audit report in Figure 10-73 is in line with the information that is provided by the display thread command shown in Figure 10-70 as it shows the same RACF authorization ID and DB2 trusted context information.

```
DB2R53   db2jcc_a DRDA         05:55:19.09 AUTHCHG  TYPE:               ESTABLISH TRUSTED CONTEXT
 DB2R53   ppli     C67C3ED5C0F5                      OBJECT OWNER:       AUTHID
 DISTSERV SERVER                                     SECURITY LABEL:
 REQLOC  :::9.12.6.70                                CONTEXT NAME:       CTXIDID
 ENDUSER :DB2R53                                     CONTEXT ROLE:       DB0BEP
 WSNAME  :wtsc63.itso.ibm.co                         USER ROLE:
 TRANSACT:db2jcc_application                         PREV. SYSAUTHID:    DB2R53
                                                     REUSE AUTHID:
                                                     SERVAUTH NAME:
                                                     JOB NAME:
                                                     ENCRYPTION:         NONE
                                                     TCP/IP USED:        JoesIP
```

Figure 10-73 Audit report establish trusted context

Distributed identity: RACF authorization ID mapping

This auditing event provides information about the distributed identity user and the RACF authorization ID mapped to it. The audit report shown in Figure 10-74 confirms the distributed identity user name (DataJoe) and the RACF authorization ID (DB2R51) that was mapped by RACF in return for the RACROUTE REQUEST=VERIFY call issued by DB2.

```
DB2R53   db2jcc_a DRDA        05:55:23.54 AUTHCHG  TYPE:              N/P
 DB2R53   ppli     C67C3ED5C0F5                      IP ADDR:           090C0646
 DISTSERV SERVER                                   DERIVED LOCAL UID:  DataJoe
 REQLOC  :::9.12.6.70                              COMMS ADDR TYPE:    TCP/IP
 ENDUSER :DB2R53                                   PORT:               0443
 WSNAME  :wtsc63.itso.ibm.co                       CLIENT PRODUCT ID:  JCC03590
 TRANSACT:db2jcc_application
  -------- -------- ------------ ----------- -------- ----------------------------------
 DB2R51   db2jcc_a DRDA        05:55:23.54 AUTHCHG  TYPE:              END OF IDENTIFY
 DataJoe  ppli     C67C3ED5C0F5                    PREVIOUS AUTHID:    DataJoe
 DISTSERV SERVER                                   SECONDARY AUTHID:   DB0BIDID    SYS1
 REQLOC  :::9.12.6.70                              STATUS:             SUCCESS
 ENDUSER :DB2R53                                   CURRENT SQLID:      DB2R51
 WSNAME  :wtsc63.itso.ibm.co
 TRANSACT:db2jcc_application
```

Figure 10-74 Audit report distributed identity: RACF authorization ID mapping

Archived

Reuse trusted context

This auditing event provides information about the trusted context authorization ID switching performed by DB2. The audit report in Figure 10-75 confirms the information provided by the display command output shown in Figure 10-71 on page 422 as it confirms trusted context thread reuse with an authorization ID of DB2R51 and a DB2 role of DB0BIDIDMAP.

```
DB2R51   db2jcc_a DRDA         05:55:23.54 AUTHCHG  TYPE:                REUSE TRUSTED
CONTEXT
 DATAJOE  ppli    C67C3ED5C0F5                OBJECT OWNER:     AUTHID
 DISTSERV SERVER                                      SECURITY LABEL:
 REQLOC  :::9.12.6.70                                 CONTEXT NAME:       CTXIDID
 ENDUSER :DB2R53                                      CONTEXT ROLE:       DB0BEP
 WSNAME  :wtsc63.itso.ibm.co                          USER ROLE:          DB0BIDIDMAP
 TRANSACT:db2jcc_application                          PREV. SYSAUTHID:    DB2R53
                                                      REUSE AUTHID:       DB2R51
                                                      SERVAUTH NAME:
                                                      JOB NAME:
                                                      ENCRYPTION:
                                                      TCP/IP USED:
```

Figure 10-75 Audit report trusted context switching

Archived

11

Utilities

DB2 10 includes a variety of improvements to utilities.

Utilities are enhanced to support all new functions in DB2 10, such as universal table space (UTS) enhancements, online schema changes, inline LOBs, XML, and other new functions. A new DB2 Sort for z/OS tool provides high-speed sort processing for data that is stored in DB2 for z/OS and can help the execution of utilities.

Utilities also include more widespread use of the System z platform functions. For example, several utilities integrate the option to use FlashCopy at the table space level, providing improvements in availability and performance.

Finally, DB2 10 utilities show the trend towards providing autonomic utility usage.

In this chapter, we discuss the following topics:

- Support FlashCopy enhancements
- Autonomic statistics
- RECOVER with BACKOUT YES
- Online REORG enhancements
- Increased availability for CHECK utilities
- IBM DB2 Sort for z/OS
- UTSERIAL elimination
- REPORT utility output improvement

We describe utilities enhancements that are related specifically to performance in Chapter 13, “Performance” on page 533.

© Copyright IBM Corp. 2010. All rights reserved.

11.1 Support FlashCopy enhancements

DB2 10 for z/OS provides improvements in the backup and recovery area.

DB2 10 supports the following enhancements in integrating FlashCopy technology in the image copy and recover utilities:

- Support for FlashCopy image copies that are produced at data set level using the FlashCopy technology and registered in SYSIBM.SYSCOPY as any other image copy. The LOAD, REBUILD INDEX, and REORG utilities can use these image copies to produce inline image copies of data and indexes.
- The exploitation of FlashCopy image copies by the COPY, REORG, LOAD, REBUILD INDEX, and RECOVER utilities.
- The ability to create consistent COPY SHRLEVEL CHANGE image copies.

FlashCopy 2 is required, otherwise DB2 will revert to traditional methods. If the target volume is primary volume on a metro mirror remote copy, the Remote Pair FlashCopy function (also called *preserve mirror*) provided by microcode and DFSMSdss OA24811 is also required. A new DSNZPARM FLASHCOPY_PPRC (APAR PM26762) controls the options.

Because FlashCopy works at the single controller level, defining separate storage pools for source and target objects is important.

We describe improvements to RECOVER TABLESPACE in 11.3, “RECOVER with BACKOUT YES” on page 448.

In this section, we first describe DB2 FlashCopy image copies that can be created by the COPY utility. In 11.1.2, “FlashCopy image copies and utilities other than COPY” on page 437, we look at the usage of FlashCopy image copies with other utilities.

11.1.1 Getting started with FlashCopy image copies with the COPY utility

FlashCopy image copy is useful for making image copies of large DB2 objects; however, using FlashCopy does not mean that you will obtain better performance than sequential image copies with small DB2 objects. For small objects, you may get better performance using a system level backup, which avoids the cost of data set allocations.

Because FlashCopy image copy does not support incremental copies, do not consider FlashCopy image copy as a replacement for incremental image copies.

Also consider that, if the hardware does not support FlashCopy, the system uses IDCAMS Repro to copy the data set, in which case the only advantage is that Repro does not impact the DB2 buffer pool.

You can ask DB2 to create a FlashCopy image copy with the COPY utility using one of the following options:

- Set DSNZPARM FLASHCOPY_COPY to YES

  This setting lets DB2 create a FlashCopy image copy automatically. You do not need COPYDDN specified on your utility control statement.

If you use FLASHCOPY_COPY=YES and specify the COPYDDN keyword on your utility control statement and your utility job JCL contains a valid DD statement for SYSCOPY, DB2 automatically creates two image copies with one COPY utility execution. You get a FlashCopy image copy and a sequential copy, which is created from the FlashCopy.

- Specify FLASHCOPY YES or FLASHCOPY CONSISTENT on your utility control statement

  Both settings overwrite the setting that you specified for DSNZPARM FLASHCOPY_COPY. If you specify YES or CONSISTENT for the FLASHCOPY utility control statement and if you specify the COPYDDN keyword, DB2 automatically creates two image copies with one utility execution.

The image copy is a VSAM cluster. You can set the name of the VSAM cluster that is allocated for you as part of the COPY utility using one of the following methods:

- Set DSNZPARM FCCOPYDDN as default value for your image copy data sets.

  The FCCOPYDDN subsystem parameter defines the default value that is used for the FCCOPYDDN parameter of the FLASHCOPY clause of the DB2 utility control statement. FCCOPYDDN is meaningful only when you request the creation of a FlashCopy image copy either through the FLASHCOPY utility control statement or by setting DSNZPARM FLASHCOPY_COPY to YES. FCCOPYDDN specifies the template for deriving the data set name for the FlashCopy image copy.

  The default value is `HLQ.&DB..&SN..N&DSNUM..&UQ.`

  The variables that you can use here are the same as for the data set name in the TEMPLATE utility control statements.

  Important: You can choose an appropriate value for your environment; however, the value must include DSNUM or PART.

  You must specify a value here. You cannot leave this DSNZPARM parameter blank. However, during the assembly of the DSNZPARM module, this parameter is not checked if the value specified here ends up in a valid data set name.

- Use the utility control statement keyword FCCOPYDDN.

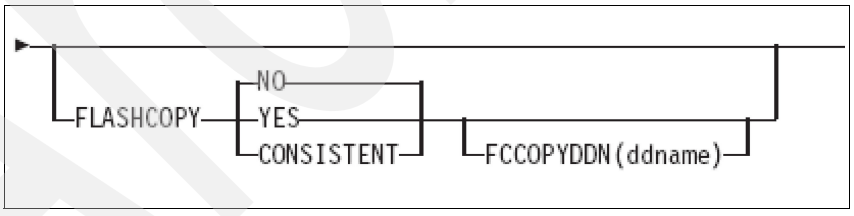


Figure 11-1 FCCOPY keyword on COPY syntax

If you specify FLASHCOPY YES or FLASHCOPY CONSISTENT on the utility control statement or if FlashCopy is enabled for the copy utility through DSNZPARM FLASHCOPY_COPY set to YES, and if you omit the FCCOPYDDN keyword, DB2 uses the template that you specified in DSNZPARM.

Example 11-1 shows JCL and the utility control statement that makes use of the FlashCopy function.

Example 11-1 SAMPLE COPY FLASHCOPY YES

```
//UTIL EXEC DSNUPROC,SYSTEM=DBOB,UID='TEMP',UTPROC=''
```

```
//DSNUPROC.SYSIN    DD  *
  COPY TABLESPACE DB1.TS1 FLASHCOPY YES
```

Example 11-2 shows the job output for this image copy job. DFSMSDSS created a one-to-one copy of the VSAM cluster for DB1.TS1. The name of the generated data set is DB0BI.DB1.TS1.N00001.CY1OWMQP.

Example 11-2 Sample COPY FLASHCOPY YES job output

```
DSNU000I    238 20:45:37.21 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   238 20:45:37.25 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    238 20:45:37.25 DSNUGUTC -  COPY TABLESPACE DB1.TS1 FLASHCOPY YES
 DSNU421I    238 20:45:37.41 DSNUGFUM - START OF DFSMS MESSAGES

1PAGE 0001     5695-DF175  DFSMSDSS V1R11.0 DATA SET SERVICES     2010.238 20:45
-ADR030I (SCH)-PRIME(01), DCB VALUES HAVE BEEN MODIFIED FOR SYSPRINT
  COPY DATASET(INCLUDE( -
  DB0BD.DSNDBC.DB1.TS1.I0001.A001 )) -
  RENAMEU( -
  (DB0BD.DSNDBC.DB1.TS1.I0001.A001 , -
  DB0BI.DB1.TS1.N00001.CY1OWMQP )) -
  REPUNC ALLDATA(*) ALLEXCP CANCELERROR SHARE -
  WRITECHECK TOLERATE(ENQF)
 ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'COPY '
 ADR109I (R/I)-RI01 (01), 2010.238 20:45:37 INITIAL SCAN OF USER CONTROL STATEMENTS COMPLETED
 ADR050I (001)-PRIME(01), DFSMSDSS INVOKED VIA APPLICATION INTERFACE
 ADR016I (001)-PRIME(01), RACF LOGGING OPTION IN EFFECT FOR THIS TASK
0ADR006I (001)-STEND(01), 2010.238 20:45:37 EXECUTION BEGINS
0ADR711I (001)-NEWDS(01), DATA SET DB0BD.DSNDBC.DB1.TS1.I0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
                          DB0BI.DB1.TS1.N00001.CY1OWMQP USING STORCLAS DB0BDATA, DATACLAS DB0B, AND
MGMTCLAS MCDB22
0ADR806I (001)-TOMI (03), DATA SET DB0BD.DSNDBC.DB1.TS1.I0001.A001 COPIED USING A FAST REPLICATION FUNCTION
0ADR801I (001)-DDDS (01), DATA SET FILTERING IS COMPLETE. 1 OF 1 DATA SETS WERE SELECTED: 0 FAILED
SERIALIZATION AND 0
                          FAILED FOR OTHER REASONS
0ADR454I (001)-DDDS (01), THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
0                          DB0BD.DSNDBC.DB1.TS1.I0001.A001
0ADR006I (001)-STEND(02), 2010.238 20:45:37 EXECUTION ENDS
0ADR013I (001)-CLTSK(01), 2010.238 20:45:37 TASK COMPLETED WITH RETURN CODE 0000
0ADR012I (SCH)-DSSU (01), 2010.238 20:45:37 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN CODE IS 0000

 DSNU422I    238 20:45:37.87 DSNUGFCD - END OF DFSMS MESSAGE
 DSNU010I    238 20:45:37.88 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

As mentioned previously, the template that you specify in DSNZPARM FCCOPYDDN is not checked when the DSNZPARM module is created. Therefore, if you specify an invalid name, the utility job fails with return code 8 and an error message similar to the one shown in Example 11-3.

Example 11-3 Invalid name error message

```
DSNU050I    239 13:53:49.00 DSNUGUTC -  COPY TABLESPACE DB1.TS1 FLASHCOPY YES
DSNU1032I   239 13:53:49.00 DSNUGTDS - INVALID QUALIFIER 1. IN TEMPLATE SYSZPFC
DSNU012I    239 13:53:49.01 DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST
RETETURN CODE 8
```

In our scenario, we used the following template:

```
FCCOPYDDN=DB0BI.1..&&SN..N&&DSNUM..&&UQ.,
```

The second qualifier is the number one (1), which makes the name an invalid data set name.

Creating FlashCopy and sequential image copies

If you want to create a sequential image copy in addition to a FlashCopy image copy, you can use one of the following methods:

- Specify the COPYDDN or FCCOPYDDN keyword on your COPY control statement.
- Use the COPYTOCOPY utility afterwards to create those from the FlashCopy image copy.

Before we discuss how to create both FlashCopy and sequential image copies during one utility execution, let us review the simplest cases (that is creating image copies without specifying COPYDDN). Figure 11-2 illustrates two of these simple cases.

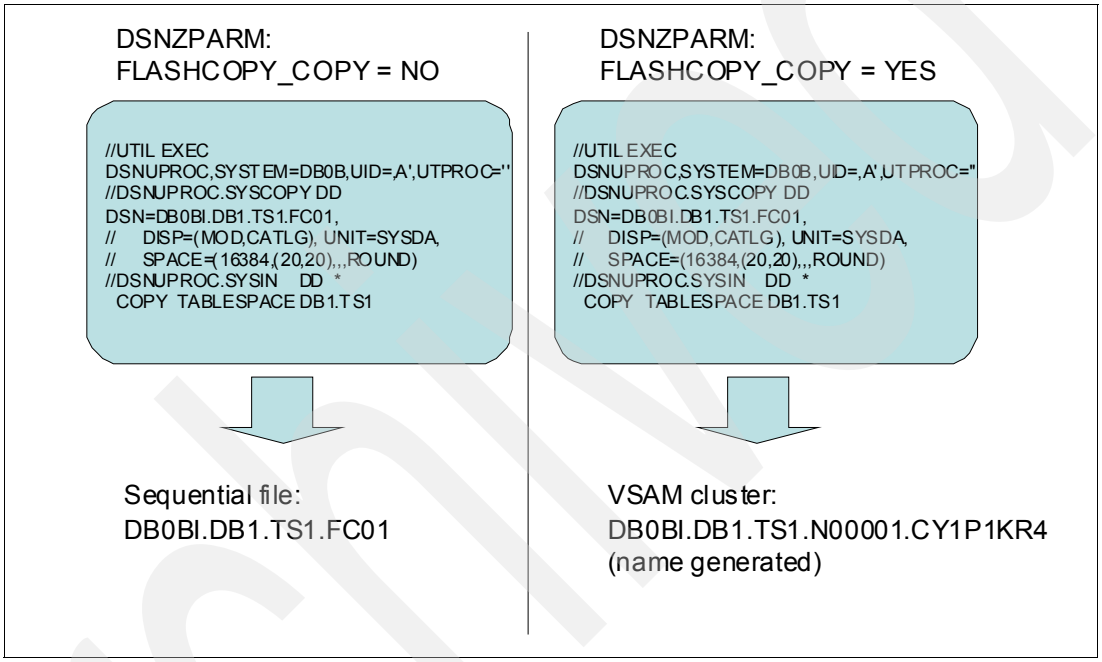


Figure 11-2 FLASHCOPY_COPY=YES versus FLASHCOPY_COPY=NO in DSNZPARM

Figure 11-2 shows the JCL and utility control statement. On the left side of the figure, the DSNZPARM FLASHCOPY_COPY statement is set to NO. Thus, DB2 does not consider FlashCopy image copies unless you explicitly request it on the utility control statement. The utility control statement on the left side does not request a FlashCopy. DB2 creates only a sequential copy using the data set name and allocation information as specified in the SYSCOPY DD statement.

On the right side of the figure, the DSNZPARM FLASHCOPY_COPY statement is set to YES. With this setting, DB2 creates a FlashCopy image copy if you specify the control statement shown on the left side. The DD specification for SYSCOPY is now ignored.

Using COPYDDN to create sequential image copies from FlashCopy image copies

If you want to create both a FlashCopy image copy and additionally a sequential image copy, you must use the JCL and utility control statement shown in Figure 11-3.

On the left hand side, we still use DSNZPARM FLASHCOPY_COPY=NO, that is flash copies are not created by default. As a result you have to specify the FLASHCOPY YES keyword on your utility control statement. It is not necessary to also specify COPYDDN to complete the

image copy with RC 0, because now DB2 only creates a FlashCopy image copy. If, however, you want to get both, FlashCopy image copy and a sequential copy, you have to add the COPYDDN keyword to the utility control statement and must make sure that a valid DD card for SYSCOPY is part of the JCL also.

On the right hand side, we assume that this DB2 subsystem operates with DSNZPARM FLASHCOPY_COPY set to YES. As a result, you can now omit the FLASHCOPY keyword on the utility control statement. DB2 knows that you want FlashCopy image copies from the DSNZPARM setting. If you omit the COPYDDN keyword, DB2 only creates the FlashCopy image copy. If you specify the COPYDDN keyword as shown in Figure 11-3, you must make sure that a valid DD card for SYSCOPY is also part of your JCL. If this is the case, your utility execution creates two copies of your table space, the FlashCopy image copy and the sequential copy named as specified in your DD statement. If you fail to specify a valid DD card, DB2 only creates a FlashCopy image copy but abends when it comes to the creation of the sequential copy.

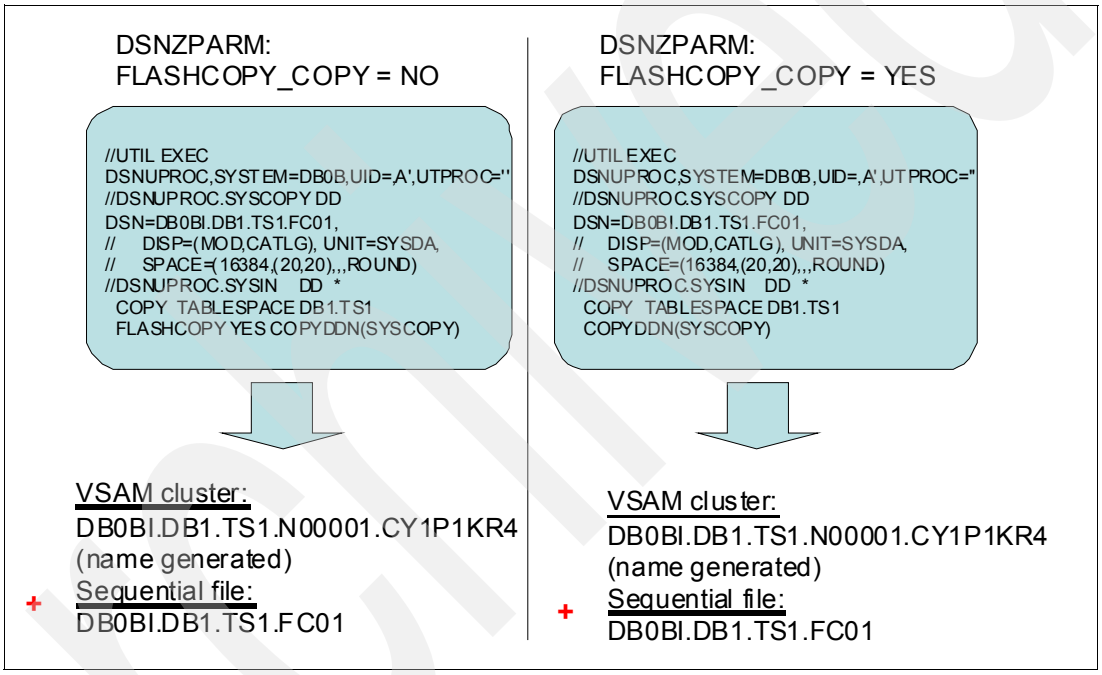


Figure 11-3 Create FlashCopy image copy and sequential copy depending on ZPARM setting

Note: If you specify FLASHCOPY YES and want to create a sequential image copy also, you have to specify the COPYDDN keyword. It is not sufficient to only add a DD statement with the default SYSCOPY DD name to the JCL.

Table 11-1 shows you the combination of a few cases, that is different DSNZPARM settings combined with utility control statements and the resulting image copies.

Table 11-1 DSNZPARM, FLASHCOPY keyword, resulting copy

Utility control statement keyword	FLASHCOPY_COPY=NO	FLASHCOPY_COPY=YES
FLASHCOPY NO	Sequential copy created	Sequential copy created
FLASHCOPY YES	FlashCopy image copy created	FlashCopy image copy created

Utility control statement keyword	FLASHCOPY_COPY=NO	FLASHCOPY_COPY=YES
FLASHCOPY NO COPYDDN(...)	Sequential copy created	Sequential copy created
FLASHCOPY YES COPYDDN(....)	FlashCopy image copy + sequential copy created	FlashCopy image copy + sequential copy created
FLASHCOPY keyword omitted	Sequential copy created	FlashCopy image copy created
FLASHCOPY keyword omitted COPYDDN (...)	Sequential copy created	FlashCopy image copy + sequential copy created
FLASHCOPY YES JCL has no DD card	FlashCopy image copy created	FlashCopy image copy created
FLASHCOPY keyword omitted COPYDDN (....) JCL has no DD car	Utility fails, no copy created	FlashCopy image copy created and then utility execution fails

Attention: Table 11-1 lists a few possible combinations of utility keywords, DSNZPARM settings, and JCL statements specified. The list of possible combinations is not complete by far but it gives you an idea of what happens when you specify or omit one or the other component.

As a general rule, you can say that, whenever you expect DB2 to create a sequential copy, you must specify a valid DD card on your utility JCL or specify the COPYDDN keyword.

In addition, whenever DB2 creates a FlashCopy image copy and you additionally want a sequential copy, you must specify the COPYDDN keyword on your utility control statement.

If you specify both, FLASHCOPY YES and COPYDDN or RECOVERYDDN, DB2 creates the sequential copy from the FlashCopy. The creation of the FlashCopy is fast. Even if you run your FlashCopy with SHRLEVEL REFERENCE, the page set being copied is unavailable for read and write access only for a short period of time. So if still rely on additional sequential image copies, the time it takes to create them does not cause unavailability of data for the applications.

Use COPYTOCOPY with FlashCopy image copy input

The COPYTOCOPY utility has been enhanced to accept FlashCopy image copies as input for the creation of sequential image copies. As was possible before, the output can create up to four image copies: local primary, local backup, recovery site primary, and recovery site backup. Example 11-4 shows a sample extract of the messages generated for a COPYTOCOPY utility with FlashCopy image copy input copy.

Example 11-4 COPYTOCOPY messages for FlashCopy image copy input

```
33.65 DSNUGUTC -  COPYTOCOPY TABLESPACE SEGDB.SEGTS COPYDDN(SYSCOPY)
33.70 DSNU2BCC - LOCAL SITE FLASH COPY DATA SET DBOBI.SEGDB.SEGTS.N00001.C0H41V6A
with
 START_RBA 00003B1E5E13 IS IN USE BY COPYTOCOPY
 FOR TABLESPACE SEGDB.SEGTS
33.72 DSNU2BDR - COPYTOCOPY PROCESSING COMPLETED FOR
 TABLESPACE SEGDB.SEGTS
 ELAPSED TIME = 00:00:00
 NUMBER OF PAGES COPIED=3
```

```
33.72 DSNU2BDR - COPYTOCOPY COMPLETED. ELAPSED TIME = 00:00:00
33.73 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Create FlashCopy image copies of your partitioned table space

For partitioned table spaces you can also create FlashCopy image copies. Because FlashCopy image copies are at page set level, even if you ask for an image copy of the whole partitioned table space, DB2 creates one flash copy data set for each partition. As shown in Figure 11-4, the additional individual sequential copy is generated from the five FlashCopy image copies.

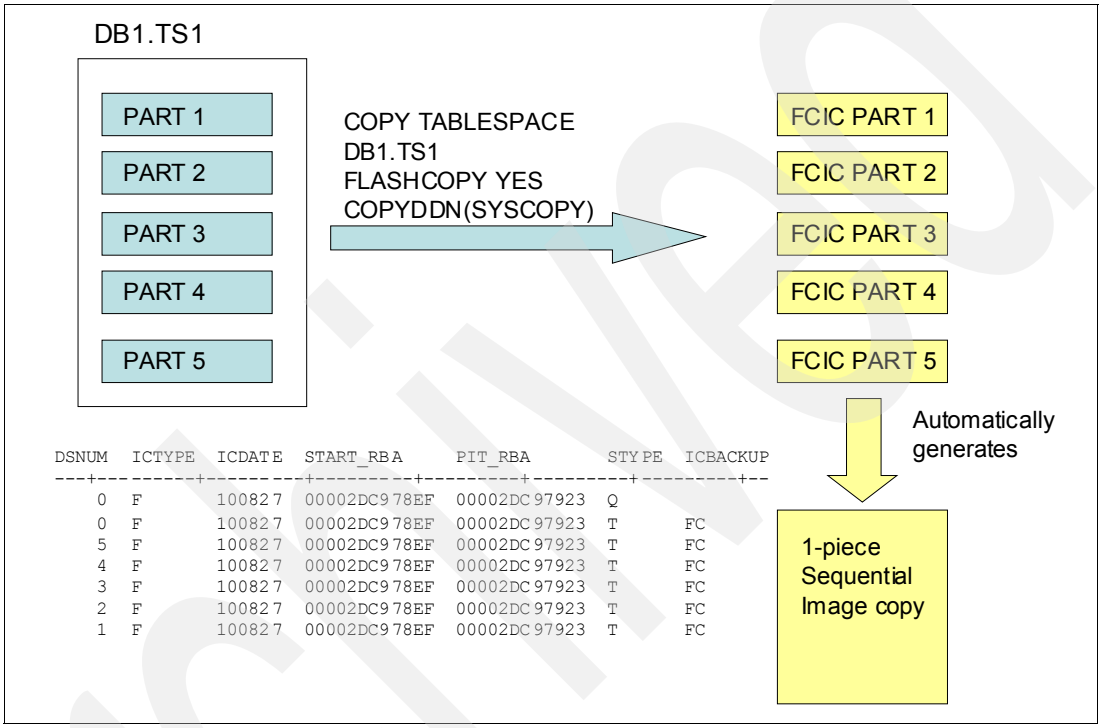


Figure 11-4 FlashCopy image copy of partitioned table space plus sequential copy

We extracted the table shown in Figure 11-4 from SYSIBM.SYSCOPY and ordered it by TIMESTAMP descending.

In the table of the extracted SYSIBM.SYSCOPY entries, the first row is the sequential copy, the second row is generated so that DB2 knows that there is a complete flash copy of your partitioned table space available. If you do a flash copy at partition level, this entry is not present.

Recovering partitioned table spaces using FlashCopy image copies

As shown in Figure 11-4, the result of the execution of just one COPY utility for your partitioned table space consists on five separate FlashCopy image copies. If you need to recover separate partitions from the generated FlashCopy image copies, you can do so by specifying the DSNUM parameter on your RECOVER control statement. For example, to recover just partition 3 to this image copy, specify:

```
RECOVER TABLESPACE DB1.TS1 DSNUM 3 - this would end in a recovery to current
```

If you want to recover just partition 3 to a prior image copy, specify the DSNUM keyword with a target RBA or copy data set name.

Archived

If you want to recover the entire table space to the copy in Figure 11-4, you must distinguish between specifying an RBA and a copy data set name. If you use the TORBA or TOLRSN keyword, you can execute just one RECOVER TABLESPACE utility control statement in which you omit the DSNUM keyword and specify the PIT_RBA for this image copy set from SYSIBM.SYSCOPY.

Note: As shown in Figure 11-4, the START_RBA and PIT_RBA is the same for all partitions of your partitioned table space. The data set names for the copies of each of the partitions are different from each other.

Use the following RECOVER statement for the entire table space to the image copy that you took for it:

```
RECOVER TABLESPACE DB1.TS1 TORBA x'00002DC97923'
```

Example 11-5 shows an extract of the messages that the RECOVER utility generated when we recovered a partitioned table space with three partitions using the TORBA option described previously.

Example 11-5 Messages from RECOVER utility

```
2010.273 07:05:26 EXECUTION BEGINS
DATA SET DBOBI.IODDB01.IODTS4.N00001.COMYCRCQ PREALLOCATED WITH NEW NAME
DBOBD.DSNDBC.IODDB01.IODTS4.I0001.A001, IN CATALOG UCAT.DBOBDATA, ON VOLUME(S): SBOXJ4
DATA SET DBOBI.IODDB01.IODTS4.N00001.COMYCRCQ COPIED USING A FAST REPLICATION FUNCTION
DATA SET DBOBI.IODDB01.IODTS4.N00002.COMYCRDA PREALLOCATED WITH NEW NAME
DBOBD.DSNDBC.IODDB01.IODTS4.I0001.A002, IN CATALOG UCAT.DBOBDATA, ON VOLUME(S): SBOXJ4
DATA SET DBOBI.IODDB01.IODTS4.N00002.COMYCRDA COPIED USING A FAST REPLICATION FUNCTION
DATA SET DBOBI.IODDB01.IODTS4.N00003.COMYCRDB PREALLOCATED WITH NEW NAME
DBOBD.DSNDBC.IODDB01.IODTS4.I0001.A003, IN CATALOG UCAT.DBOBDATA, ON VOLUME(S): SBOXJ4
DATA SET DBOBI.IODDB01.IODTS4.N00003.COMYCRDB COPIED USING A FAST REPLICATION FUNCTION
DATA SET FILTERING IS COMPLETE. 3 OF 3 DATA SETS WERE SELECTED: 0 FAILED SERIALIZATION AND
0
FAILED FOR OTHER REASONS
THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
 DBOBI.IODDB01.IODTS4.N00001.COMYCRCQ
 DBOBI.IODDB01.IODTS4.N00002.COMYCRDA
 DBOBI.IODDB01.IODTS4.N00003.COMYCRDB
2010.273 07:05:27 EXECUTION ENDS
2010.273 07:05:27 TASK COMPLETED WITH RETURN CODE 0000
2010.273 07:05:27 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN CODE IS 0000
```

As shown in Example 11-5, DB2 automatically searches for the correct data sets that contains the copies for each of the different partitions and restores one at a time. However, If you want to recover the entire table space to a specific FlashCopy image copy (TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY are specified), you must specify the data set number on the DSNUM parameter in the RECOVERY control statement for each partition or piece that is recovered *and* pick the correct DSNAME from SYSIBM.SYSCOPY.

Important: If the FlashCopy image copy is migrated or deleted, then recovery proceeds from the sequential image copies if available.

Incremental image copies

You cannot create incremental FlashCopy image copies. A FlashCopy image copy is always a one-to-one full flash copy of the VSAM cluster holding your operational data. In addition, you also cannot create incremental sequential image copies after you run the copy utility with

FLASHCOPY YES. This situation is also the case if you take a FlashCopy image copy and a sequential image copy as part of the latest utility execution.

The “dirty bits” on the underlying operational page sets are not turned off during a FlashCopy image copy. The creation of the sequential image copy (if created during the same utility execution as used for the FlashCopy image copy) also does not reset any dirty bits. The reason for that is that the sequential copy is created from the FlashCopy image copy and does not at all touch the operational VSAM cluster.

As a result, if you run a COPY utility to create a sequential image copy any time you created a FlashCopy image copy and if you specify FULL NO on the utility control statement, DB2 ignores this option and acts as though you specified FULL YES. That is, it creates a full image copy instead. This utility execution now resets the “dirty bits” and subsequent utility executions with FULL NO create incremental image copies again. However, whenever you create a FlashCopy image copy of your page set, the same issue occurs again and DB2 again ignores the FULL YES keyword on the COPY utility control statement.

Generate consistent image copies without disruption

To this point, we have discussed two possible settings (YES and NO) for keyword FLASHCOPY on the COPY utility syntax. Both options can be used with both available SHRLEVELS, that is REFERENCE and CHANGE. One other interesting option, which you can use only with SHRLEVEL CHANGE, is FLASHCOPY CONSISTENT.

When you produce an image copy using these keywords, DB2 generates a potentially inconsistent FlashCopy image copy first and then uses backout processing to make the image copy consistent.

Figure 11-5 illustrates this behavior.

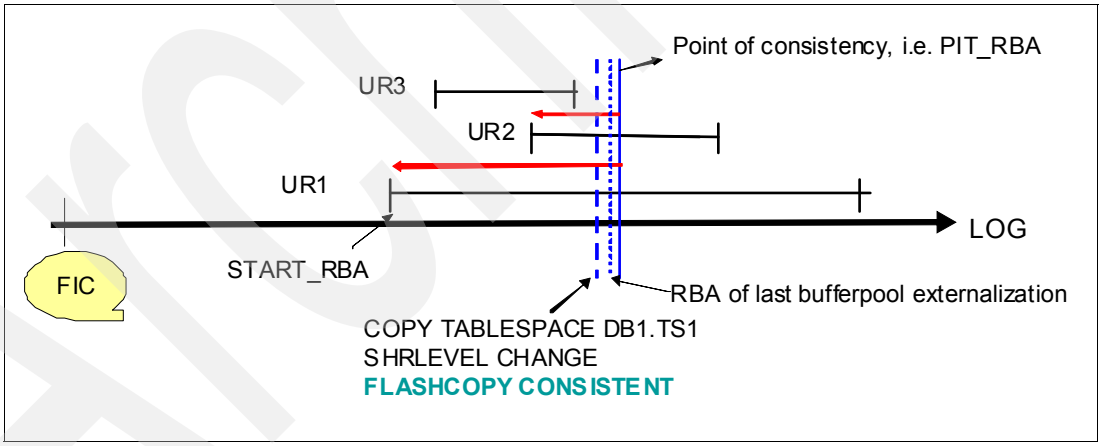


Figure 11-5 FLASHCOPY CONSISTENT

As shown in this example, there are three URs that are currently associated with table space DB1.TS1. When you issue the COPY TABLESPACE statement (left-most vertical dashed line), UR3 is already committed, and UR1 and UR2 is in in-flight status. The vertical line in the middle (dotted) represents the RBA at which all changed pages for the object being copied were last externalized to disk. The right-most solid line is the point of consistency, that is the point to which all URs that were active at this time need to be rolled back. The RBA identified here is used as PIT_RBA in SYSIBM.SYSCOPY for this FlashCopy image copy entry. UR1 needs to be backed out the furthest. The RBA of the first update in this unit of recovery is START_RBA that DB2 records in SYSIBM.SYSCOPY.

Thus, with a subsequent RECOVER using this FlashCopy image copy as recovery base, DB2 needs to read the log beginning at this START_RBA. If DB2 reads only the log beginning at PIT_RBA and UR1 is committed later, everything between the first update and the PIT_RBA would be missing.

As a starting point, DB2 creates a FlashCopy image copy that contains all the uncommitted data for UR1 and UR2. Then, it identifies the point on the log that is considered the point of consistency. Therefore, this point becomes the PIT_RBA. Starting from this PIT_RBA, DB2 backs out all changes done by UR1 and UR2 up to this point on the FlashCopy image copy.

Important: DB2 does this backout only on the FlashCopy image copy. It does not touch the active URs 1 and 2 while doing the back out. UR1 and UR2 can run to completion as intended when starting them.

There is no work to do for UR3, because it completed prior to the image copy and does not reach beyond the PIT_RBA.

After the back out process is complete, the FlashCopy image copy that was created does not cause interruption to the operational DB2 subsystem and is consistent to the point of consistency.

Looking at a real sample

Thus far, we have described how FLASHCOPY CONSISTENT works from a theoretical point of view. Let us now have a look at real job output that illustrates this discussion.

Although a UR is currently uncommitted on our table space DB1.TS1, we issue the copy utility and use the following utility control statement:

```
COPY TABLESPACE DB1.TS1 SHRLEVEL CHANGE FLASHCOPY CONSISTENT
```

Important: Remember that FLASHCOPY CONSISTENT works and makes sense for only SHRLEVEL CHANGE image copies. If you submit a COPY utility with SHRLEVEL REFERENCE, the utility waits for *x* minutes (the value of your utility wait time) to gain exclusive access to the page set that is supposed to be copied. If the utility can get exclusive access, CONSISTENT does not make sense, because the copy is always consistent, and if it does not get exclusive access, the utility fails.

Let us now examine the job output shown in chunks in the following few examples.

Example 11-6 shows the first part of the utility messages for the job statement stated previously. DB2 first creates a SHRLEVEL CHANGE FlashCopy image copy.

Example 11-6 First part of FLASHCOPY CONSISTENT

```
DFSMSDSS INVOKED VIA APPLICATION INTERFACE
RACF LOGGING OPTION IN EFFECT FOR THIS TASK
2010.270 12:03:20 EXECUTION BEGINS
DATA SET DBOBD.DSNDBC.DB1.TS1.I0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
DBOBI.DB1.TS1.N00001.C0IIRMWJ USING STORCLAS DBOBDATA, DATACLAS DBOB, AND MGMTCLASS MCDB22
DATA SET DBOBD.DSNDBC.DB1.TS1.I0001.A001 COPIED USING A FAST REPLICATION FUNCTIO
DATA SET FILTERING IS COMPLETE. 1 OF 1 DATA SETS WERE SELECTED: 0 FAILED SERIALIZATION AND
0 FAILED FOR OTHER REASONS
THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
 DBOBD.DSNDBC.DB1.TS1.I0001.A001
2010.270 12:03:21 EXECUTION ENDS
2010.270 12:03:21 TASK COMPLETED WITH RETURN CODE 0000
```

```
2010.270 12:03:21 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN CODE IS 0000
```

After this statement completes, DB2 identifies the checkpoint that is the closest checkpoint prior to the PIT_RBA. It uses this checkpoint to identify the status of open URs. As a result, it reports the UR counts in the job output as shown in Example 11-7.

Example 11-7 Messages for FlashCopy image copy consistency processing

```
DSNU1540I   273 11:44:45.58 DSNUGFCD - CONSISTENCY PROCESSING FOR FLASHCOPY IS
DSNU513I  -DB0B 273 11:44:45.61 DSNUCALZ - FLASHCOPY WITH CONSISTENCY LOG APPLY RANGE IS
RBA 00003D4D9377 LRSN 00003D4D9377 TO RBA 00003D507EEF LRSN 00003D507EEF
DSNU1511I -DB0B 273 11:44:45.62 DSNUCALZ - FAST LOG APPLY WAS NOT USED FOR FLASHCOPY WITH
CONSISTENCY
DSNU1510I   273 11:44:45.62 DSNUCBLA - LOG APPLY PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU1550I -DB0B 273 11:44:45.62 DSNUCALC - LOGCSR IS STARTED FOR MEMBER , PRIOR CHECKPOINT
RBA = X'00003D4D9377'
DSNU1551I -DB0B 273 11:44:45.62 DSNUCALC - LOGCSR IS FINISHED FOR MEMBER , ELAPSED TIME =
00:00:00
DSNU1552I -DB0B 273 11:44:45.62 DSNUCALC - LOGCSR PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU1553I -DB0B 273 11:44:45.62 DSNUCALC - FLASHCOPY WITH CONSISTENCY DETECTS THE FOLLOWING
ACTIVE URS:
             INFLIGHT = 1, INABORT = 0, INDOUBT = 0, POSTPONED ABORT = 0 COMMITTED = 0,
ABORTED = 0

               MEM    T  CONNID     CORRID     AUTHID    PLAN   S     URID DATE       TIME
                      B TSO      DB2R8        DB2R8    DSNESPCS F 00003D4FE98D 2010-09-30
15.44.35
                   DBNAME  SPACENAME DBID/PSID PART     RBA
                  DB1      TS1       011E/0002 0001 00003D4FEA4C
DSNU1554I -DB0B 273 11:44:45.63 DSNUCALU - LOGUNDO IS STARTED FOR MEMBER
DSNU1556I -DB0B 273 11:44:45.63 DSNUCALU - LOGUNDO IS FINISHED FOR MEMBER = 00:00:00
DSNU1557I -DB0B 273 11:44:45.63 DSNUCALU - LOGUNDO PHASE COMPLETE, ELAPSED TIME , =
00:00:00
DSNU1541I   273 11:44:45.64 DSNUGFCD - CONSISTENCY PROCESSING FOR FLASHCOPY IS FINISHED,
ELAPSED TIME = 00:00:00
DSNU010I    273 11:44:45.64 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE IS 0
```

Message DSNU1553I identifies the URs that need to be backed out. In our case, the URID is 00003D4FE98D. The information about the first update in this UR is located at RBA 00003D4FEA4C (refer to column RBA). This RBA is now used as START_RBA in the resulting SYSIBM.SYSCOPY record.

Using information in SYSIBM.SYSCOPY

You can identify FlashCopy image copies in SYSIBM.SYSCOPY looking at columns TYPE and ICTYPE. TYPE shows a value of F and ICTYPE a value of FC. You need to know this information, for example, if you plan to have a mixture between FlashCopy image copies and sequential copies with incremental image copies.

The STYPE column in SYSIBM.SYSCOPY has a new type of value that is related to FlashCopy image copies. Remember that the following information exists for each piece or part in the DB2 catalog:

STYPE = T	FlashCopy copy is consistent
STYPE = N	FlashCopy copy is not consistent
STYPE = Q	Sequential copy is consistent
STYPE = U	Sequential copy is not consistent
TTYPE =	One character indicating the type of utility which made the copy

The SYSIBM.SYSCOPY record for a RECOVER utility execution with the BACKOUT YES keyword specified has an ICTYPE= P and a STYPE=B.

11.1.2 FlashCopy image copies and utilities other than COPY

The following DB2 for z/OS utilities can use FlashCopy to create image copies:

- The COPY utility
- The LOAD utility
- The REBUILD INDEX utility
- The REORG INDEX utility
- The REORG TABLESPACE utility

If you use FlashCopy image copies with the COPY and LOAD utilities, it can include the following additional phases of execution depending on the options you specify in the utility control statement:

LOGAPPLY	If you specify CONSISTENT for either the COPY or LOAD utility, the utility identifies the most recent checkpoint for each member. All objects that are copied are updated to the same log point to prepare for backout processing.
LOGCSR	If you specify CONSISTENT for either the COPY or LOAD utility, the utility reads the logs during this phase to identify the uncommitted work that needs to be backed out of the image copy.
LOGUNDO	If you specify CONSISTENT for either the COPY or LOAD utility, the utility backs out uncommitted work to make the image copy consistent.
SEQCOPY	If additional sequential format copies are requested, the COPY utility creates them from the FlashCopy image copy during this phase.

The following utilities accept the VSAM data sets produced by FlashCopy as input:

- COPYTOCOPY
- DSN1COMP
- DSN1COPY
- DSN1PRNT
- RECOVER

The MERGECOPY utility: MERGECOPY is not in this list. As explained earlier, incremental copies cannot be made using the FlashCopy functions.

In addition to DSNZPARM FLASHCOPY_COPY, and FCCOPYDDN, the following FlashCopy are related to DSNZPARMs:

- FLASHCOPY_LOAD
- FLASHCOPY_REORG_TS
- FLASHCOPY_REBUILD_INDEX
- FLASHCOPY_REORG_INDEX

The default setting for all these FLASHCOPY DSNZPARMs is NO. This is also true for FLASHCOPY_COPY.

We cover specifics for each of the utilities in the sections that follow.

FlashCopy image copies and LOAD

The behavior regarding FlashCopy in conjunction with the LOAD utility is almost identical to the behavior that we described previously for the COPY utility. You can set DSNZPARM

FLASHCOPY_LOAD to YES or NO. The CONSISTENT option for FlashCopy is not a valid setting for the DSNZPARM. If you use the default setting of NO, DB2 does not consider producing a FLASHCOPY without explicitly requesting it on the utility control statement. If you change the DSNZPARM to YES and if you use the following utility control statement, DB2 creates a FLASHCOPY automatically as part of the LOAD utility execution:

```
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC  CCSID(00037,00000,00000)
 INTO TABLE
 "DB2R8".
 "TB1" .......
```

The creation of the FlashCopy image copy starts as soon as the RELOAD or BUILD phase completes, depending on whether you have created an index or indexes.

The same situation occurs if you leave DSNZPARM FLASHCOPY_LOAD with a setting of NO and specify FLASHCOPY YES or CONSISTENT on the utility control statement. Whether you run your LOAD utility with the default setting of SHRLEVEL REFERENCE using FLASHCOPY YES or CONSISTENT really does not make any difference. The resulting FlashCopy image copy is consistent by default.

If you want to produce sequential copies and FlashCopy image copies, specify the COPYDDN or RECOVERYDDN keyword.

Attention: COPYDDN and RECOVERYDDN are only valid for LOAD ... REPLACE. If you specify both, FLASHCOPY and COPYDDN, the sequential copy is made during the reload phase of your LOAD utility. After RELOAD (and optionally BUILD) completes, DB2 takes an additional FlashCopy.

If you request only the creation of a FlashCopy and do not specify COPYDDN or RECOVERYDDN, DB2 creates an image copy not only if you run your LOAD utility with keyword REPLACE but also with RESUME YES. With DB2 10, if you use FlashCopy image copies, DB2 lets you create “inline” image copies with both the RESUME YES and REPLACE options.

LOAD with SHRLEVEL CHANGE and FLASHCOPY CONSISTENT

Up to DB2 9, when you used the LOAD utility with SHRLEVEL CHANGE and RESUME YES, you could not create inline image copies, because they are allowed only for LOAD utilities with the REPLACE option specified. Alternatively, SHRLEVEL CHANGE is not allowed for LOAD REPLACE operations.

With DB2 10, it is possible to create image copies for SHRLEVEL CHANGE LOAD operations. You cannot create sequential image copies, but you can produce the FlashCopy image copies with the execution of LOAD. As for the COPY utility SHRLEVEL CHANGE option, you can use FLASHCOPY YES and FLASHCOPY CONSISTENT, which gives you the flexibility of creating potentially inconsistent or consistent image copies with the LOAD utility. These image copies are created using the same method that we described for COPY in “Generate consistent image copies without disruption” on page 434.

FlashCopy image copies and REORG TABLESPACE

As with the LOAD utility, you can create FlashCopy image copies as part of the REORG TABLESPACE execution. As part of the utility control statement, you can specify FLASHCOPY NO or YES or CONSISTENT. Following the nature of a REORG TABLESPACE the utility, the result of it is always consistent, which mean that specifying YES or CONSISTENT does not lead to any different results.

As discussed with COPY and LOAD previously, you can set the DSNZPARM FLASHCOPY_REORG_TS option to YES or NO. NO is the default. If you use the default, you must explicitly code the FLASHCOPY YES or FLASHCOPY CONSISTENT statement on the utility control statement if you want to produce FlashCopy image copies automatically after the reorganization of a table space. If you use YES, DB2 always creates a FlashCopy image copy automatically as part of the REORG TABLESPACE execution.

However, although this all sounds quite straightforward and similar to what we discussed previously, there are a few differences in the use of FlashCopy image copies in conjunction with REORG TABLESPACE, which we discuss in the following sections.

FlashCopy image copies and REORG TABLESPACE SHRLEVEL CHANGE or REFERENCE

Independent from talking about FlashCopy image copies, when you use SHRLEVEL CHANGE or REFERENCE on your REORG TABLESPACE control statement, DB2 requires you to specify a SYSCOPY DD name for a sequential copy in the utility JCL. Thus, taking an image copy is mandatory with these SHRLEVEL settings. Because it is mandatory, you do not need to specify the COPYDDN keyword on the utility control statement. By default, DB2 assumes COPYDDN(SYSCOPY) is specified and, therefore, always searches for a SYSCOPY DD statement in the JCL.

This assumption on COPYDDN(SYSCOPY) does not change with the introduction of FlashCopy image copies. It is independent from the DSNZPARM FLASHCOPY_REORG_TS setting, and it is also not dependent on whether you specify the FLASHCOPY keyword on the utility control statement. REORG TABLESPACE does not create a FlashCopy image copy with no sequential copy.

If you fail to specify a valid SYSCOPY DD name, the utility fails with RC 8 and the message shown in Example 11-8.

Example 11-8 Error message you get when no SYSCOPY DD specified

```
DSNUGUTC -  REORG TABLESPACE DB1.TS1 SHRLEVEL REFERENCE FLASHCOPY CONSISTENT
DSNURORG - A REQUIRED DD CARD OR TEMPLATE IS MISSING. NAME=SYSCOPY
DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

If you specify a valid SYSCOPY DD name, DB2 creates the sequential copy as part of the RELOAD phase and the FlashCopy image copy is created after the REORG TABLESPACE activities complete.

Restriction: If you run REORG TABLESPACE with SHRLEVEL CHANGE or REFERENCE, you must have a sequential copy first. FlashCopy image copies can be taken only in addition to the sequential copy and not instead of it.

FlashCopy image copies and REORG TABLESPACE SHRLEVEL NONE

If you reorganize your table space using SHRLEVEL NONE, DB2 behaves differently from what we just described for SHRLEVEL CHANGE and REFERENCE.

For SHRLEVEL NONE, you do not have to have a valid SYSCOPY DD statement coded in your JCL. With SHRLEVEL NONE on REORG, you are not forced to take an image copy at all. So as a consequence, if you do not specify any of the following options, DB2 does not attempt to create an image copy at all as part of the REORG TABLESPACE execution:

- COPYDDN and valid SYSCOPY DD name
- FLASHCOPY YES

- FLASHCOPY CONSISTENT
- DSNZPARM FLASHCOPY_REORG_TS=YES

Alternatively, if you specify any of these keywords or parameters, DB2 takes one or more image copies as part or REORG TABLESPACE execution.

Here is a list of examples and their results:

- `REORG TABLESPACE DB1.TS1 (DSNZPARM FLASHCOPY_REORG_TS = NO)`

  No image copy is taken.

- `REORG TABLESPACE DB1.TS1 (DSNZPARM FLASHCOPY_REORG_TS = YES)`

  FlashCopy image copy taken.

- `REORG TABLESPACE DB1.TS1 COPYDDN(SYSCOPY)`
  `(DSNZPARM FLASHCOPY_REORG_TS = NO)` and `NO SYSCOPY DD` in JCL

  Utility does not complete successfully (RC 8).

- `REORG TABLESPACE DB1.TS1 COPYDDN(SYSCOPY)`
  `(DSNZPARM FLASHCOPY_REORG_TS = NO)` and `SYSCOPY DD` in JCL

  Sequential image copy taken.

- `REORG TABLESPACE DB1.TS1 COPYDDN(SYSCOPY) FLASHCOPY YES`
  `(DSNZPARM FLASHCOPY_REORG_TS = NO)` and `SYSCOPY DD` in JCL

  Sequential image copy and FlashCopy image copy taken.

FlashCopy image copies and REBUILD INDEX

You can use the REBUILD index utility to repopulate indexes from the currently existing table data. You typically run REBUILD index in the following situations:

- Your index space is in rebuild pending (RBDP) state. The RBDP state is a restrictive state. That is, the index is not available for any SQL processing at this time. RBDP can be set as a result of various actions, such as recovering the underlying table space to a prior point in time.
- Your index space is in advisory rebuild pending state (ARBDP). ARBDP is set when you alter a column that is part of the index from fixed length to variable length.
- Your index space is in page set rebuild pending state (PSRBDP). If LPL recovery of an index fails due to a temporary resource unavailable condition, then the index can be placed in PSRBDP.

Archived

You can reset the RBDP state using various methods, which we do not discuss here. However, one method is to use the REBUILD INDEX utility. Figure 11-6 provides an example.

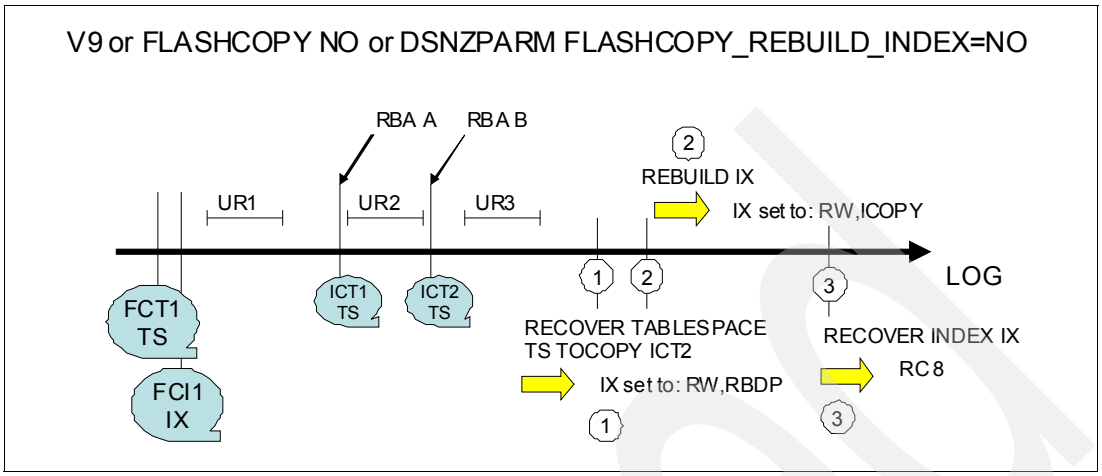


Figure 11-6 REBUILD_INDEX without FLASHCOPY

This scenario assumes that you are operating in either DB2 9 for z/OS or in DB2 10 but that you do use the FlashCopy function. On the left side of the picture, there are image copies FCT1 for the table space and FCI1 for an index that is defined on the table stored in the table space and for which copy is enabled. Changes are made to the table and the index.

Two incremental image copies are created from the table space at RBA A and RBA B. There are no incremental image copies of the index space, because it is not possible to take incremental copies of indexes. UR3 might have done updates to your table data that need to be rolled back.

As a consequence at (1), you recover the table space to incremental image copy ICT2. This point-in-time recovery sets the RBDP state for the index. You can remove the RBDP state from the index by running the REBUILD INDEX utility. Assuming that the REBUILD INDEX utility, at (2), completes successfully, the copy-enabled index is placed in informational copy pending (ICOPY) state. ICOPY does not prevent any SQL statement from invoking the index, because it is only an advisory state. However, ICOPY means that you should take an image copy and that there is currently no valid image copy for that index. Thus, RECOVER INDEX, at (3), does not work.

If you try to run RECOVER INDEX later, the utility will fail with return code 8 and message DSNU522I as shown in Example 11-9.

Example 11-9 DSNU552I trying to recover TS in ICOPY

```
DSNU522I  -DBOB 274 07:56:56.97 DSNUCASA - RECOVER CANNOT PROCEED FOR INDEXSPACE
DB1.TB1RIX
BECAUSE A NON-RECOVERABLE EVENT HAS BEEN
ENCOUNTERED FROM SYSIBM.SYSCOPY WHICH HAS
DBNAME=DB1 TSNAME=TS1 DSNUM=0 ICTYPE=Z START_RBA=X'00003EB02E10'
```

DB2 10 and the FLASHCOPY function improves this situation. Figure 11-7 illustrates the changed scenario. The scenario is the same, except at the point where the REBUILD INDEX utility is executed. This time, FLASHCOPY YES is specified on the utility control statement, which leads to the creation of a FlashCopy image copy as part of the REBUILD INDEX utility. A subsequent RECOVER INDEX utility, at (3), can now complete successfully, because the recovery base after the REBUILD INDEX utility is the FlashCopy image copy that was created.

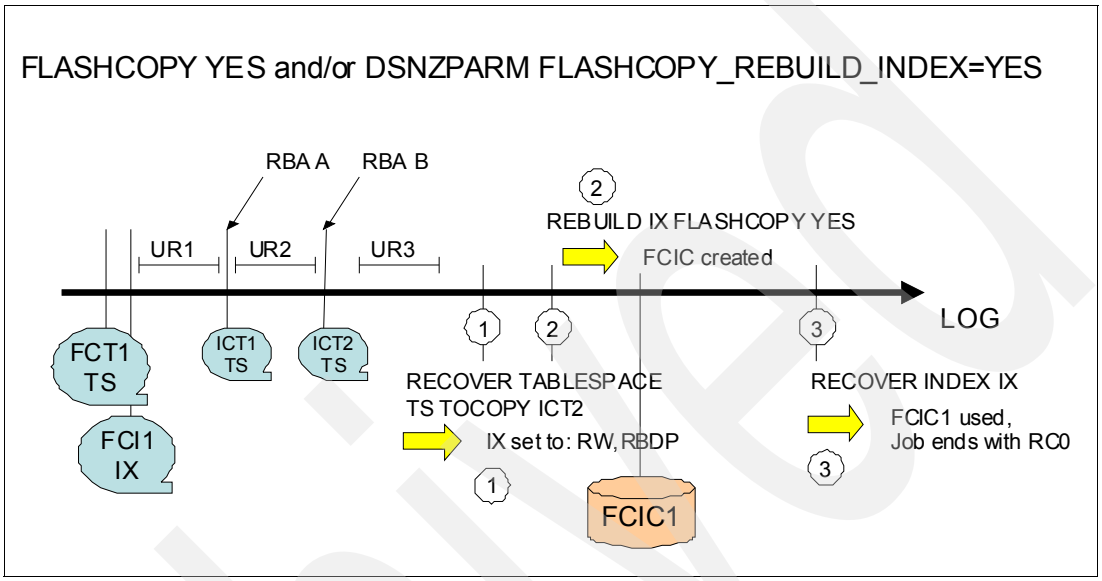


Figure 11-7 REBUILD INDEX with FLASHCOPY

If you run the REBUILD INDEX utility and do not explicitly identify the indexes but instead use the ALL option, as shown, DB2 only takes image copies for the copy-enabled indexes:

```
REBUILD INDEX(ALL) TABLESPACE DB1.TS1 FLASHCOPY YES
```

So, if you have a mixture of copy-enabled and not copy-enabled indexes in one table space, the REBUILD INDEX utility automatically identifies the indexes that are not copy-enabled and lists them in the REBUILD INDEX utility output, as shown in Example 11-10. As shown in bold font in this example, the DB1.TB1RIX index space is copied using FlashCopy. The data set name of the FlashCopy image copy is DB0BI.DB1.TB1RIX.N00001.C0P2XHUP. Further down in the job output, the message DSNU425I indicates that no FlashCopy was produced for the DB1.TB1RIX2 index space because it is not copy enabled.

Example 11-10 REBUILD INDEX FLASHCOPY YES output extract

```
DATA SET DBOBD.DSNDBC.DB1.TB1RIX.I0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
DBOBI.DB1.TB1RIX.N00001.C0P2XHUP USING STORCLAS DBOBDATA, DATACLAS DBOB, AND
MGMTCLAS MCDB22
DATA SET DBOBD.DSNDBC.DB1.TB1RIX.I0001.A001 COPIED USING A FAST REPLICATION
FUNCTION
DATA SET FILTERING IS COMPLETE. 1 OF 1 DATA SETS WERE SELECTED: 0 FAILED
SERIALIZATION AND 0 FAILED FOR OTHER REASONS
THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
 DBOBD.DSNDBC.DB1.TB1RIX.I0001.A001
2010.275 05:14:27 EXECUTION ENDS
2010.275 05:14:27 TASK COMPLETED WITH RETURN CODE 0000
2010.275 05:14:27 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN CODE IS 0000
```

```
76 DSNUGFCD - END OF DFSMS MESSAGE
DSNU425I  -DBOB 275 05:14:27.23 DSNUGFCR - INDEXSPACE DB1.TB1RIX2   DOES NOT HAVE
THE COPY YES ATTRIBUTE
```

FlashCopy image copies and REORG Index

Up to DB2 9, the REORG INDEX utility did not provide a way to create image copies as part of the utility execution. This function is also true if you run REORG INDEX with the SHRLEVEL REFERENCE or SHRLEVEL CHANGE option. This function is different from what we discussed for REORG TABLESPACE. REORG TABLESPACE with SHRLEVEL REFERENCE or SHRLEVEL CHANGE always requires a full image copy to be created as part of the utility run.

However, if you run REORG for copy enabled indexes, the image copies that were taken prior to this REORG are no longer usable for subsequent RECOVER INDEX utilities. This situation is similar to what we described in “FlashCopy image copies and REBUILD INDEX” on page 440.

Figure 11-8 illustrates this situation. RECOVER INDEX as shown in the bottom corner of the figure fails with return code 8. Message DSNU514I indicates that a LOG NO event prior to this RECOVER INDEX utility prevents DB2 from using image copies that were taken prior to this LOG NO event. In our case, the LOG NO event is the REORG INDEX utility.

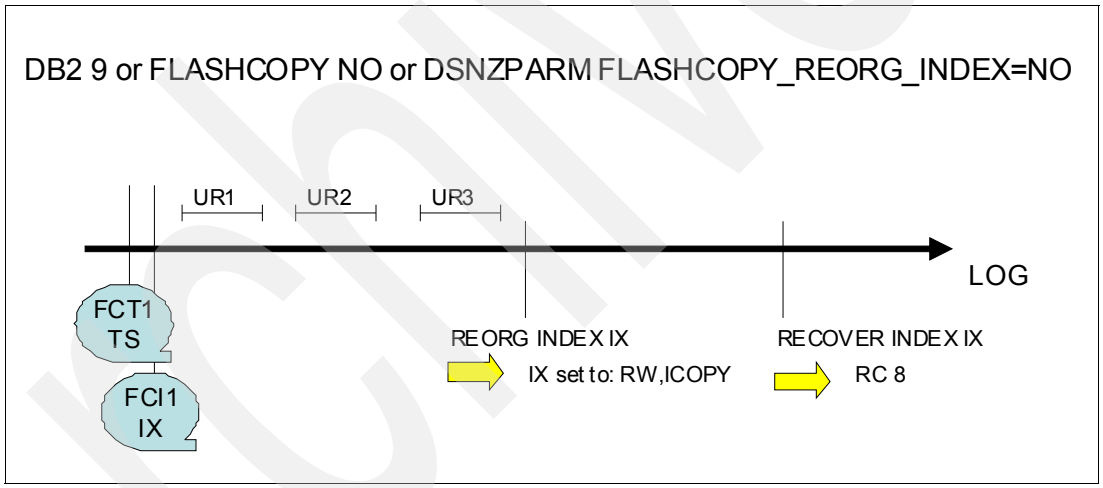


Figure 11-8 REORG INDEX without FLASHCOPY

With DB2 10, you can prevent DB2 from running into this problem by either specifying FLASHCOPY YES on the utility control statement or by setting DSNZPARM FLASHCOPY_REORG_INDEX to YES.

If you run REORG INDEX with FLASHCOPY YES or DSNZPARM FLASHCOPY_REORG_INDEX set to YES and if the index that you reorganize is not copy enabled, the REORG runs to completion with return code 0. Message DSNU425I in the utility output informs you that no image copy is taken for this index.

The following utilities accept the VSAM data sets that are produced by FlashCopy as input:

- COPYTOCOPY
- DSN1COMP
- DSN1COPY
- DSN1PRNT
- RECOVER

When you use a FlashCopy image copy data set as input, do not specify the FULLCOPY option.

The UNLOAD utility does not accept FlashCopy image copies as input. To use a FlashCopy image copy as the source for the UNLOAD utility, use the COPYTOCOPY utility to create sequential format image copies from the FlashCopy image copy.

11.1.3 Requirements and restrictions for using FlashCopy image copies

To use the FlashCopy function requires the system and security restrictions that we discuss in this section.

System requirements

The following system requirements must be met:

- DB2 10 NFM
- FlashCopy Version 2
- All data sets have to be on SMS managed volumes

Operational restrictions

FlashCopy is subject to certain operational restrictions that can cause a utility to either resort to traditional I/O methods to complete a copy operation, even when you have explicitly requested FlashCopy support in either the subsystem parameter or the utility control statement. In some cases, the utility will abort the copy operation altogether.

The circumstances in which the utilities might not be able to complete a copy operation by using FlashCopy include:

- FlashCopy Version 2 disk volumes are not available.
- The source tracks are already the target of a FlashCopy operation.
- The target tracks are the source of a FlashCopy operation.
- The maximum number of relationships between source and target tracks is exceeded for the FlashCopy operation.
- The input data set is less than one cylinder. In this case, if FlashCopy is not used the copy is performed by IDCAMS.

In the event that FlashCopy cannot be used, the utilities complete the copy operation using traditional I/O methods to copy the object, which can result in longer than expected execution time.

Only one utility can create a FlashCopy image copy of an object at a time. For example, if the COPY utility is running on an object with SHRLEVEL CHANGE and FLASHCOPY specified, and the LOAD utility is started on the same object also with SHRLEVEL CHANGE and FLASHCOPY specified, the LOAD utility receives message DSNU180I with a return code of 8 to indicate that the LOAD utility is not compatible with the COPY utility.

The RECOVER PARALLEL and REUSE keywords are ignored for FLASHCOPY. The system can use parallelism anyway. Independent of the REUSE parameter, the system might or might not choose to scratch and reallocate the data sets.

Security considerations regarding FlashCopy

Up to DB2 9, when you execute a DB2 online utility, such as LOAD, RECOVER, and other utilities, access to VSAM clusters is handled in DB2 itself. For example, User A who does not

have any privilege to access the VSAM cluster for table space TS1 directly can recover this page set using the RECOVER utility.

This behavior changes with DB2 10 for z/OS when FLASHCOPY comes into play. If either the FLASHCOPY YES or FLASHCOPY CONSISTENT option is specified, the batch user ID that invokes a utility must have the authority to execute the DFSMSdss COPY command and the required access level authority per data set.

A security administrator or resource owner can control access to resources by creating a discrete profile, which protects one resource, or a generic profile, which protects one or more resources. Each profile has a defined, universal access level and an access list that permits users and group IDs specific levels of access authority.

When profiles control resources, DFSMSdss usage can be restricted. Your installation can put DFSMSdss in a protected library to restrict its use. Your installation can also limit the use of certain DFSMSdss commands, functions, and keywords by defining resource profiles in the FACILITY class and restricting access to those profiles. To use a protected command, function, or keyword, you need READ access authority to the applicable profile. Refer to *DFSMSdss Storage Administration Reference,* SC35-0423*,* for detailed information about resource controls.

Even if DFDSS is not explicitly protected in your environment, it is still most likely that data base administrators might run into the typical error message shown in Example 11-11. We encountered this message when we tried to recover the page set with an FlashCopy image copy as the recovery base.

Example 11-11 RACF error message when trying to involve FlashCopy in recover

```
ICH408I USER(IOD1    ) GROUP(STUDENT ) NAME(DB2 STUDENT         )  058
  DBDSN1.DSNDBC.IODDB01.IODTS3.I0001.A001 CL(DATASET ) VOL(CF01DB)
  INSUFFICIENT ACCESS AUTHORITY
  FROM DBDSN1.** (G)
  ACCESS INTENT(UPDATE )  ACCESS ALLOWED(READ   )
```

11.2 Autonomic statistics

The first step in providing an autonomic way of running utilities is the configuration of a set of administration stored procedures as mentioned in 9.9.3, “DB2 statistics routines” on page 332. After you configure autonomic monitoring, DB2 relies on scheduled calls to the ADMIN_UTL_MONITOR stored procedure to monitor your statistics.

When old, missing, or conflicting statistics are identified, the ADMIN_UTL_EXECUTE stored procedure invokes RUNSTATS within a defined maintenance window, executes it based on a defined profile, and resolves the problems. The ADMIN_UTL_EXECUTE stored procedure uses the options that are defined in RUNSTATS profiles to invoke the RUNSTATS stored procedure. The ADMIN_UTL_MODIFY stored procedure is called at regular intervals to clean up the log file and alert history.

A RUNSTATS profile is a saved set of options for the RUNSTATS utility that apply for a particular table. DB2 uses RUNSTATS profiles for autonomic statistics maintenance. You can also use RUNSTATS profiles to quickly invoke the RUNSTATS utility with a predefined set of options.

You can specify a complete set of RUNSTATS options in a profile, specify only a few options, or even specify only a single option. The options that you specify are stored in the

PROFILE_TEXT column of the SYSIBM.SYSTABLES_PROFILES catalog table. If you do not specify values for the options when you create the profile, DB2 uses default values, as in any RUNSTATS invocation.

RUNSTATS profiles are saved as a single row in the SYSIBM.SYSTABLES_PROFILES catalog table. After your create a profile for a table, you can specify that DB2 uses the same options that were specified in the table when you invoke RUNSTATS again later. When you automate statistics maintenance, DB2 creates or updates the single profile for each table that is not excluded from autonomic maintenance. Because only a single RUNSTATS profile can exist for each table, DB2 uses any options that you have specified in existing profiles for a particular table when the ADMIN_UTL_MONITOR stored procedure first executes for autonomic statistics monitoring.

Regardless of whether profiles exist or whether autonomic statistics maintenance is enabled, you can always invoke the RUNSTATS utility and specify customized options in the traditional manner, without using the profile.

DB2 uses RUNSTATS profiles when you enable autonomic statistics maintenance.

When you first enable autonomic statistics maintenance, the ADMIN_UTL_MONITOR stored procedure sets a profile for each monitored table based on the existing statistics. However, if a profile already exists for a table, DB2 uses that existing profile.

To set a RUNSTATS profile, issue the following utility control statement to explicitly specify the collection options in the profile:

```
RUNSTATS TABLESPACE ts-name TABLE table-name runstats-options SET PROFILE
```

Figure 11-9 shows these options in more details.

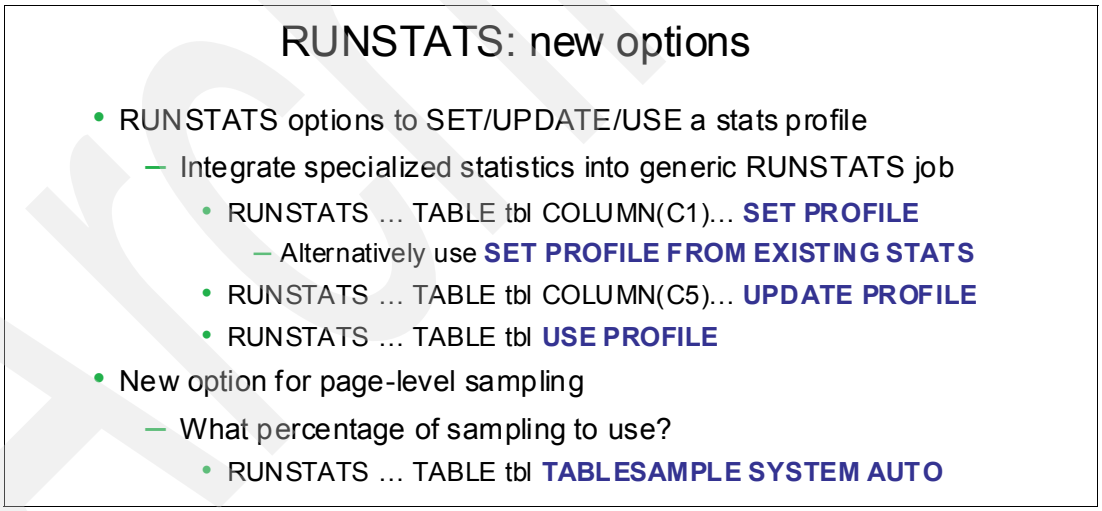


Figure 11-9 Profile definition for RUNSTATS

DB2 records the values specified by runstats-options in the PROFILE_TEXT column of the SYSIBM.SYSTABLES_PROFILES catalog table. DB2 uses the default values for any options that are not specified.

Issue the following utility control statement to automatically specify options in the profile based on the existing statistics for the specified table:

```
RUNSTATS TABLESPACE ts-name TABLE table-name SET PROFILE FROM EXISTING STATS
```

No statistics are collected when you invoke the RUNSTATS utility with the SET PROFILE option. If a profile already exists for the specified table, that profile is replaced with the new one and the existing profile is lost because only one profile can exist for a particular table.

Example 11-12 shows the input and output of a RUNSTATS PROFILE execution.

Example 11-12 RUNSTATS PROFILE utility execution

```
1DSNU000I    302 22:56:00.25 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   302 22:56:00.28 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    302 22:56:00.28 DSNUGUTC -  RUNSTATS TABLESPACE DSN8D10A.DSN8S10D TABLE(DSN81010.DEPT) SET PROFILE FROM
  EXISTING STATS
 DSNU1360I -DBOB 302 22:56:00.31 DSNUGPRF - THE STATS PROFILE FOR TABLE DEPT HAS BEEN DELETED IN SYSTABLES_PROFILES.
 DSNU1357I -DBOB 302 22:56:00.31 DSNUGPRF - THE STATS PROFILE FOR TABLE DEPT HAS BEEN SET IN SYSTABLES_PROFILES.
 DSNU610I  -DBOB 302 22:56:00.31 DSNUGPRF - SYSTABLES_PROFILES CATALOG UPDATE FOR DSN81010.DEPT SUCCESSFUL
 DSNU010I    302 22:56:00.32 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

11.2.1 Using RUNSTATS profiles

You can use RUNSTATS profiles to invoke the RUNSTATS utility with a predefined set of statistics collection options. DB2 also uses RUNSTATS profiles when you enable autonomic statistics maintenance.

To invoke RUNSTATS by using a profile, issue the following RUNSTATS control statement:

```
RUNSTATS TABLESPACE ts-name TABLE table-name USE PROFILE
```

DB2 collects statistics for the table specified by table-name according to the collection options that are specified in the profile, and issues a message to indicate that the profile was used. If no profile exists for the specified table, DB2 issues an error message.

11.2.2 Updating RUNSTATS profiles

You can modify options to change the statistics that are collected by existing RUNSTATS profiles that you have created, or those that are created for autonomic statistics monitoring by the ADMIN_UTL_MONITOR stored procedure.

To modify an existing RUNSTATS profile, issue following RUNSTATS control statement:

```
RUNSTATS TABLESPACE ts-name TABLE table-name runstats-options UPDATE PROFILE
```

DB2 replaces any existing options that are specified in PROFILE_TEXT column of the SYSIBM.SYSTABLES_PROFILES catalog table with values options that are specified in runstats-options for the table that is specified by table-name. Any options that are not specified remain unchanged in the existing profile. If no profile exists for the specified table, DB2 issues an error message.

11.2.3 Deleting RUNSTATS profiles

You can delete an existing RUNSTATS profile. To delete existing RUNSTATS profiles, issue the following utility control statement:

```
RUNSTATS TABLESPACE ts-name TABLE options DELETE PROFILE
```

11.2.4 Combining autonomic and manual statistics maintenance

When autonomic statistics maintenance is enabled, you can still invoke the RUNSTATS utility to capture statistics manually.

When autonomic statistics monitoring and maintenance is enabled, DB2 uses RUNSTATS profiles to maintain the statistics for each table that is not excluded from autonomic maintenance. However, you can still explicitly invoke the RUNSTATS utility at any time either in the traditional manner, or by using profiles at any time.

To effectively combine autonomic and manual statics maintenance activities, you might follow the following recommendations:

- Before enabling autonomic statistics maintenance, consider whether to delete all existing RUNSTATS profiles by issuing RUNSTATS control statements and specifying the DELETE PROFILE option. By doing that, you enable DB2 to create new RUNSTATS profiles based on analysis of your existing statistics.

  However, this step is optional if you prefer that DB2 uses the settings of your existing RUNSTATS profiles for autonomic maintenance.
- When you want to collect statistics with different settings than those that are used for autonomic maintenance, use the traditional method for invoking the RUNSTATS utility and explicitly specify the options that you want to use.

Invoking RUNSTATS in that manner has no impact on the options that are specified in the profile, and periodic autonomic maintenance can continue unchanged. However, because the manual invocation of RUNSTATS does not change the RUNSTATS profile, the manually collected statistics might be lost at the next invocation of RUNSTATS that uses the profile. Consequently, you might want to update the profile to use the new options that were specified in the manual invocation.

- When you want to manually invoke the collection of statistics outside of the autonomic maintenance windows, but with the usual settings, you can specify the USE PROFILE option in the RUNSTATS control statement.
- When you want to modify the settings that are used for autonomic maintenance, you can issue a RUNSTATS control statement with the UPDATE PROFILE option and specify that options that you want to change. After you update the profile, DB2 uses the new options for autonomic maintenance activities.

11.3 RECOVER with BACKOUT YES

Whether you recover to the current point or a point-in-time recovery, up to DB2 9 for z/OS, the recover always identifies the best fitting recovery base and then performs a forward log recovery up to the point on the log that you specify as the target point in time.

Archived

Let us assume that the most recent recovery base is about 24 hours old and that the recovery base is a regular sequential image copy as shown in Figure 11-10. Let us assume also that you plan to use the RECOVER utility to remove the inserts that started about 1 hour ago.

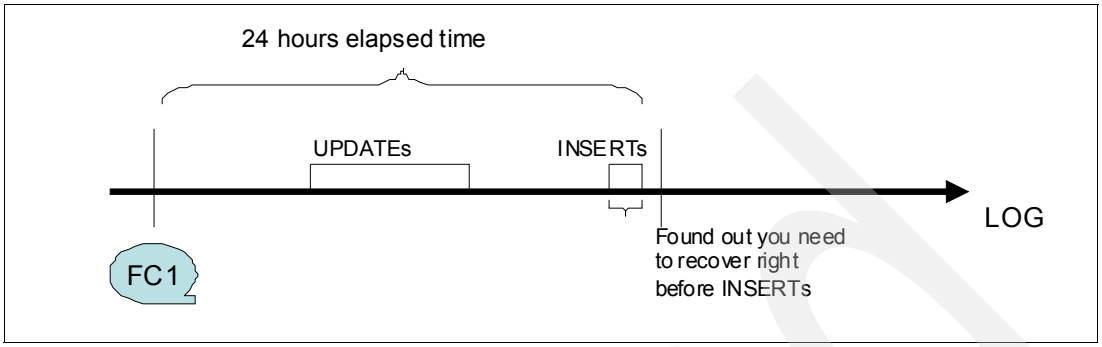


Figure 11-10 RECOVER BACKOUT YES: Base situation

In DB2 9, you can remove the inserts by recovering the entire table space to an RBA that is previous to the first insert or during the inserts, as shown in Figure 11-11. DB2 creates a new VSAM cluster, restores the image copy data set, reads SYSIBM.SYSLGRNX to find RBA ranges during which the cluster was involved in updates, and applies the needed log records reading forward in the log. All this activity can take a significant amount of elapsed time and resources.

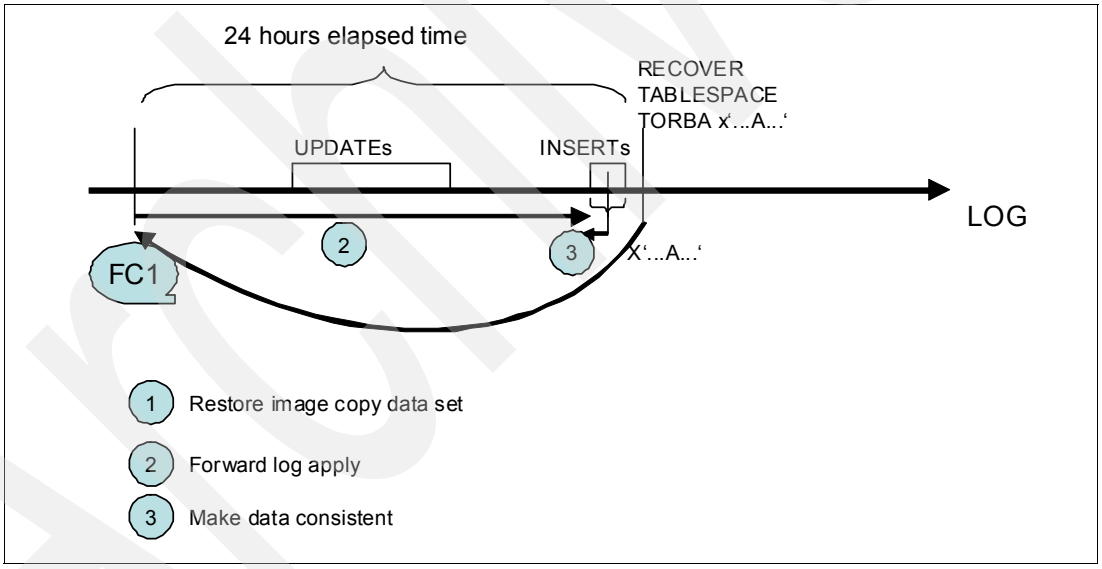


Figure 11-11 RECOVER TABLESPACE without BACKOUT YES

With DB2 10, you can now back out changes starting from the currently operational page set using the RECOVER TABLESPACE statement and the BACKOUT YES option. You can use BACKOUT YES together only with TOLOGPOINT or TORBA. As shown in Figure 11-12, DB2 performs the following steps to handle the BACKOUT YES request.

1. First, DB2 identifies the latest checkpoint that occurred prior to the point in time (RBA) that you specified on the RECOVER TABLESPACE statement.
2. Starting from there, DB2 performs a current status rebuild to identify open URs that it needs to handle as part of the BACKOUT YES process. We call this the LOGCSR phase.
3. When completed, DB2 lists all open URs in the RECOVER TABLESPACE job output.

4. With this knowledge, DB2 can now back out data up to the RBA that you specified as the point in time RBA and can go back even further, if needed, to make the data consistent.

In our example, the point-in-time RBA x'...A...' was in the middle of the stream of inserts. To make this UR consistent, DB2 continues to back out all inserts belonging to this URID.

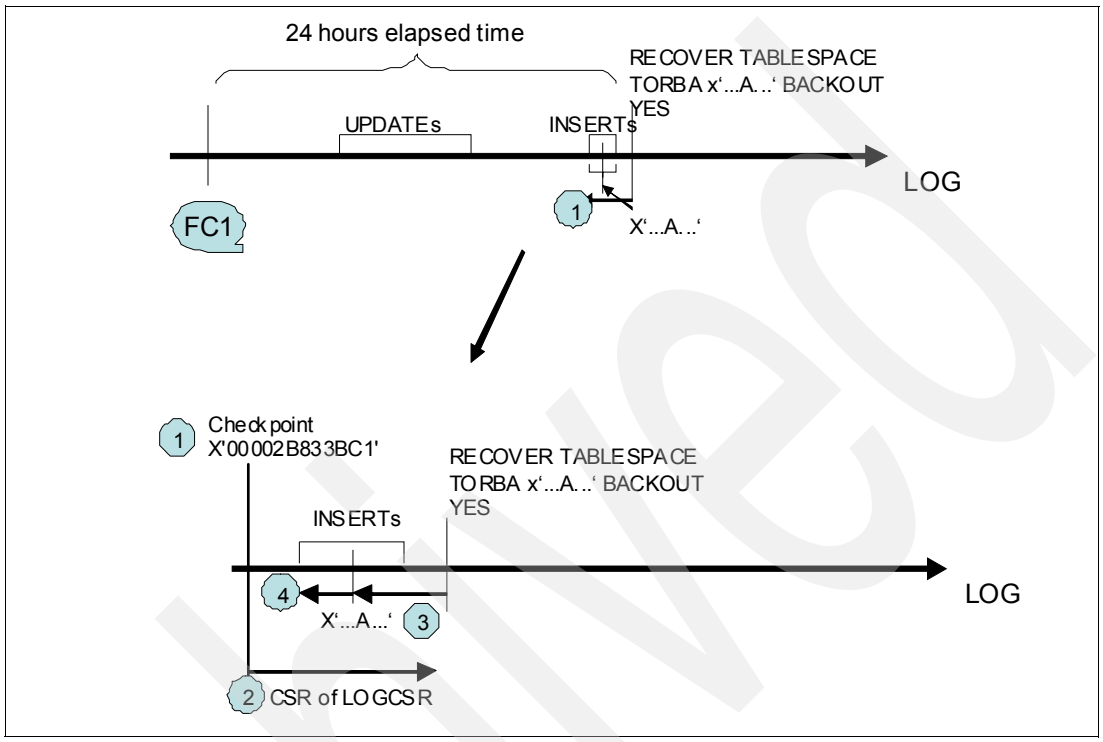


Figure 11-12 RECOVER TABLESPACE ... BACKOUT YES

Example 11-13 shows the original utility messages that illustrates this scenario.

Example 11-13 RECOVER TABLESPACE ... BACKOUT YES job output

```
DSNU050I    235 19:08:59.33 DSNUGUTC -  RECOVER TABLESPACE SLBDB.SLBTS TORBA X'2B838400' BACKOUT YES
 DSNU1542I -DBOB 235 19:08:59.33 DSNUCAIN - RECOVER BACKOUT ANALYSIS PROCEEDING FROM CURRENT LOGPOINT OF X'00002B840312'
DSNU532I  -DBOB 235 19:08:59.36 DSNUCALA - RECOVER TABLESPACE SLBDB.SLBTS   START
 DSNU1550I -DBOB 235 19:08:59.39 DSNUCALC - LOGCSR IS STARTED FOR MEMBER         , PRIOR CHECKPOINT RBA =
 X'00002B833BC1'
 DSNU1551I -DBOB 235 19:08:59.39 DSNUCALC - LOGCSR IS FINISHED FOR MEMBER         , ELAPSED TIME = 00:00:00
 DSNU1552I -DBOB 235 19:08:59.39 DSNUCALC - LOGCSR PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU1553I -DBOB 235 19:08:59.39 DSNUCALC - RECOVER UTILITY DETECTS THE FOLLOWING ACTIVE URS:
             INFLIGHT = 0, INABORT = 0, INDOUBT = 0, POSTPONED ABORT = 0 COMMITTED = 1, ABORTED = 0

                MEM    T  CONNID     CORRID     AUTHID    PLAN   S     URID       DATE       TIME
                       B TSO      DB2R8        DB2R8    DSNESPCS E 00002B83698B 2010-08-23 23.03.45
                   DBNAME  SPACENAME DBID/PSID PART     RBA
                  SLBDB    SLBTS     0166/0002 0000 00002B836A87

 DSNU1554I -DBOB 235 19:08:59.43 DSNUCALU - LOGUNDO IS STARTED FOR MEMBER
 DSNU1556I -DBOB 235 19:08:59.43 DSNUCALU - LOGUNDO IS FINISHED FOR MEMBER         , ELAPSED TIME = 59:08:59
 DSNU1557I -DBOB 235 19:08:59.43 DSNUCALU - LOGUNDO PHASE COMPLETE, ELAPSED TIME = 59:08:59
 DSNU500I    235 19:08:59.45 DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:00:00
 DSNU010I    235 19:08:59.45 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

After this recovery, the SYSIBM.SYSCOPY table contains the following information:

ICTYPE	P (point-in-time recovery)
PIT_RBA	The RBA or LOGPOINT that you specified in your TORBA or TOLOGPOINT keyword.
START_RBA	The RBA that represents the end of the utility execution
STYPE	B (RECOVER utility with BACKOUT keyword)

Remember that, starting with DB2 9, a point-in-time recovery does not leave you in an open UR. So, in our example, even though the PIT_RBA represents a point on the log that is in the middle of an open UR, the data represents the situation as it was just before the beginning of this UR.

This scenario is uncommon because only one table space with only one transaction was involved. In a real environment, this scenario probably would look quite different. Figure 11-13 shows a slightly more complex scenario.

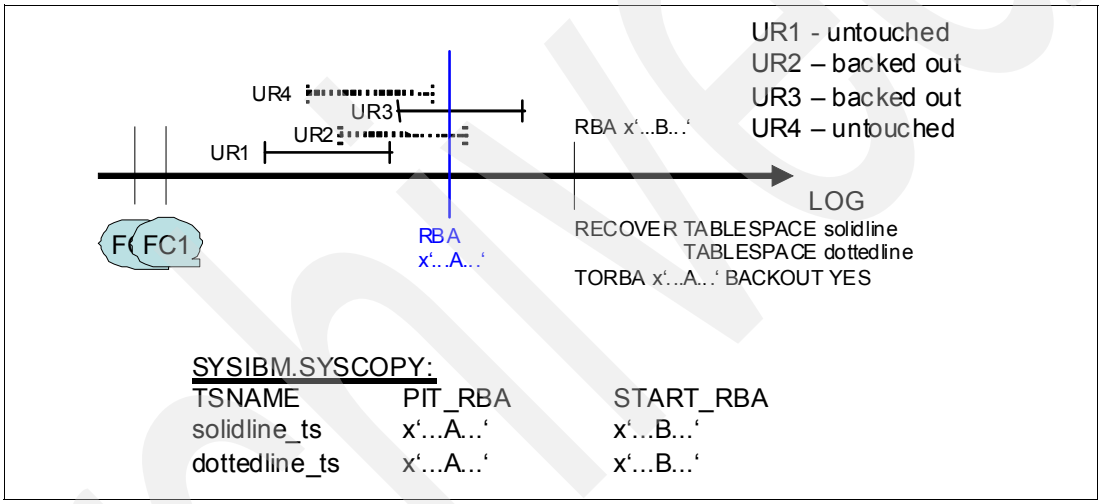


Figure 11-13 Recover multiple table spaces to the same UR with BACKOUT YES

In this scenario, we assume that we have two different table spaces, solidline_ts and dottedline_ts. UR1 and UR3 invoke solidline_ts. Both URs were committed at the time the RECOVER TABLESPACE utility was issued. The same is true for UR2 and UR4, which work with dottedline_ts. The RECOVER utility statement initiates a point-in-time recovery. The point in time that we picked for this example is somewhere in the middle of UR2 and UR3. As a result, DB2 backs out all changes for UR3 and UR2 until x'..A..' and then continues beyond this RBA, because beginning with DB2 9 for z/OS, every execution of the RECOVER utility results in consistent data.

The PIT_RBA in the corresponding row in SYSIBM.SYSCOPY is the RBA that you specified for your recovery. It is not the RBA of the last backed out record. The START_RBA that goes into SYSIBM.SYSCOPY is the RBA the system was currently on when the RECOVER utility finished.

After running the RECOVER, you cannot run any subsequent RECOVER TABLESPACE ... BACKOUT YES job with RBAs specified that are smaller than the START_RBA of the latest BACKOUT YES recovery that you ran. An attempt to do so ends in the error message shown in Example 11-14.

Example 11-14 Error message preventing BACKOUT YES recovery

```
DSNU556I 16:49:31.17 DSNUCASA - RECOVER CANNOT PROCEED FOR TABLESPACE
BACKYES.BACKTS
BECAUSE A SYSIBM.SYSCOPY RECORD HAS BEEN ENCOUNTERED WHICH HAS
DBNAME=BACKYES TSNAME=BACKTS DSNUM=0 ICTYPE=P STYPE=B
STARTRBA=X'00002D60AED0' LOWDSNUM=0 HIGHDSNUM=0
```

However, if you have to restore data to an earlier point in time, you can do so by not specifying the BACKOUT YES keyword on the utility control statement. DB2 then uses the regular recovery steps with the image copy that is as close as possible prior to the specific point-in-time RBA.

11.4 Online REORG enhancements

When we discuss *online REORG*, we mean REORG.... SHRLEVEL REFERENCE or REORG.... SHRLEVEL CHANGE. This function is enhanced throughout several DB2 releases to bring more usability and to help users with continuous availability.

This section describes several improved functions in DB2 10 to online REORG.

11.4.1 Improvements to online REORG for base tables spaces with LOBs

The improvements that we describe here are available only for partitioned table spaces in DB2 10 NFM.

In DB2 9, when REORG is run against a partition-by-growth table space with LOBs, the data records from part *n* before REORG need to go back into part *n* after REORG. Thus, there is no data movement across partitions when LOB is present, which can lead to REORG failure due to space issues.

DB2 10 addresses this issue with AUX YES support. Data movement across partitions during REORG on partition-by-growth with LOB is possible by means of REORG moving the corresponding LOB data.

The new keyword AUX YES/NO is available for REORG TABLESPACE SHRLEVEL REFERENCE and SHRLEVEL CHANGE utility executions. The AUX keyword allows you to specify whether DB2 reorganizes associated auxiliary LOB table spaces along with the base table space.

A value of YES means that LOB objects are handled automatically, and NO means that LOB objects are not touched during REORG.

When AUX YES is specified, DB2 fetches the row ID for each row as the rows are unloaded and it then probes the auxiliary index(es) to unload the corresponding LOB or LOBs. In contrast, if you REORG the LOB table space by itself, DB2 does an index scan of the auxiliary index and unloads the LOBs in row ID sequence. Thus, although the result of using AUX YES is almost the same as though you separately reorganized the base table space, there is a performance cost of being able to move rows from one partition to another. This performance

cost can be mitigated by using a large buffer pool for the auxiliary indexes separate from the LOB table spaces.

If the performance cost is unacceptable to you and if you do not need to move rows across partitions, then you might want to explicitly specify AUX NO.

DB2 chooses a default value for the AUX keyword as described in Table 11-2 for the REORG options REBALANCE, SHRLEVEL NONE, DISCARD, or REORP.

Table 11-2 REORG TABLESPACE ... AUX option

REORG options	AUX YES	AUX NO	AUX omitted
REBALANCE	YES	REORG rejected	YES
SHRLEVEL NONE	REORG rejected	REORG rejected	N/A
DISCARD	YES	LOB table space in CHKP afterwards	YES
REORP	YES	NO	YES

If none of the conditions shown in Table 11-2 apply, then depending on the table space type, the defaults listed in Table 11-3 apply.

Table 11-3 REORG TABLESPACE ... AUX option

Table space type	AUX YES	AUX NO	AUX omitted
Partition-by-growth	YES	NO	YES[a]
Range-partitioned	YES	NO	NO
Classic partitioned	YES	NO	NO

a. The default for partition-by-growth can be NO if MAXPARTS is 1 to avoid the extra cost of AUX YES.

For example, if you reorganize a range-partitioned table space using the following statement, the reorganization handles only the base table space as indicated in cell range-partitioned or AUX omitted, and LOBs are not touched as part of this REORG:

```
REORG TABLESPACE TESTDB.TESTTS SHRLEVEL REFERENCE
```

If you try to reorganize your table space with the following statement, you receive return code 8, which indicates that SHRLEVEL NONE (second body row in Table 11-2) does not allow you to use the AUX keyword. As a consequence the REORG utility run is rejected.

```
REORG TABLESPACE TESTDB.TESTTS SHRLEVEL NONE AUX NO
```

If you run REORG with one of the combinations of keywords and objects that show a YES in Table 11-2, DB2 automatically reorganizes the LOB table spaces that are associated to the table space that is reorganized.

Assume that you run the REORG utility using the following utility control statement to reorganize a classic partitioned table space:

```
REORG TABLESPACE TESTDB.TESTTS SHRLEVEL REFERENCE AUX YES
```

Archive

As shown in Table 11-2, REORG handles both the base table space and the associated auxiliary table space or spaces. Example 11-15 on page 454 shows the job output.

Messages DSNU1155I provide the names of the auxiliary table spaces that are reorganized with the TESTDB.TESTTS base table space.

As for any other online REORG, an inline copy for the base table space is mandatory. Refer to message DSNU400I for information about the creation of the image copy. We allocated a dedicated image copy data set in the JCL using the following JCL:

```
//DSNUPROC.SYSCOPY DD DSN=DB2R8.testdb.testts.A1,
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,,ROUND),
//     UNIT=SYSDA
```

The REORG SHRLEVEL CHANGE syntax automatically uses DD name SYSCOPY for its inline copy and did the same thing during our utility execution. However, the REORG TABLESPACE syntax does not allow you to specify other DD name references for the reorganized LOB table spaces. As a consequence, you have to choose between the following methods to have an image copy of all reorganized table spaces:

- Let DB2 put the reorganized LOB objects into COPY pending state and, if you want to clear the COPY PENDING state, run the COPY utility after the REORG has finished.
- Use TEMPLATE together with the REORG utility. TEMPLATE generates image copy names for not only the base table space, but also for the reorganized auxiliary table spaces.

Note that for some applications, you might not need to create image copies for the LOBs because you do not want DB2 to manage the recoverability for the LOBs.

Example 11-15 REORG ... AUX YES job output

```
DSNU000I    231 17:44:55.33 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
 DSNU1044I   231 17:44:55.35 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
0DSNU050I    231 17:44:55.36 DSNUGUTC -  REORG TABLESPACE TESTDB.TESTTS SHRLEVEL REFERENCE
AUX YES
 DSNU1155I -DBOB 231 17:44:56.28 DSNURFIT - AUXILIARY TABLESPACE TESTDB.TESTTSL1 WILL BE
REORGANIZED IN PARALLEL WITH BASE TABLESPACE
 DSNU1155I -DBOB 231 17:44:56.28 DSNURFIT - AUXILIARY TABLESPACE TESTDB.TESTTSL2 WILL BE
REORGANIZED IN PARALLEL WITH BASE TABLESPACE
 DSNU1155I -DBOB 231 17:44:56.28 DSNURFIT - AUXILIARY TABLESPACE TESTDB.TESTTSL3 WILL BE
REORGANIZED IN PARALLEL WITH BASE TABLESPACE
 DSNU251I    231 17:44:56.35 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=3 FOR TABLESPACE TESTDB.TESTTS PART 1
 DSNU251I    231 17:44:56.35 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=5 FOR TABLESPACE TESTDB.TESTTS PART 2
 DSNU251I    231 17:44:56.35 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=0 FOR TABLESPACE TESTDB.TESTTS PART 3
 DSNU252I    231 17:44:56.35 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=8 FOR TABLESPACE TESTDB.TESTTS
 DSNU250I    231 17:44:56.35 DSNURULD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU400I    231 17:44:56.59 DSNURBID - COPY PROCESSED FOR TABLESPACE TESTDB.TESTTS
                     NUMBER OF PAGES=13
                     AVERAGE PERCENT FREE SPACE PER PAGE = 13.76
                     PERCENT OF CHANGED PAGES =100.00
                     ELAPSED TIME=00:00:00
 DSNU303I  -DBOB 231 17:44:56.61 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=3
FOR TABLE DB2R8.PARTTB PART=1
 DSNU303I  -DBOB 231 17:44:56.61 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=5
FOR TABLE DB2R8.PARTTB PART=2
```

```
 DSNU303I  -DBOB 231 17:44:56.61 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=0
FOR TABLE DB2R8.PARTTB PART=3
 DSNU304I  -DBOB 231 17:44:56.61 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=8
FOR TABLE DB2R8.PARTTB
 DSNU302I    231 17:44:56.61 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS
PROCESSED=8
 DSNU300I    231 17:44:56.61 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU042I    231 17:44:56.61 DSNUGSOR - SORT PHASE STATISTICS -
                       NUMBER OF RECORDS=8
                       ELAPSED TIME=00:00:00
 DSNU349I  -DBOB 231 17:44:56.65 DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=8 FOR
INDEX DB2R8.PARTTB_IX1
 DSNU258I    231 17:44:56.65 DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=1
 DSNU259I    231 17:44:56.65 DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU387I    231 17:44:56.85 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU428I    231 17:44:56.85 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
TESTDB.TESTTS
 DSNU1157I   231 17:44:56.85 DSNURSWT - INLINE IMAGE COPIES ARE NOT CREATED FOR AUXILIARY
TABLE SPACES REORGANIZED AND ARE LEFT IN COPY PENDING
 DSNU568I  -DBOB 231 17:44:57.38 DSNUGSRX - INDEX DB2R8.PARTTB_IX1 IS IN INFORMATIONAL COPY
PENDING STATE
 DSNU381I  -DBOB 231 17:44:57.38 DSNUGSRX - TABLESPACE TESTDB.TESTTSL1 IS IN COPY PENDING
 DSNU381I  -DBOB 231 17:44:57.38 DSNUGSRX - TABLESPACE TESTDB.TESTTSL2 IS IN COPY PENDING
 DSNU381I  -DBOB 231 17:44:57.38 DSNUGSRX - TABLESPACE TESTDB.TESTTSL3 IS IN COPY PENDING
 DSNU010I    231 17:44:57.38 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
```

Messages DSNU1157I and DSNU3811I in Example 11-15 show that we did not use TEMPLATE for our utility execution and therefore all LOB table space are now in COPY pending. A quick way to copy all these table spaces is to use FlashCopy image copies. Refer to 11.1, “Support FlashCopy enhancements” on page 426 for detailed information about FlashCopy image copies.

To draw attention to the necessary subsequent image copies, the job ends with return code 4.

FlashCopy note: Specifying FLASHCOPY YES also does not provide relief here. If you use FLASHCOPY YES, DB2 only creates FlashCopy image copies for the base table space, but not for the auxiliary table spaces. Refer to 11.1, “Support FlashCopy enhancements” on page 426 for more about FlashCopy image copies.

As mentioned earlier, you have the chance to let the REORG utility complete with RC 0. You can use a TEMPLATE statement just before the REORG, which generates the needed SYSCOPY DD names for “inline” image copies for the LOB objects also.

Example 11-16 shows the job output DB2 generates if you use the following utility control statements:

```
TEMPLATE SCOPY1  UNIT(SYSDA) DISP(MOD,CATLG,CATLG)
                 SPACE=(10,10) TRK
                 DSN(DB2R8.&SN..D&JDATE..T&TIME..P&PART.)

REORG TABLESPACE DSN00063.PARTTB
SHRLEVEL REFERENCE
AUX YES COPYDDN(SCOPY1)
```

The TEMPLATE statement should contain &SN or &TS to make sure that DB2 can create separate copy data sets for each object that needs to be copied. You can specify only one

template for all the image copies. You cannot distinguish between the LOB and the base objects.

Example 11-16 shows that image copy data sets have been allocated for each individual page set involved in the reorganization. As a consequence of having completed the copies by the end of the utility execution, the return code is 0 and the whole reorganized table space and auxiliary table spaces are available immediately after the job finishes.

Example 11-16 REORG ... AUX YES and TEMPLATE job output

```
DSNU050I    231 18:43:45.54 DSNUGUTC -  TEMPLATE SCOPY1 UNIT(SYSDA) DISP(MOD, CATLG, CATLG)
SPACE=(10, 10) TRK DSN( DB2R8.&SN..D&JDATE..T&TIME..P&PART.)
 DSNU1035I   231 18:43:45.54 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
ODSNU050I    231 18:43:45.54 DSNUGUTC -  REORG TABLESPACE TESTDB.TESTTS SHRLEVEL REFERENCE
AUX YES COPYDDN(SCOPY1)
 DSNU1155I -DBOB 231 18:43:46.50 DSNURFIT - AUXILIARY TABLESPACE TESTDB.TESTTSL1 WILL BE
REORGANIZED IN PARALLEL WITH BASE TABLESPACE
 DSNU1155I -DBOB 231 18:43:46.50 DSNURFIT - AUXILIARY TABLESPACE TESTDB.TESTTSL2 WILL BE
REORGANIZED IN PARALLEL WITH BASE TABLESPACE
 DSNU1155I -DBOB 231 18:43:46.50 DSNURFIT - AUXILIARY TABLESPACE TESTDB.TESTTSL3 WILL BE
REORGANIZED IN PARALLEL WITH BASE TABLESPACE
 DSNU1038I   231 18:43:46.54 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=SCOPY1
                      DDNAME=SYS00001
                      DSN=DB2R8.TESTTS.D2010231.T224345.P00000
 DSNU1038I   231 18:43:46.57 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=SCOPY1
                      DDNAME=SYS00002
                      DSN=DB2R8.TESTTSL1.D2010231.T224345.P00000
 DSNU1038I   231 18:43:46.60 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=SCOPY1
                      DDNAME=SYS00003
                      DSN=DB2R8.TESTTSL2.D2010231.T224345.P00000
 DSNU1038I   231 18:43:46.64 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=SCOPY1
                      DDNAME=SYS00004
                      DSN=DB2R8.TESTTSL3.D2010231.T224345.P00000
 DSNU251I    231 18:43:46.71 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=3 FOR TABLESPACE TESTDB.TESTTS PART 1
 DSNU251I    231 18:43:46.71 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=5 FOR TABLESPACE TESTDB.TESTTS PART 2
 DSNU251I    231 18:43:46.71 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=0 FOR TABLESPACE TESTDB.TESTTS PART 3
 DSNU252I    231 18:43:46.71 DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS
UNLOADED=8 FOR TABLESPACE TESTDB.TESTTS
 DSNU250I    231 18:43:46.71 DSNURULD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU400I    231 18:43:47.01 DSNURBID - COPY PROCESSED FOR TABLESPACE TESTDB.TESTTS
                     NUMBER OF PAGES=13
                     AVERAGE PERCENT FREE SPACE PER PAGE = 13.76
                     PERCENT OF CHANGED PAGES =100.00
                     ELAPSED TIME=00:00:00
 DSNU400I    231 18:43:47.04 DSNURBID - COPY PROCESSED FOR TABLESPACE TESTDB.TESTTSL1
                     NUMBER OF PAGES=41
                     AVERAGE PERCENT FREE SPACE PER PAGE =  0.19
                     PERCENT OF CHANGED PAGES =100.00
                     ELAPSED TIME=00:00:00
 DSNU400I    231 18:43:47.05 DSNURBID - COPY PROCESSED FOR TABLESPACE TESTDB.TESTTSL2
                     NUMBER OF PAGES=41
                     AVERAGE PERCENT FREE SPACE PER PAGE =  0.19
                     PERCENT OF CHANGED PAGES =100.00
                     ELAPSED TIME=00:00:00
 DSNU400I    231 18:43:47.06 DSNURBID - COPY PROCESSED FOR TABLESPACE TESTDB.TESTTSL3
                     NUMBER OF PAGES=41
                     AVERAGE PERCENT FREE SPACE PER PAGE =  0.19
```

```
                    PERCENT OF CHANGED PAGES =100.00
                    ELAPSED TIME=00:00:00
 DSNU303I  -DB0B 231 18:43:47.07 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=3
FOR TABLE DB2R8.PARTTB PART=1
 DSNU303I  -DB0B 231 18:43:47.07 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=5
FOR TABLE DB2R8.PARTTB PART=2
 DSNU303I  -DB0B 231 18:43:47.07 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=0
FOR TABLE DB2R8.PARTTB PART=3
 DSNU304I  -DB0B 231 18:43:47.07 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=8
FOR TABLE DB2R8.PARTTB
 DSNU302I    231 18:43:47.08 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS
PROCESSED=8
 DSNU300I    231 18:43:47.08 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU042I    231 18:43:47.08 DSNUGSOR - SORT PHASE STATISTICS -
                    NUMBER OF RECORDS=8
                    ELAPSED TIME=00:00:00
 DSNU349I  -DB0B 231 18:43:47.13 DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=8 FOR
INDEX DB2R8.PARTTB_IX1
 DSNU258I    231 18:43:47.13 DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=1
 DSNU259I    231 18:43:47.13 DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:00
 DSNU387I    231 18:43:47.42 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU428I    231 18:43:47.43 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
TESTDB.TESTTS
 DSNU428I    231 18:43:47.43 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
TESTDB.TESTTSL1
 DSNU428I    231 18:43:47.43 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
TESTDB.TESTTSL2
 DSNU428I    231 18:43:47.43 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
TESTDB.TESTTSL3
 DSNU568I  -DB0B 231 18:43:47.95 DSNUGSRX - INDEX DB2R8.PARTTB_IX1 IS IN INFORMATIONAL COPY
PENDING STATE
 DSNU010I    231 18:43:47.96 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Attention: This function introduces incompatible changes for some objects types or types of REORG actions. If you omit the AUX keyword and do either of the following actions, AUX objects are now reorganized together with the base table space:

- REORG partition-by-growth table space
- REORG ... REBALANCE
- REORG ... DISCARD
- REORG an object currently in REORP status

Remember that if you do not use TEMPLATE with REORG, DB2 runs those reorganizations and ends with return code 4, because the AUX objects are put into COPY pending status and because they are available only for RW access.

Increased logging volume during REORG

The online REORG allows read/write access to a table space during most of the utility execution. Because write access is allowed on the operational data sets, the changes made while the utility works on the shadow copy of the page set must be applied to the shadow copy at some point in time by reading the log records that were written since the utility execution started.

When you create large LOB objects, it is common to specify the NOT LOGGED parameter for the LOB table space to avoid too much logging volume. Using the NOT LOGGED attribute is quite common and does not really cause issues during normal operation, for example because ROLLBACKs work for LOBs due to the special locking that is done for LOB objects. However, if DB2 allows you to reorganize LOBs with SHRLEVEL CHANGE, it is necessary to

turn on full logging for all LOB objects for the duration of the REORG utility run, which is done automatically by DB2. You do not have to change anything for the table space definition.

11.4.2 REORG SHRLEVEL options for LOB table spaces

Up to DB2 Version 8, the only possible SHRLEVEL option for the reorganization of LOBs was SHRLEVEL NONE, with the following restrictions:

- No read/write availability during entire duration of the utility
- No space reclamation on the LOB table space
- No inline copy is created

As of DB2 9, you can run REORG TABLESPACE SHRLEVEL REFERENCE on an LOB table space, which provides better availability, space reclamation, and overall performance.

In DB2 10, REORG TABLESPACE SHRLEVEL CHANGE on an LOB table space is supported. This option provides even greater availability by enabling full read/write access during almost the entire duration of REORG.

In DB2 10 CM, you can still use SHRLEVEL NONE. You also get a new message with a warning that the function will be deprecated. After you move to DB2 10 NFM, if you run the same job, it gives return code 0 and a warning message that the function is not available. In NFM, you must change utility jobs from SHRLEVEL NONE to either REFERENCE or SHRLEVEL CHANGE.

Table 11-4 provides an overview of the availability of SHRLEVEL for LOB REORG, depending on the DB2 version that you use.

Table 11-4 Available SHRLEVELs for LOB REORG depending on DB2 versions

REORG LOB SHRLEVEL	DB2 V8 and earlier	DB2 9	DB2 10 CM	DB2 10 NFM
NONE	YES	YES	YES	NO
REFERENCE	NO	YES	YES	YES
CHANGE	NO	NO	NO	YES

Just as a reminder, the improvements discussed in 11.4.1, “Improvements to online REORG for base tables spaces with LOBs” on page 452 are available for REORG using SHRLEVEL REFERENCE or SHRLEVEL CHANGE.

Option SHRLEVEL CHANGE is already available for you starting with DB2 10 CM and SHRLEVEL NONE is no longer honored in DB2 10 NFM.

11.4.3 Improved usability of REORG of disjoint partition ranges

When you reorganize partitioned table spaces in DB2 8 or DB2 9, you have the option to not reorganize all partitions but to either pick one partition or a range of partitions and run REORG TABLESPACE for only a few of them. However, you cannot reorganize several separate partitions or multiple partition ranges or mix single partitions with partition ranges in one REORG TABLESPACE control statement. Thus, for example, if you want to reorganize partition 3, 8, and the range of partitions 12-15, you must issue three different REORG TABLESPACE jobs.

With DB2 10 for z/OS, you can reorganize a list of single partitions and partition ranges in one utility control statement as shown in Figure 11-14. In this figure, the little arrow shows that you can go back after you specified one partition or one range or partitions and add more selections separated by comma. A SYSIBM.SYSCOPY row with the appropriate STYPE will be inserted for each involved partition. Parallelism will depend on the number of tasks assigned considering on number of indexes and partitions.

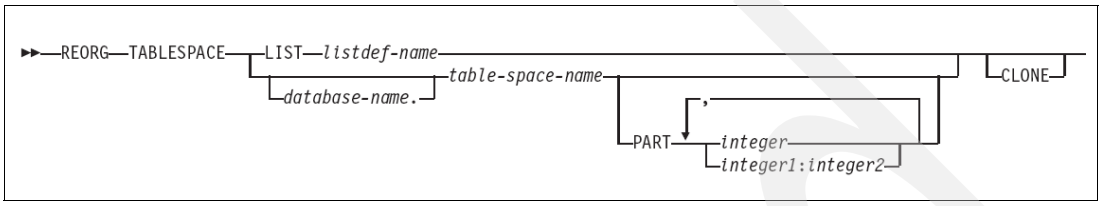


Figure 11-14 Enhanced REORG TABLESPACE syntax for PART keyword

Thus, you can specify:

```
REORG TABLESPACE TESTDB.TS1 PART 3 , 8 , 12:15
```

Important: The partition number that you specify here must be the physical partition number.

The following values are not allowed in this context:

- Duplicate partition numbers
- Overlapping partition ranges
- A partition range with a starting partition number that is greater than the ending partition number
- Out-of-sequence partition ranges
- REBALANCE of non contiguous partitions

After the REORG of partition ranges, you might consider to REORG INDEX the affected non-partitioning indexes (NPIs).

Note that the results of rotating a specific physical partition from its original logical number to last (refer to 4.5, “Support for rotating partitions” on page 100 to learn more about rotating physical partitions) have impact on partition numbering. Let us assume that you executed a few ALTER TABLE ... ROTATE. Now you run a -DIS DB(..) command, and DB2 returns to you the result shown in Figure 11-15.

Archive

```
DSNT360I  -DBOB **********************************
DSNT361I  -DBOB *  DISPLAY DATABASE SUMMARY
                *    GLOBAL
DSNT360I  -DBOB **********************************
DSNT362I  -DBOB     DATABASE = ROTATE  STATUS = RW
                    DBD LENGTH = 4028
DSNT397I  -DBOB
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
TS1      TS    0001 RW
TS1      TS    0005 RW
TS1      TS    0003 RW
    -THRU      0004
TS1      TS    0002 RW
TS1      TS         RW
******* DISPLAY OF DATABASE ROTATE   ENDED     **********************
DSN9022I  -DBOB DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

Figure 11-15 -DIS DB command output after a couple of ROTATE commands

The order in which the partitions are listed in Figure 11-15 follows the logical partition order. The partitions that you see are the physical partition numbers. When you run a DB2 online utility and you want to run it on specific partitions, you must use the physical partition numbers on the utility control statement.

So, for example, if you want to reorganize logical partitions 3-5 storing data for the months March through May, use the following utility control statement:

```
REORG TABLESPACE ROTATE.TS1 2:4
```

This improved usability is available in DB2 10 NFM.

LISTDEF is widely used among DB2 8 and 9 customers. As a consequence the LISTDEF syntax for REORG is enhanced as shown in Figure 11-16, so that you can specify multiple partitions or partition ranges to be reorganized in one REORG statement.

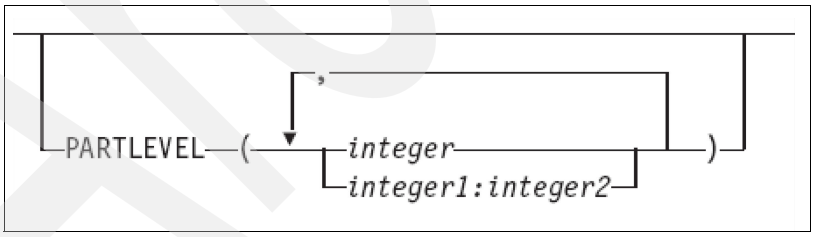


Figure 11-16 LISTDEF syntax change for multiple partition ranges

11.4.4 Cancelling blocking claimers with REORG FORCE

Today, database administrators face a tremendous task in coordinating the concurrent executions of the DB2 REORG utility and SQL applications. For the REORG utility to complete, the utility (even the REORG SHRLEVEL CHANGE) must gain exclusive control over the database objects that are reorganized by draining all access to them. For this break-in to happen while concurrent SQL applications are running, the utility must rely on well-behaved applications that commit frequently. If there are long running noncommitting readers or overlapping claimers, this break-in is not always possible.

If DB2 cannot break into the other running processes on a given table space, online REORG fails. If online REORG fails after running many hours, it is not restartable in most cases. So, the resources used up to this point on reorganizing the shadow data sets are wasted.

With DB2 10, the REORG utility includes an option that allows you to cancel blocking claimers when it needs to get exclusive control over the target objects. Figure 11-17 shows the syntax for REORG that addresses this issue.

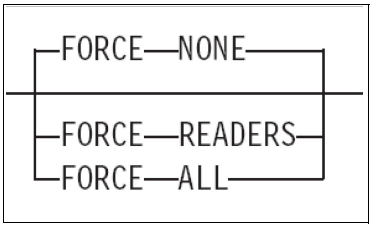


Figure 11-17 FORCE keyword on REORG

You have the following options here:

FORCE NONE	Specifies that no action is taken when REORG performs drain. The REORG utility waits for the claimers to commit. The utility will time out or restart when the drain fails, as determined by existing conditions.
FORCE READERS	Specifies that read claimers are canceled when REORG is requesting a drain all on the last RETRY iteration. Also refer to 4.7, “Long-running reader warning message” on page 106 to learn how to identify long-running readers automatically and have them reported to you so that you can take pro-active action.
FORCE ALL	Specifies that both read and write claimers are canceled when REORG is requesting a drain all or drain writers on the last RETRY iteration.

Looking at the two options FORCE READERS and FORCE ALL, you can assume that FORCE READERS should not cause too much trouble and almost certainly give you the intended benefits. This situation should especially be the case if you use DSNZPARM LRDTHLD and know that readers usually behave quite well.

Archive

There might be a downside with the FORCE ALL option. Let us assume the situation shown in Figure 11-18. A long running UR is currently working on the table space for which you intend to run your REORG TABLESPACE utility. You start the utility after this UR did thousands of inserts without committing the work. Online REORG with option FORCE ALL starts reorganizing on the shadow data sets.

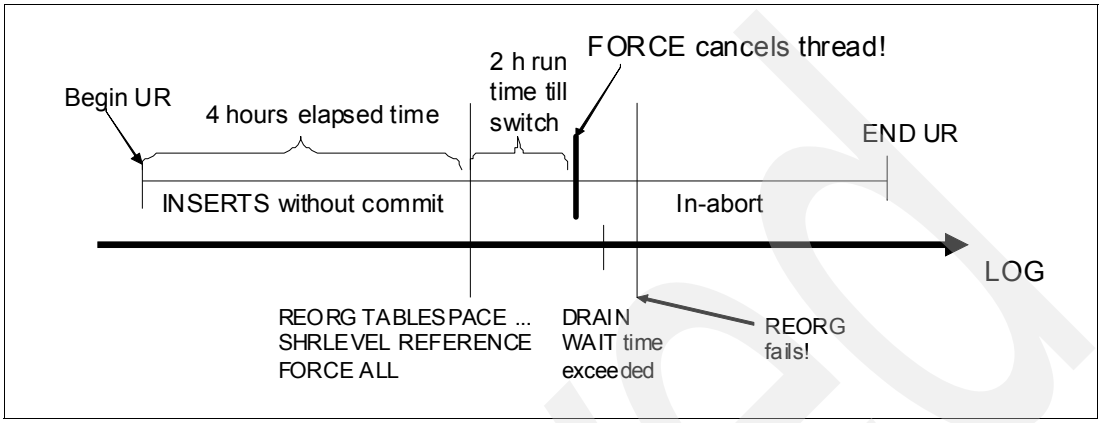


Figure 11-18 Possible effect of option FORCE ALL on REORG

When the data is reorganized on the shadow data set, online REORG needs exclusive access to the table space to do the final SWITCH from the shadow data sets to the operational page sets. Because this unit of recovery is still holding the operational resources and because you used FORCE ALL on the utility control statement, DB2 cancels the thread with the long running UR.

Cancelling a thread does not mean that all locks are immediately released. DB2 must now roll back the uncommitted changes to maintain data consistency. If the rollback does not complete within the period of time that you specified as DRAIN WAIT time on the utility control statement, the REORG utility abends. In our example in Figure 11-18, we assume that this is the case.

As a consequence, using FORCE ALL on our REORG table space statement leads to the following results:

- Work of UR rolled back. The job that was intended to do this work has to be resubmitted and work needs to be re-done.
- The REORG used up a lot of resources, but in the end the data is still not reorganized. You must run REORG again at some other time.

You must specify the REORG FORCE ALL parameter on the initial REORG utility control statement. You cannot use ALTER UTILITY to change the initial setting of FORCE while the utility runs.

Restrictions for FORCE READERS and FORCE ALL

When we tried to generate meaningful job output messages, we created an open unit of recovery using read and write operations with SPUFI and AUTOCOMMIT set to NO. Let us first use FORCE READERS. We selected all rows of a table residing in the table space which we planned to REORG using the following utility control statement:

```
REORG TABLESPACE TESTDB.TESTTS SHRLEVEL REFERENCE AUX YES FORCE READERS
```

What we expected was that the hanging unit of recovery would be cancelled and that the REORG would be able to complete with the SWITCH phase. The result would be a reorganized table space. Instead, the REORG utility hangs.

In our testing, the utility tried to run six times and finally ended with the set of messages and return code 8 shown in Example 11-17.

Example 11-17 Unsuccessful attempt to work with FORCE READERS

```
DSNU1122I -DBOB 276 09:01:40.66 DSNURSWD - JOB DB2R8B PERFORMING
REORG
       WITH UTILID TEMP UNABLE TO DRAIN TESTDB.TESTTS.
       RETRY 1 OF 6 WILL BE ATTEMPTED IN 180 SECONDS
DSNU1122I -DBOB 276 09:05:10.79 DSNURSWD - JOB DB2R8B PERFORMING
REORG
       WITH UTILID TEMP UNABLE TO DRAIN TESTDB.TESTTS.
       RETRY 2 OF 6 WILL BE ATTEMPTED IN 180 SECONDS
DSNU1122I -DBOB 276 09:08:40.86 DSNURSWD - JOB DB2R8B PERFORMING
REORG
       WITH UTILID TEMP UNABLE TO DRAIN TESTDB.TESTTS.
       RETRY 3 OF 6 WILL BE ATTEMPTED IN 180 SECONDS
DSNU1122I -DBOB 276 09:12:10.93 DSNURSWD - JOB DB2R8B PERFORMING
REORG
       WITH UTILID TEMP UNABLE TO DRAIN TESTDB.TESTTS.
       RETRY 4 OF 6 WILL BE ATTEMPTED IN 180 SECONDS
DSNU1122I -DBOB 276 09:15:41.01 DSNURSWD - JOB DB2R8B PERFORMING
REORG
WITH UTILID TEMP UNABLE TO DRAIN TESTDB.TESTTS.
       RETRY 5 OF 6 WILL BE ATTEMPTED IN 180 SECONDS
DSNU1122I -DBOB 276 09:19:11.09 DSNURSWD - JOB DB2R8B PERFORMING
REORG
       WITH UTILID TEMP UNABLE TO DRAIN TESTDB.TESTTS.
       RETRY 6 OF 6 WILL BE ATTEMPTED IN 180 SECONDS
DSNU590I  -DBOB 276 09:22:41.16 DSNURDRN - RESOURCE NOT AVAILABLE, REASON=X'00C
PROCESSING
DSNT360I  -DBOB **********************************
DSNT361I  -DBOB *  DISPLAY DATABASE SUMMARY
               *    GLOBAL CLAIMERS
DSNT360I  -DBOB **********************************
DSNT362I  -DBOB     DATABASE = TESTDB  STATUS = RW
                   DBD LENGTH = 8066
DSNT397I  -DBOB
NAME     TYPE PART  STATUS           CONNID   CORRID      CLAIMINFO
-------- ---- ----- ---------------- -------- ------------ --------
TESTTS   TS    0001 RW,UTRO          TSO      DB2R8        (CS,C)
-                  AGENT TOKEN 543
TESTTS   TS    0002 RW,UTRO          TSO      DB2R8        (CS,C)
   -                  AGENT TOKEN 543
TESTTS   TS    0003 RW,UTRO          TSO      DB2R8        (CS,C)
   -                  AGENT TOKEN 543
TESTTS   TS         RW,UTRO          TSO      DB2R8        (CS,C)
   -                  AGENT TOKEN 543
******* DISPLAY OF DATABASE TESTDB   ENDED      **********************
DSN9022I  -DBOB DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
DSNU568I  -DBOB 276 09:22:41.72 DSNUGSRX - INDEX DB2R8.PARTTB_IX1 IS IN
INFORMATIONAL COPY PENDING
DSNU012I    276 09:22:41.73 DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST
RETURN CODE = 8
```

Archived

The CLAIMERS that are shown in the -DISPLAY DATABASE command output show agent token 543 as claimer.

DB2 has not yet released the lock on the table. If we issue a -DIS DB(TESTDB)SP(*) LOCKS command, the result is shown in Example 11-18. As shown from the LOCKINFO, agent 543 has only read interest on the page set.

Example 11-18 -DIS DB ... LOCKS for database object

```
NAME     TYPE PART  STATUS            CONNID   CORRID       LOCKINFO
-------- ---- ----- ----------------- -------- ------------ ---------
TESTTS   TS    0001 RW,UTRO           TSO      DB2R8        H-IS,P,C
    -                  AGENT TOKEN 543
TESTTS   TS    0002 RW,UTRO           TSO      DB2R8        H-IS,P,C
    -                  AGENT TOKEN 543
```

So what is causing the REORG utility not to be able to complete the SWITCH phase? The reason is that FORCE READERS and FORCE ALL can cancel only the application that is holding locks on the page set if the agent is currently active in DB2.

If you look at the output of a -DISPLAY THREAD(*) command in the given situation, you see the result shown in Example 11-19. Only the thread with token 561 is a real active thread within this DB2 subsystem and only this one would be affected by FORCE READERS or FORCE ALL. The indicator as of whether the thread is currently active in DB2 is the asterisks (*) in column A of message DSN402I.

Example 11-19 -DIS THREAD(*) output for agent 543

```
DSNV401I  -DBOB DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DBOB ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
RRSAF    T     5457 DBOBADMT_DMN STC      ?RRSAF   00A2     2
RRSAF    T        5 DBOBADMT_II  STC      ?RRSAF   00A2     3
TSO      N       72 DB2R8        DB2R8              00AF     0
TSO      T       20 DB2R8        DB2R8    DSNESPCS 00AF   543
TSO      T  *     3 DB2R8        DB2R8              00AF   561
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I  -DBOB DSNVDT '-DIS THREAD' NORMAL COMPLETION
***
```

Attention: If the REORG utility with FORCE READERS or FORCE ALL can kick out with a DRAIN and actually force out readers or both readers and writers, the utility output messages do not indicate which URs (that is, which users, plans, packages, and so on) were kicked out.

11.5 Increased availability for CHECK utilities

When you run the CHECK DATA or CHECK LOB utility on a DB2 subsystem of version 9 or less and the utility detects referential integrity violations, it places the dependent object into check pending (CHKP) restrictive state. As a result, the table space which was fully accessible before you ran the CHECK DATA or CHECK LOB utility is now completely restricted for any SQL access. This can be a major issue for some DB2 users who expect a maximum in high availability in their DB2 operations.

The described behavior of DB2 9 and under can be an inadequate constraint for you for the following reasons:

- Today table spaces can bear inconsistencies but are accessible because they have no restricted DBET-state (CHKP). Running any of the check utilities results in the situation that a table space suddenly is no longer accessible although contents have not changed.
- Usually if a table space has inconsistencies this is limited to a small portion of the data. Having a restricted DBET-state makes the table space unavailable in its entirety.

With DB2 10, utilities CHECK DATA and CHECK LOB have been changed so that it is more in line with the behavior CHECK INDEX shows today already. When you run all of these utilities with the new functionality turned on, DB2 no longer sets the CHKP status for table space or LOB objects when you run those utilities and DB2 finds inconsistencies. At a first sight you might find this strange, but this effectively does not cause you any problems. Running the check utility is a completely voluntary action. Error handling is integrated into DB2 for dealing with inconsistencies in DB2 objects so that there is no need for the CHECK utility to make objects inaccessible. The main purpose of the CHECK utilities is to report problems and help you to obtain details about possible inconsistencies.

A DSNZPARM is introduced by DB2 10 CM to allow you to turn this new functionality on or off: CHECK_SETCHKP YES or NO.

- CHECK_SETCHKP=NO is the default value and means that you do not want the check pending state set if inconsistencies are found.
- CHECK_SETCHKP=YES tells DB2 to function as it used to in DB2 9 for z/OS and prior.

Figure 11-19 shows a job output for a CHECK DATA utility execution on a DB2 subsystem with CHECK_SETCHKP set to YES. As expected, it set the CHKP DBET state on table space CONSIST.TS1 after finding a few inconsistencies.

```
01.55 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
01.57 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
01.58 DSNUGUTC -  CHECK DATA TABLESPACE CONSIST.TS1 SCOPE ALL
01.62 DSNUKDST - CHECKING TABLE DB2R8.CONSISTTB1
01.71 DSNUGSOR - SORT PHASE STATISTICS -
NUMBER OF RECORDS=32
ELAPSED TIME=00:00:00
...
...
DSNUKERK - ROW (RID=X'000000021F') HAS NO PARENT FOR DB2R8.CONSISTTB1.EMPN1EZK
DSNUKERK - ROW (RID=X'0000000220') HAS NO PARENT FOR DB2R8.CONSISTTB1.EMPN1EZK
DSNUKDAT - CHECK TABLE DB2R8.CONSISTTB1 COMPLETE, ELAPSED TIME=00:00:00
.72 DSNUGSRX - TABLESPACE CONSIST.TS1 IS IN CHECK PENDING
DSNUK001 - CHECK DATA COMPLETE,ELAPSED TIME=00:00:00
DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
```

Figure 11-19 CHECK DATA with inconsistencies found and CHECK_SETCHKP=YES

In contrast to this, we ran the exact same job with DSNZPARM CHECK_SETCHKP set to NO, as shown in Figure 11-20. If you compare both job outputs, you notice that CHKP now is not set.

```
01.55 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
01.57 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
01.58 DSNUGUTC -  CHECK DATA TABLESPACE CONSIST.TS1 SCOPE ALL
01.62 DSNUKDST - CHECKING TABLE DB2R8.CONSISTTB1
01.71 DSNUGSOR - SORT PHASE STATISTICS -
NUMBER OF RECORDS=32
ELAPSED TIME=00:00:00
...
...
DSNUKERK - ROW (RID=X'000000021F') HAS NO PARENT FOR DB2R8.CONSISTTB1.EMPN1EZK
DSNUKERK - ROW (RID=X'0000000220') HAS NO PARENT FOR DB2R8.CONSISTTB1.EMPN1EZK
DSNUKDAT - CHECK TABLE DB2R8.CONSISTTB1 COMPLETE, ELAPSED TIME=00:00:00
DSNUK001 - CHECK DATA COMPLETE,ELAPSED TIME=00:00:00
DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
```

Figure 11-20 CHECK DATA with inconsistencies found and CHECK_SETCHKP=NO

However, both jobs end with return code 4, which should alarm and urge you to further investigate and correct the situation that prevented the job from ending with return code 0.

Table 11-5, shows the effect of running various CHECK utilities under different circumstances and the resulting DBET state. So, for example, the first row reads as follows:

- You run a CHECK LOB utility which identifies errors.
- Before you ran the CHECK LOB utility the LOB table space was not restricted.
- In DB2 9 or DB2 10 with DSNZPARM CHECK_SETCHKP set to YES, DB2 sets the restrictive CHKP on your LOB table space.
- In DB2 10 with DSNZPARM CHECK_SETCHKP set to NO, the LOB table space remains unrestricted, that is the CHKP state is not set.

Table 11-5 DBET state by function

Function	DBET state	Behavior prior to DB2 10 or DB2 10 with CHECK_SETCHKP= YES	Behavior in DB2 10 with CHECK_SETCHKP= NO
CHECK LOB (errors found)	not restricted	set CHKP	not restricted
	CHKP	keep CHKP	keep CHKP
CHECK DATA ... DELETE NO (errors found)	not restricted	set CHKP	not restricted
	CHKP	keep CHKP	keep CHKP
CHECK DATA ... DELETE YES (errors found)	not restricted	not restricted	not restricted
	CHKP	not restricted	not restricted
CHECK LOB (errors not found)	not restricted	not restricted	not restricted
	CHKP	not restricted	not restricted

Function	DBET state	Behavior prior to DB2 10 or DB2 10 with CHECK_SETCHKP= YES	Behavior in DB2 10 with CHECK_SETCHKP= NO
CHECK DATA DELETE NO (errors not found)	not restricted	not restricted	not restricted
	CHKP	not restricted	not restricted
CHECK DATA DELETE YES (errors not found)	not restricted	not restricted	not restricted
	CHKP	not restricted	not restricted

This behavior is available in DB2 10 CM.

11.6 IBM DB2 Sort for z/OS

IBM DB2 Sort for z/OS, V1.1.0 (5655-W42) is a DB2 tool that provides high-speed utility sort processing for data stored in DB2 for z/OS. DB2 Sort optimizes overall system efficiency through reduced CPU consumption, and improves memory management by exploiting the advanced facilities of the IBM z/OS operating system and IBM System z server.

DB2 Sort for z/OS provides the following benefits:

- Helps reduce CPU consumption by IBM utilities sort
- Allows flexibility to optimize utility sort processing
- Improves application availability by reducing the batch window for utility maintenance

DB2 Sort for z/OS was announced 20 August 2010 by IBM U.S. Software Announcement 210-233, which is available at:

http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ca&infotype=an&appname=iSource&supplier=897&letternum=ENUS210-233

DB2 Sort for z/OS was made available on 24 September 2010. IBM DB2 Utilities Suite for z/OS, V8.1.0 (5655-K61), V9.1.0 (5655-N97), and V10.1 (5655-V41) can make use of DB2 Sort for z/OS.

This product is not designed to be a replacement for general purpose sorting products that are invoked through the standard z/OS system sort interface. DB2 Sort provides an alternative sort engine that can be used by the DB2 utilities.

For the enablement of DB2 SORT use by the utilities, you need to set a DSNZPARM DB2SORT=ENABLE in DB2 V8 and DB2 9. In DB2 10, DB2 SORT is enabled by default. If DB2SORT parameter is set to DISABLE or if DB2 Sort for z/OS cannot be found by the utilities, then sort processing continues to use DFSORT.

For versions prior to DB2 10, you must enable DB2 Sort in DB2 APAR PM12819 (PTFs: V8 UK59100 or V9 UK59101).

For DB2 Sort, the following maintenance is recommended:

- DB2 Sort APAR PM21747 (PTF UK60466)
- DB2 utilities APAR PM21277 (Open PTFs: V8 UK61211 or V9 UK61213)

The IBM DB2 Utilities where you see performance benefits are LOAD, REORG, RUNSTATS, REBUILD INDEX, CHECK INDEX, and CHECK DATA.

The following workloads more likely to benefit from utility sort processing and DB2 Sort:

- Highly-transactional workloads performing lots of insert, update, delete operations requiring RUNSTATS and REORG
- Applications such as data warehousing applications that are performing frequent or large volumes of loading data requiring LOAD and REBUILD INDEX

As for DFSORT, DSNZPARMs UTSORTAL=YES and IGNSORTN=YES are considered beneficial for sort execution.

For details, see *IBM DB2 Sort for z/OS Version 1 Release 1 User's Guide*, SC19-2961.

11.7 UTSERIAL elimination

You can experience time outs on DB2 utility jobs serializing on the global UTSERIAL lock resource when you execute many utilities concurrently. If your utility timed-out, you have to re-submit the utility job. This is not always a trivial task, because the utility might have been one step in a multi-step process and as a consequence you have to edit the used JCL quite often to perform the restart of the multi-step process. The whole situation could potentially even become worse with DB2 10, because the virtual storage bottleneck is basically removed and therefore DB2 can now serve many more parallel active threads than it could in DB2 9 and earlier.

With DB2 10 the majority of time outs on the global UTSERIAL lock resource are eliminated. The new utility lock scheme is enabled automatically during the ENFM migration process. You do not have to take any action and your existing workload running DB2 Utilities can continue to run without modifications.

DB2 now uses a new type of IRLM lock to serialize utility access to the DSNDB01.SYSUTILX table space. This lock is a commit-duration lock that is used exclusively by DB2 utilities during utility compatibility checking, with a resource name (`dbid.psid`) for the database objects that are targeted by the utility. The objective of these locks is to prevent two or more incompatible utilities from serializing successfully when neither has its compatibility information stored in DSNDB01.SYSUTILX yet. These object level locks show up in lock traces as `uobj`.

11.8 REPORT utility output improvement

DB2 9 supports the use of System Level Backups (SLB) for object level recovery. Thus, you can use system level backups for the recovery of single page sets. To create a SLB using the BACKUP SYSTEM utility, you must set the DSNZPARM SYSTEM_LEVEL_BACKUPS parameter explicitly to YES.

In DB2 9, if you turn on the functionality to use system level backups for object level recoveries, DB2 searches the information available in SYSIBM.SYSCOPY and BSDS during RECOVER and picks the most recent recovery base as basis for your recovery. If you want to know before you start the RECOVER utility whether a recent image copy or a SLB will be used for an upcoming recovery, you must manually compare the information in SYSIBM.SYSCOPY or the output of the REPORT RECOVERY output with the BACKUP SYSTEM summary section in the DSNJU004 (print log map) utility output. The REPORT RECOVERY utility does not consider SLBs in DB2 9 for z/OS.

This process can be cumbersome and time consuming, and it can also be a source for errors especially when you use data sharing with multiple BSDSs involved.

With DB2 10, the REPORT RECOVERY utility checks for the setting of DSNZPARM SYSTEM_LEVEL_BACKUPS. If SYSTEM_LEVEL_BACKUPS is set to YES, the REPORT RECOVERY utility considers all sorts of copy resources and lists them in its job output as shown in Example 11-20.

The first recorded copy is a full image copy taken on August 23. The second, fourth and fifth oldest resources containing copy information about the table space that we used as input for REPORT RECOVERY are system level backups. The type listed here is set to SLB.

Example 11-20 REPORT RECOVERY output with SLBs available

```
DSNU050I    235 16:05:33.59 DSNUGUTC -  REPORT RECOVERY TABLESPACE SLBDB.SLBTS
 DSNU581I  -DBOB 235 16:05:33.59 DSNUPREC - REPORT RECOVERY TABLESPACE SLBDB.SLBTS
 DSNU593I  -DBOB 235 16:05:33.59 DSNUPREC -  REPORT RECOVERY ENVIRONMENT RECORD:
              MINIMUM   RBA: 000000000000
              MAXIMUM   RBA: FFFFFFFFFFFF
              MIGRATING RBA: 000000000000
 DSNU582I  -DBOB 235 16:05:33.59 DSNUPPCP - REPORT RECOVERY TABLESPACE SLBDB.SLBTS SYSCOPY ROWS
                            AND SYSTEM LEVEL BACKUPS
 TIMESTAMP = 2010-08-23-15.37.15.708487,  IC TYPE = *C*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =00002B626279
 DEV TYPE  =         ,                    IC BACK =   ,   STYPE   = L,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
 LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED = Y,   TTYPE =
 JOBNAME   =         ,     AUTHID =           ,     COPYPAGESF = -1.0E+00
 NPAGESF   = -1.0E+00           ,              CPAGESF = -1.0E+00
 DSNAME    = SLBDB.SLBTS                              , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M

 TIMESTAMP = 2010-08-23-15.49.36.775583,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =00002B65B117
 DEV TYPE  =         ,                    IC BACK = FC,   STYPE   = T,  FILE SEQ = 0000,  PIT LRSN  = 00002B65B14B
 LOW DSNUM = 0001,   HIGH DSNUM = 0001,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED = Y,   TTYPE = C
 JOBNAME   = DB2R8L  ,     AUTHID =   DB2R8   ,     COPYPAGESF = -1.0E+00
 NPAGESF   = -1.0E+00           ,              CPAGESF = -1.0E+00
 DSNAME    = DBOBI.SLBDB.SLBTS.N00001.CYWJOAOV          , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M

 TIMESTAMP = 2010-08-23-15.58.08.859580,  TYPE    = SLB    ,                                  RBLP =00002B63FEEB
 TOKEN     = C4C2F0C2C67933C8A1AFBC8E00002B63FEEB          , MEMBER NAME =      , DATA COMPLETE LRSN =00002B66BE23

 TIMESTAMP = 2010-08-23-16.00.52.402213,  TYPE    = SLB    ,                                  RBLP =00002B646BAD
 TOKEN     = C4C2F0C2C679363B2ED0C44B00002B646BAD          , MEMBER NAME =      , DATA COMPLETE LRSN =00002B6AE9FB

 TIMESTAMP = 2010-08-23-16.01.47.490481,  IC TYPE = *X*,  SHR LVL =  ,  DSNUM    = 0000,  START LRSN =00002B6F6EE2
 DEV TYPE  =         ,                    IC BACK =   ,   STYPE   =  ,  FILE SEQ = 0000,  PIT LRSN  = 000000000000
 LOW DSNUM = 0000,   HIGH DSNUM = 0000,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED = Y,   TTYPE =
 JOBNAME   = DB2R8L  ,     AUTHID =   DB2R8   ,     COPYPAGESF = -1.0E+00
 NPAGESF   = -1.0E+00           ,              CPAGESF = -1.0E+00
 DSNAME    = SLBDB.SLBTS                              , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M

 TIMESTAMP = 2010-08-23-16.02.08.422318,  TYPE    = SLB    ,                                  RBLP =00002B646BAD
 TOKEN     = C4C2F0C2C6793683ACC620C300002B646BAD          , MEMBER NAME =      , DATA COMPLETE LRSN =00002B71073B

 TIMESTAMP = 2010-08-23-16.02.31.166329,  IC TYPE =  F ,  SHR LVL = R,  DSNUM    = 0000,  START LRSN =00002B74FE9B
 DEV TYPE  =         ,                    IC BACK = FC,   STYPE   = T,  FILE SEQ = 0000,  PIT LRSN  = 00002B74FECF
 LOW DSNUM = 0001,   HIGH DSNUM = 0001,   OLDEST VERSION = 0000,   LOGICAL PART = 0000,   LOGGED = Y,   TTYPE = C
 JOBNAME   = DB2R8L  ,     AUTHID =   DB2R8   ,     COPYPAGESF = -1.0E+00
 NPAGESF   = -1.0E+00           ,              CPAGESF = -1.0E+00
 DSNAME    = DBOBI.SLBDB.SLBTS.N00001.CYWJ5XF3          , MEMBER NAME =         , INSTANCE = 01, RELCREATED = M
```

If DSNZPARM SYSTEM_LEVEL_BACKUP is set to NO, DB2 is not allowed to use the system level backups for any object level recoveries. In this case the same REPORT RECOVERY execution does not list the SLBs.

In addition and independent from the DSNZPARM setting, the REPORT RECOVERY utility has a new section at the bottom of the job output which shows information about system level

backups that exist or used to exist for this specific DB2 subsystem. Example 11-21 shows you this part of the REPORT RECOVERY utility output.

Example 11-21 REPORT RECOVERY utility output new section

```
START STCK                          DATA COMPLETE   DATA/LOG COMPLETE
 DATA               LOG                RBLP          LRSN        DATE      LTIME     LOCATION NAME
 ----------------  ----------------  ------------  ------------  --------------------  -------------
 C6576BD96976554E  00000000000000000  000004496E96  0000044A1630  2010-07-27  19.02.54  DB0B
 TOKEN = C4C2F0C2C6576BD96976554E000004496E96      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
 C65774DE18B08BCC  00000000000000000  0000044FC000  000004513F0F  2010-07-27  19.40.02  DB0B
 TOKEN = C4C2F0C2C65774DE18B08BCC0000044FC000      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
 C65775BE6A86D98E  00000000000000000  0000044FE820  000004554000  2010-07-27  19.43.58  DB0B
 TOKEN = C4C2F0C2C65775BE6A86D98E0000044FE820      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
 C65775C8F1304049  00000000000000000  0000044FE820  000004579AE0  2010-07-27  19.44.09  DB0B
 TOKEN = C4C2F0C2C65775C8F130404900000044FE820      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
 C657791B88C7F14C  00000000000000000  0000045C5090  0000045DF4D8  2010-07-27  19.59.01  DB0B
 TOKEN = C4C2F0C2C657791B88C7F14C0000045C5090      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
 C6577CA08664498E  00000000000000000  0000046507E6  00000468FCA8  2010-07-27  20.14.45  DB0B
 TOKEN = C4C2F0C2C6577CA08664498E0000046507E6      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
 C65842C897C29E4C  00000000000000000  0000048C76F8  0000048C9E08  2010-07-28  11.01.18  DB0B
 TOKEN = C4C2F0C2C65842C897C29E4C0000048C76F8      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
 C67933C8A1AFBC8E  00000000000000000  00002B63FEEB  00002B66BE23  2010-08-23  15.58.08  DB0B
 TOKEN = C4C2F0C2C67933C8A1AFBC8E00002B63FEEB      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
 C679363B2ED0C44B  00000000000000000  00002B646BAD  00002B6AE9FB  2010-08-23  16.00.52  DB0B
 TOKEN = C4C2F0C2C679363B2ED0C44B00002B646BAD      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
 C6793683ACC620C3  00000000000000000  00002B646BAD  00002B71073B  2010-08-23  16.02.08  DB0B
 TOKEN = C4C2F0C2C6793683ACC620C300002B646BAD      INCREMENTAL = N, SUBSYSTEM ID = DB0B, MEMBER NAME =
 Z/OS = 1.11,  CAPTURE CATALOG INFO = Y,   LOG COPY POOL = N
```

The fact that you see multiple SLBs in Example 11-21 does not mean that you can still access all of them during potential RECOVER utility executions. In our case, the copypool definition only allows two versions so all but the youngest two SLBs do not exist anymore.

Going back to Example 11-20, the generated output is a bit misleading here as well. The following SLB is no longer physically available:

```
TIMESTAMP = 2010-08-23-15.58.08.859580,  TYPE    = SLB    ,                                  RBLP =00002B63FEEB
 TOKEN     = C4C2F0C2C67933C8A1AFBC8E00002B63FEEB          , MEMBER NAME =      , DATA COMPLETE LRSN =00002B66BE23
```

The SLB might not be available, but there can be dump copies that are still available for use.

12

Installation and migration

In this chapter, we provide information to help you evaluate the changes in DB2 10 for z/OS and to plan for a successful installation or migration to DB2 10 for z/OS.

This chapter includes the following topics:

- Planning for migration
- DB2 10 product packaging
- Command changes
- Catalog changes
- Implications of DB2 catalog restructure
- DSNZPARM change summary
- EXPLAIN tables in DB2 10
- SMS-managed DB2 catalog
- Skip level migration
- Fallback
- Improvements to DB2 installation and samples
- Simplified installation and configuration of DB2-supplied routines
- Eliminating DDF private protocol
- Precompiler NEWFUN option

© Copyright IBM Corp. 2010. All rights reserved.

12.1 Planning for migration

Before you begin the installation or migration process, look at the big picture. You need to be aware of the major requirements to get from your current version of DB2 to DB2 10 for z/OS. You need to know where you are currently and where you need to be before you embark on this process considering DB2, z/OS, and tools. You need to decide if it is convenient for you to take advantage of skip-level migration from DB2 V8 NFM directly to DB2 10 conversion mode (CM). Once you have started a migration, you are able to fall back but you can only remigrate the same way you tried before since the catalog will have the imprint of the level you went to.

Terminology note: In prior DB2 releases, the acronym *CM* was used to mean *compatibility mode*. In DB2 10, the CM acronym means *conversion mode*.

Figure 12-1 shows the life cycle of the versions of DB2 and the minimum level of z/OS that is required. At the time of the writing this book, the latest three versions of DB2 are fully supported and end of service for DB2 V8 is scheduled for 30 April 2012.

If you are running DB2 V7, you can order DB2 V8 although it is withdrawn from marketing.

Version	PID	Generally available	OS prereq	Marketing withdrawal	End of service
V3	5685-DB2	December 1993	MVS V4R3	February 2000	January 2001
V4	5695-DB2	November 1995	MVS V4R3	December 2000	December 2001
V5	5655-DB2	June 1997	MVS V4R3	December 2001	December 2002
V6	5645-DB2	June 1999	OS/390 V1R3	June 2002	June 2005
V7	5675-DB2	March 2001	OS/390 V2R7	March 2007	June 2008
V8	5625-DB2	March 2004	z/OS V1R3	September 2009	April 2012
V9	5635-DB2	March 2007	z/OS V1R7	TDB	TBD
V10	5605-DB2	October 2010	z/OS V1R10	TBD	TBD

Figure 12-1 DB2 version summary

For up to date DB2 product support life cycle information, go to:

http://www.ibm.com/support/entry/portal/Planning/Software/Information_Management/DB2_for_z~OS

Click **Product Support Lifecycle** under Featured Planning Links.

DB2 10 for z/OS operates on System z or equivalent processors running in 64-bit mode. The following processors are supported:

- z890
- z990
- z9®
- z10
- z196 and later processors

Note that z800 and z900 processors are *not* supported and some hardware functions might be available only on the later models. The processors must have enough real storage to satisfy the combined requirements of DB2 and z/OS. DB2 10 for z/OS generally requires increased real storage as compared to DB2 9 for z/OS.

Installation or migration to DB2 10 for z/OS might require that you migrate other system products to a new release. Refer to Table 12-1 for a list of minimum levels for required software, but check the program directory for the latest information.

Table 12-1 System software requirements

Program Number	Product name	Minimum level
5694-A01	z/OS	v01.10.00
	DFSMS	v01.10.00
	Language Environment® Base Services	v01.10.00
	Security Server/RACF	v01.10.00
5605-DB2	IRLM	v02.03.00
5655-G44	IBM SMP/E for z/OS	v03.04.00

Depending on when you start your migration to DB2 10, you might need to migrate to z/OS V1R11 or a later release because end of service for z/OS V1R10, which is the minimum required level, was scheduled for October 2011. You also need to have z/OS Unicode Services and appropriate conversion definitions for DB2 to function properly.

For more information about dates for z/OS, refer to:

`http://www.ibm.com/systems/z/os/zos/support/zos_eos_dates.html`

You can find other Information Management product life cycle dates at:

`http://www.ibm.com/software/data/support/lifecycle/`

Generally, new functions, including changes to existing functions, statements, and limits, are available only in new-function mode, unless explicitly stated otherwise. Exceptions to this general statement include optimization and virtual storage enhancements, which are also available in conversion mode unless stated otherwise. In Versions 8 and 9, most utility functions were available in conversion mode. However, for Version 10, most utility functions work only in new-function mode. Refer to Chapter 11, “Utilities” on page 425 for details. Virtual storage constraint relief is available in CM after REBIND. Refer to Chapter 3, “Scalability” on page 51 for details.

In the following sections, we discuss the prerequisites when planning and preparing systems to migrate to DB2 10.

Before you begin, review the program directories and informational APARs II4477 for DB2 9 and II4474 for DB2 V8 for the latest list of maintenance to apply before you migrate to DB2 10.

12.1.1 DB2 10 pre-migration health check job

Before migrating to DB2 10 CM, make sure that you run premigration job DSNTIJPA on the DB2 V8 system or DB2 9 system. You can run this job at any time during the premigration planing stages or any time that you want to determine the cleanup work that you need to do to DB2 V8 or DB2 9 to prepare for migration to DB2 10.

It is possible that DSNTIJPA was not delivered with your DB2 V8 or DB2 9 system software. DSNTIJPA job was delivered through the service stream in APAR PM04968. Apply this APAR, and run this job during the early stages of DB2 10 migration planning.

The PM04968 job checks for the existence of the following conditions:

- For migrations from DB2 V8, stored procedures that specify DSNWZPR as the external module name
- User-defined DB2 catalog indexes that reside on user-managed storage
- User-defined DB2 catalog indexes that reside on DB2-managed storage
- Stored procedures that are defined to run in the DB2 stored procedures address space
- Packages and plans that were bound prior to DB2 Version 5
- Authorizations IDs and roles with the EXECUTE privilege of the EXECUTE WITH GRANT OPTION privilege for one or more DB-supplied stored procedures
- Incomplete column definitions
- Schemas that have any EBCDIC EXPLAIN tables
- EXPLAIN tables that are not in the current release format
- DB2 EXPLAIN tables named DSN_PTASK_TABLE that have one or more column names containing a variant EBCDIC character such as the number (#) symbol (code point x'7b' in code page 37)
- Plans that contain DBRMs
- Plans that were bound with the ACQUIRE(ALLOCATE) option
- Packages that were bound with parallelism
- Packages and plans that were bound with DBPROTOCOL(PRIVATE) BIND option
- Obsolete objects that can be dropped after migration to new-function mode
- Package names and collection IDs that contain special characters that might cause package-not-found errors
- Packages that are affected by the additional authorization checking for UPDATE and DELETE statements
- For migration from DB2 9, all packages that reference the DSN_XMLVALIDATE user-defined function
- Catalog table spaces that have ?? enabled, which will be disabled during migration to CM
- Catalog table spaces that have ?? enabled, which will be disabled during ENFM
- Packages and plans that have a self-referencing table

12.1.2 Fallback SPE and maintenance

Your DB2 V8 or DB2 9 systems must be at the proper service level and you must have applied the fallback SPE APAR PK56922 and its prerequisite fixes before you can migrate to DB2 10 for z/OS. Your DB2 9 or DB2 V8 system must be started with the *fallback SPE* applied or the migration to DB2 10 terminates with the messages shown in Example 12-1.

If you use data sharing, all members of the data sharing group must have the fallback SPE applied before any member can be migrated to DB2 10. The fallback SPE must be on all active DB2 group members for DB2 10 to start.

Archive

Example 12-1 Message if fallback SPE is not applied

```
DSNR001I  -DB8A RESTART INITIATED
DSNR045I  -DB8A DSNRRPRC DB2 SUBSYSTEM IS STARTING
AND
          IT WAS NOT STARTED IN A
          PREVIOUS RELEASE WITH THE FALLBACK SPE APPLIED.
          FALLBACK SPE APAR: PK56922
          NEW RELEASE LEVEL: 0000D670
          KNOWN LEVEL(S): 0000D3010000D3250000D3500000D45000000000
DSNV086E -DB8A DB2 ABNORMAL TERMINATION REASON=00D96001
IEF450I DB8AMSTR DB8AMSTR - ABEND=S04F U0000 REASON=00000000
```

Keep in mind that migration to DB2 10 can be done only from DB2 V8 NFM or DB2 9 NFM. You cannot migrate from a release prior to DB2 V8 or from DB2 V8 CM or DB2 9 CM mode to DB2 10.

DB2 10 continues to use the same general approach for installing and migrating as with DB2 V8 and DB2 9. You can use the following CLIST modes:

- Use the INSTALL option to install a new DB2 10 subsystem.
- Use the MIGRATE option if you have existing DB2 V8 or DB2 9 systems that you want to migrate to DB2 10. If you use data sharing, run the MIGRATE option for each member of the data sharing group to get each member into DB2 10 CM.
- Use the ENFM option to enable new-function mode for a subsystem or an entire data sharing group. ENFM is a *group-wide* enablement. You need to run it only once for each data sharing group.
- Use the UPDATE option to update or change DSNZPARMs.

Figure 12-2 shows the migration stages to migrate from DB2 9 to DB2 10 for z/OS.

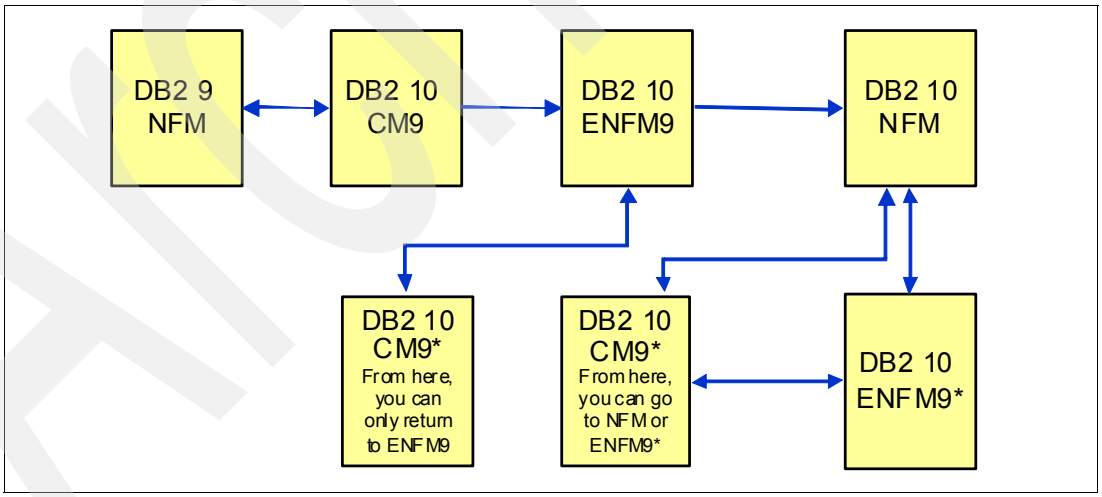


Figure 12-2 Migrating from DB2 9 to DB2 10 and fallback paths

Figure 12-3 shows the migration stages to migrate from DB2 V8 for z/OS to DB2 10 for z/OS.

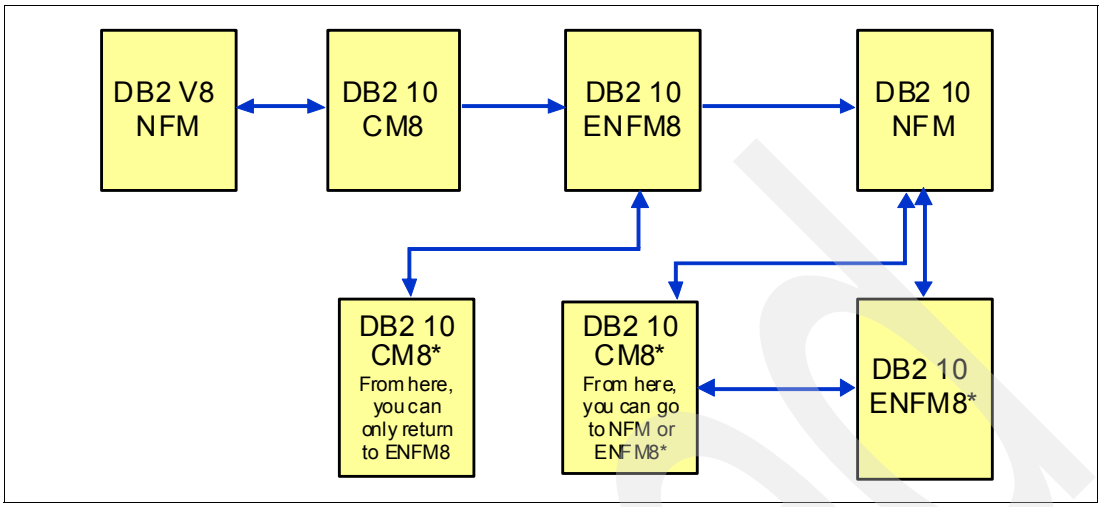


Figure 12-3 Migrating from DB2 V8 to DB2 10 and fallback paths

DB2 V8 provides *compatibility mode* (CM) and *enable-new-function mode* (ENFM). DB2 9 provides *conversion mode* (CM and CM*) and *enable-new-function mode* (ENFM and ENFM*).

DB2 10 renames the migration modes to take into account skip level migration. The mode names in DB2 10 are *conversion mode 8* (CM8 and CM8*), *conversion mode 9* (CM9 and CM9*), *enable new-function mode 8* (ENFM8 and ENFM8*), and *enable new-function mode 9* (ENFM9 and ENFM9*). These names so that you can determine the release of DB2 from which you migrated and return to the proper release level mode. They provide the ability for you to fall back to the prior mode and allow DB2 to implement all the fallback code in the new release and not the old release.

Important: You cannot return to the previous release of DB2 after you enter DB2 10 ENFM. Do *not* move to ENFM until you are sure that you do not need to return to the previous release.

The code base for DB2 10 ENFM and NFM is the same as CM. As shown in Figure 12-2 and Figure 12-3, you can always drop back to the previous mode in the migration path.

As further clarification:

- You cannot fall back to DB2 9 after you reach DNFM from DB2 9 (ENFM9); however, you can return to DB2 10 CM 9 (CM9*).
- You cannot fallback to DB2 V8 after you reach ENFM from V8 (ENFM8); however, you can return to DB2 10 CM 8 (CM8*).

Rebind is required for a long list of improvements in virtual storage use, optimization, and parallelism. Coexistence in data sharing needs to be complete before rebinding on DB2 10. The key improvements from REBIND in CM include SQL IN-list improvements, SQL pagination enhancements, query parallelism improvements, and more aggressive view and table expression merge.

If you are migrating from DB2 V8, run RUNSTATS before you REBIND for improved statistics for cluster ratio, data repeat factor, and high cardinality non-uniform distribution of data. If during your migration you have only one opportunity to rebind, it is best to do it in CM mode. If

you are migrating from DB2 V8, the convergence of the temporary database into the work file temporary database occurs in DB2 10 CM.

The PLANMGMT subsystem parameter in the DSN6SPRM macro affects the behavior of the REBIND PACKAGE and REBIND TRIGGER PACKAGE commands. PLANMGMT can have a value of OFF, BASIC, or EXTENDED. The default value is OFF for DB2 9. In DB2 10, the default value changes from to EXTENDED. If you move from DB2 9 and have used access path management, you might see some improvements. If you did not use access path management, you might note an increase in BIND CPU time from this change. If you want to reduce the CPU time for BIND, then set this parameter to OFF or BASIC.

12.1.3 Partitioned data set extended support

Another requirement is the use of partitioned data set extended (PDSE). You must define SDSNLOAD and SDSNLOD2 libraries as a PDSE instead of a partitioned data set (PDS). Some load modules exceed the 16 MB size, and a PDS does not support this size load module. The DSNALLOC installation job now allocates these libraries as PDSE.

12.1.4 Convert bootstrap data set to expanded format

Before migrating to DB2 10, make sure that the bootstrap data set (BSDS) is converted to the expanded format that became available in DB2 V8. This expanded format supports up to 10,000 entries per copy of archive logs and up to 93 entries per copy of active logs and entries for TCP/IP v6.

If you are migrating from DB2 9, you converted the BSDS to this format already, and no further action is required on your part.

If you are migrating from DB2 V8, you must verify the format of the BSDS. In DB2 V8, it was optional whether you converted the BSDS to this expanded form. You can check the format of the current BSDS by running the print log map stand-alone utility program (DSNJU004). Examine the output produced by DSNJU004 for the messages listed in Example 12-2 to confirm that the conversion occurred.

Example 12-2 Messages to indicate whether BSDS was converted

```
DSNJCNVB CONVERSION PROGRAM HAS NOT RUN

DSNJCNVB CONVERSION PROGRAM HAS RUN
```

Keep in mind that converting the BSDS requires a member outage. Make sure to take a backup of the BSDS before you convert it. You need to plan accordingly. If you fail to convert the BSDS prior to the first time you start DB2 10 in conversion mode, DB2 fails with the messages shown in Example 12-3.

Example 12-3 Expanded format BSDS required to start DB2 10

```
DSNY001I -DB8A SUBSYSTEM STARTING
DSNJ157I -DB8A BSDS HAS NOT BEEN CONVERTED. DSN=DB8AU.BSDS01
DSNJ119I -DB8A BOOTSTRAP ACCESS INITIALIZATION PROCESSING FAILED
DSNV086E -DB8A DB2 ABNORMAL TERMINATION REASON=00E80084
```

If you use the installation CLIST to generate your migration jobs, step DSNTCNVB of job DSNTIJUZ converts the BSDS to the new format if it is not already converted. However, this job does not resize the underlying VSAM data sets. Be aware that you run the risk of causing

the BSDS data sets to go into secondary extents or run the risk of running out of space after you take advantage of the capability to record 10,000 archive log and 93 active log data sets.

If you have not yet converted the BSDS to the new format, a better approach is to follow the steps outlined in the *DB2 Version 8 Installation Guide*, GC18-7418, which provides the steps that are required to perform the task and to run the DSNJCNVB conversion program. It provides the opportunity to resize the BSDS at the same time that you convert it. If you do not resize the BSDS when you convert it and if you later need to enlarge the size of the underlying data sets, you need to take an additional system outage. For your reference, Example 12-4 shows the output of a successful run of the DSNJCNVB conversion program.

Example 12-4 Output of successful run of DSNJCNVB conversion program

```
CONVERSION OF BSDS DATA SET - COPY 1, DSN=DB8AU.BSDS01
         SYSTEM TIMESTAMP   - DATE=2010.222  LTIME=18:29:46.69
         UTILITY TIMESTAMP  - DATE=2006.221  LTIME=20:11:55.68
         PREVIOUS HIKEY     - 04000053
         NEW HIKEY          - 040002F0
         RECORDS ADDED      - 669
CONVERSION OF BSDS DATA SET - COPY 2, DSN=DB8AU.BSDS02
         SYSTEM TIMESTAMP   - DATE=2010.222  LTIME=18:29:46.69
         UTILITY TIMESTAMP  - DATE=2006.221  LTIME=20:11:55.68
         PREVIOUS HIKEY     - 04000053
         NEW HIKEY          - 040002F0
         RECORDS ADDED      - 669
DSNJ260I  DSNJCNVB BSDS CONVERSION FOR DDNAME=SYSUT1 COMPLETED SUCCESSFULLY
DSNJ260I  DSNJCNVB BSDS CONVERSION FOR DDNAME=SYSUT2 COMPLETED SUCCESSFULLY
DSNJ200I  DSNJCNVB CONVERT BSDS UTILITY PROCESSING COMPLETED SUCCESSFULLY
```

Example 12-5 shows the output of an unsuccessful run of the DSNJCNB conversion program against a BSDS that is already converted. In this case, there is no damage done to the BSDS, so there is no corrective action needed if DSNJCNVB was run when it was not needed.

Example 12-5 Output of DSNJCNVB BSDS conversion program on an already converted BSDS

```
DSNJ440I  BSDS HAS ALREADY BEEN CONVERTED, DDNAME=SYSUT1
DSNJ440I  BSDS HAS ALREADY BEEN CONVERTED, DDNAME=SYSUT2
DSNJ261I  DSNJCNVB BSDS CONVERSION FOR DDNAME=SYSUT1 WAS NOT SUCCESSFUL
DSNJ261I  DSNJCNVB BSDS CONVERSION FOR DDNAME=SYSUT2 WAS NOT SUCCESSFUL
DSNJ201I  DSNJCNVB CONVERT BSDS UTILITY PROCESSING WAS UNSUCCESSFUL
```

12.1.5 Plans and packages

If you have old plans and packages from V5 or before, you need to REBIND them or DB2 automatically rebinds at conversion time. (The ABIND DSNZPARM can be used to control the automatic rebind.)

You need to convert plans that contain DBRMs to packages. See PK62876 and PK85833 (DB2 9), PK79925 (V8), and PM01821 (all). DB2 automatically rebinds a DBRM into a package at execution time and chooses a collection name of its own. Verify that this automatic naming is not an issue. Otherwise, you need to convert plants that contain DBRMs yourself.

Additional information: For information about converting plans that contain DBRMs to packages, see *DB2 9 for z/OS: Packages Revisited*, SG24-7688.

REBIND is not required for migration to DB2 10, but REBIND is strongly recommended. Getting the best performance improvements and eliminating regression does depend upon rebind in most situations (for example getting current structures, better access paths, and reusing threads). Eliminating performance regression might depend upon REBIND. Storage constraint relief depends upon REBIND. Changing to use release deallocate requires a REBIND. You can stage other REBINDs over weeks of time, and REBIND is needed only once per package for the migration.

Improvements in access paths can be significant, such as stage 2 predicates that can become stage 1. REBIND in DB2 10 takes more CPU and elapsed time than in prior versions, but more concurrent REBINDs are possible in NFM.

12.2 Some release incompatibilities

For details about application and SQL release incompatibilities from either DB2 V8 or DB2 9 to DB2 10, see Chapter 2."Preparing your system to install or migrate DB2" of the *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974.

In this section we mention a few that caused maintenance to be planned.

12.2.1 CHAR and VARCHAR formatting for decimal data

In DB2 10 for z/OS, the formatting of decimal data has changed for the CHAR and VARCHAR functions and CAST specifications with a CHAR or VARCHAR result type. When the input data is decimal, any leading zeroes in the input value are removed, and leading zeroes are not added to an input value that did not already contain leading zeroes.

This implies the different behavior as detailed in Example 12-6.

Example 12-6 CHAR casting in DB2 9 NFM and DB2 10 CM

```
CREATE TABLE RGTBXXX (COL1 DEC(004 , 000));
INSERT INTO RGTBXXX VALUES(9);
INSERT INTO RGTBXXX VALUES(99);
INSERT INTO RGTBXXX VALUES(999);
INSERT INTO RGTBXXX VALUES(9999);

Query:
SELECT SUBSTR(CHAR(COL1), 2 , 4)
,HEX(SUBSTR(CHAR(COL1), 2 , 4))
FROM RGTBXXX ;
.
output in V9 NFM:
0009 F0F0F0F9
0099 F0F0F9F9
0999 F0F9F9F9
9999 F9F9F9F9
.
output in V10 CM:
    40404040
9   F9404040
99  F9F94040
999 F9F9F940
```

The reason for this incompatible change is to make DB2 for z/OS behavior more consistent with the SQL standard and to enhance the compatibility throughout the DB2 family.

See APAR PM29124, currently open, for possible relief in this area.

12.2.2 Fall back restriction for native SQL procedures

Native SQL procedures created or regenerated in DB2 10 conversion mode cannot run in DB2 9, either in coexistence or after fallback. You should avoid creating or regenerating procedures until you are sure of not falling back or keep track of and regenerate these procedures when falling back to DB2 9.

APAR PM13525, currently open, is planned to provide an implicit auto regeneration for native SQL procedures.

12.2.3 Fall back restriction for index on expression

Indexes on expressions created or regenerated in DB2 10 conversion mode cannot run in DB2 9, either in coexistence or after fallback. You should avoid creating or regenerating indexes on expressions until you are sure of not falling back or keep track of and regenerate these indexes when falling back to DB2 9.

APAR PM13631, currently open, is planned to provide an implicit auto regeneration for index on expressions.

12.3 DB2 10 product packaging

DB2 10 for z/OS incorporates or offers separately several features, which include tools for data warehouse management, data connectivity, database management and tuning, installation, and capacity planning. These features and tools work directly with DB2 applications to help you use the full potential of your DB2 system. When ordering the DB2 base product, you can select the chargeable and nonchargeable DB2 features to be included in the package.

For current and correct information about the contents of DB2 10 for z/OS packaging, check the program directories and the manuals. For more information, see “Related publications” on page 641.

IBM added many data server capabilities in DB2 10 for z/OS and reduced or removed support for some functions. In the next sections we discuss the following topics:

- Removed features and functions
- Deprecated features and functions
- Base engine and features

12.3.1 Removed features and functions

As you prepare to upgrade subsystems to DB2 10, be aware of the changes that we list in this section.

For both DB2 V8 and DB2 9

Be aware of the following changes for both DB2 UDB for z/OS Version 8 and DB2 9 for z/OS:

- DB2 Client Management feature is no longer available.
- The Control Center does not support connections to DB2 10.
- Private protocol is no longer supported. Convert to DRDA. See job DSNTP2DP and APARs PK92339 andPK64045.[1] See 12.14, “Eliminating DDF private protocol” on page 529.
- DB2 catalog tables are DB2-managed and SMS-managed. Catalog and directory tables no longer have links but have more LOBs and more table spaces. Compression for table space SPT01 is not supported. See 12.5, “Catalog changes” on page 489.
- DB2 MQ XML functions are no longer supported.
- msys for Setup DB2 Customization Center is no longer supported. Use installation panels instead.
- DB2 XML Extender is no longer supported. Use pureXML.
- REORG TABLESPACE SHRLEVEL NONE on LOB table spaces is removed. Use SHRLEVEL CHANGE or REFERENCE.
- Several DSNZPARMs are removed, no longer supported or changed defaults. See 12.7, “DSNZPARM change summary” on page 501.
- Database metadata routines and supporting objects are converted from EBCDIC to Unicode. DSNTIJRT in CM8 or CM9 checks for EBCDIC-encoded metadata objects, drops them, and recreates them as Unicode-encoded. Makes sure that authority is regranted as before this conversion. A premigration query in DSNTIJPM checks for this occurrence.
- The DSNxxx (password encryption) stored procedure is modified. The PASSWORD parameter is changed from CHAR(8) to VARCHAR(100). After DB2 enters NFM, when you run DSNTIJRT, it drops and re-creates this routine. A premigration query in DSNTIJPM checks for this occurrence.

For DB2 UDB for z/OS Version 8 only

Be aware of the following changes for DB2 UDB Version 8:

- Net.Data® is removed. WebSphere is the strategic IBM solution for delivering DB2 data to web applications.
- DB2-established stored procedure address spaces are no longer supported. Workload Manager (WLM) managed stored procedure address spaces is the strategic solution for stored procedure support, and migration to WLM managed stored procedure spaces is required for use of stored procedures in DB210.

[1] Details in *DB2 9 for z/OS: Distributed Functions*, SG24-6952.

Archived

- JDBC/SQLJ Driver for OS/390® and z/OS is no longer supported. All Java application programs and Java routines that are currently written to work with the JDBC/SQLJ Driver for OS/390 and z/OS need to be modified to work with the IBM DB2 Driver for JDBC and SQLJ (formerly known as the *DB2 Universal JDBC Driver*).

  You can find information about how to migrate JDBC and SQLJ applications from the earlier JDBC/SQLJ Driver for OS/390 and z/OS to the IBM DB2 Driver for JDBC and SQLJ in *DB2 10 for z/OS Application Programming and SQL Guide,* SC19-2969, and *DB2 10 for z/OS Application Programming Guide and Reference for Java,* SC19-2970. In addition, all WLM-managed stored procedures address spaces that are set up to execute Java routines must be modified to reference the IBM DB2 Driver for JDBC and SQLJ.
- Connections from VAX machines and the PASCAL L string data type are no longer supported.
- Creation of simple table spaces is no longer supported. DB2 10 for z/OS no longer implicitly creates simple table spaces nor allows customers to create simple table spaces. However, DB2 10 for z/OS continues to support simple table spaces created in previous versions.
- DB2 QMF™ Visionary Studio program is removed from DB2 QMF Enterprise Edition.
- DB2 Estimator is not available for DB2 10.
- BookManager-based online help is removed. The prior help support is replaced by the Information Management Software for z/OS Solutions Information Center (Information Center). The web-based information center is updated periodically during the life of each DB2 version, thus ensuring access to the most up-to-date information.

  For more information, go to:

  `http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db29.doc/db2prodhome.htm`
- AIV Extender, Text Extender, and Net Search Extender are removed.
- Java stored procedures are no longer run in resettable JVMs.

For DB2 9 for z/OS only

Be aware of the following changes for DB2 9:

- Annotated XML schema decomposition using XDBDECOMPXML and XDBDECOMPXML100MB is no longer supported. These procedures are dropped when you run job DSNTIJRT after DB2 enters NFM.
- The XSR user defined function SYSFUN.DSN_XMLVALIDATE is converted to the built-in SYSIBM.DSN_XMLVALIDATE function. After DB2 enters NFM, DSNTIJRT performs a dummy alter of the UDF to invalidate packages that use the UDF. When those packages are rebound they will use the BIF instead of the UDF.
- Accessories Suite no longer includes Optimization Service Center component and the Data Studio Workbench feature. Visual Explain for DB2 for z/OS is not available for DB2 10. The recommended query optimization and service tools for DB2 for z/OS are Optim Query Tuner and Optim Query Workload Tuner. The recommended no-charge query optimization and service tool for DB2 for z/OS is Data Studio.

  For more information, go to:

  `http://www.ibm.com/software/data/studio/`

  These tools are based and built on the foundation of Optimization Service Center and Optimization Expert.

Also consult the following developerWorks® article, *Understanding the packaging of Optim Solutions for database development, administration, and performance management*:

http://www.ibm.com/developerworks/data/library/techarticle/dm-0909optimpackaging/?S_TACT=105AGX01&S_CMP=LP

In addition, refer to recent announcements for DB2 10 for z/OS support.

12.3.2 Deprecated features and functions

As part of ongoing efforts to deliver the most current technology and to remove features that no longer provide strategic benefits, the following features are deprecated in DB2 10, which means that these functions might be dropped in the future. Do not create any new dependencies that rely on these function. If you have existing dependencies on these functions, plan to remove these dependencies.

- Simple and partitioned table spaces other than universal table spaces (UTS) are deprecated. Use alter in new-function mode to convert single-table table space to universal.
- UTS is strategic and index controlled partitioning is not permitted for UTS. Consider changing from index-controlled partitioning to table-controlled partitioning as the first step towards UTS.
- Consider changing REORG of LOB table spaces to an option other than SHRLEVEL NONE.
- Some current use of DSNHDECP is deprecated. If you have code that loads DSNHDECP and maps it with macros, plan to change that code using the new techniques.[2] If you want to have one library for multiple DSNHDECP modules, you need to make this change. User-chosen names for the DB2 application defaults (DSNHDECP) and the DB2 auth-exit modules can now be applied.
- SQL processing options NEWFUN(YES) and NEWFUN(NO) options are deprecated. Use NEWFUN(DB2 10) rather than NEWFUN(YES). Use NEWFUN(DB2 9) or NEWFUN(DB2 V8) rather than NEWFUN(YES). Use NEWFUN(DB2 9) or NEWFUN(DB2 V8) rather than NEWFUN(NO).
- The DSNHPC7 precompiler is deprecated. Use the current precompiler or coprocessor.

[2] Use the DECP address that is returned by IFICD 373, DSNALI, or DSNRLI, which confirms that you are using the same DECP module that was used to start DB2.

Archived

12.3.3 Base engine and features

In this section, we list DB2 10 base and optional features, as summarized in Figure 12-4.

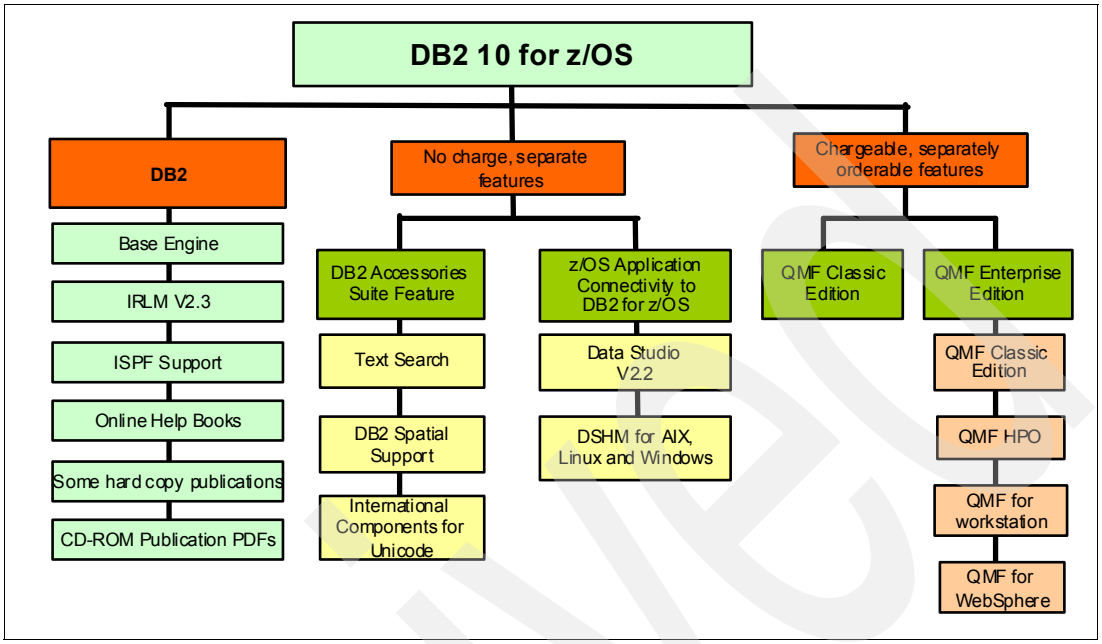


Figure 12-4 DB2 10 optional features

DB2 10 for z/OS

DB2 10 for z/OS, program number 5605-DB2, currently consists of the following FMIDs:

- Required FMIDs
  - HDBAA10 (contains DB2 Base, REXX, MQSeries®, MQListener)
  - HIYAA10 (IMS Attach - must be installed even if you do not have IMS)
  - HIZAA10 (Subsystem Initialization)
  - HIR2230 (IRLM V02.03.00)
  - HDREA10 (DB2 RACF Authorization Exit)
  - JDBAA14 (DB2 English Panels)
- Optional FMIDs
  - JDBAA12 (DB2 JDBC/SQLJ)
  - JDBAA17 (DB2 ODBC)
  - JDBAA11 (DB2 Kanji Panels, available at General Availability)

Note that DB2 Utilities Suite for z/OS Version 10 (program number 5655-V41) contains all of the IBM DB2 Utilities. Utilities are included with restricted function in the base product. They can be used on DB2 catalog and directory data and the DB2 IVP sample database. A valid separate product licence must be obtained to activate the utilities function on user data.

For details, see *Program Directory for DB2 10 for z/OS*, GI10-8829.

DB2 optional products

In this section, we describe DB2 product features that offer complementary functions to DB2 functions.

DB2 10 for z/OS orderable no-charge features

DB2 10 for z/OS provides the following orderable no-charge features:

- z/OS Application Connectivity to DB2 for z/OS consists of a component known as the DB2 Universal Database™ Driver for z/OS, Java Edition, a pure Java, type 4 JDBC driver designed to deliver high performance and scalable remote connectivity for z/OS Java-based enterprise applications on z/OS to a remote DB2 for z/OS database server. For details, see *z/OS Application Connectivity to DB2 for z/OS*, GI10-8830.
- DB2 Accessories Suite components are designed to enhance your use of the DB2 for z/OS data server. In this release of the suite, two components deliver innovative capabilities for your DB2 for z/OS environment. It includes Spatial Support for DB2 for z/OS and IBM Text Search for DB2 for z/OS, including Unicode and Internationalization support. For details, see *Program Directory for DB2 Accessories Suite for DB 10 for z/OS*, GI10-8841.
  - Spatial Support for DB2 for z/OS provides a set of spatial data types that you can use to model real-world entities, such as the locations of customers, the boundaries of parks, and the path of cable lines. You can manipulate spatial data through spatial functions, which can be invoked from within an SQL statement. You can create indexes on the spatial data, and DB2 can use them to optimize spatial query performance.
  - IBM OmniFind® Text Search Server for DB2 for z/OS allows users to issue SQL statements to satisfy familiar text search queries on data that is stored in a DB2 for z/OS database. The IBM Text Search for DB2 for z/OS feature is powered by an IBM OmniFind Text Search Server for DB2 for z/OS and includes Linux on System z server support.

    Omnifind Text Search Server (on three CD-ROMs for Linux, Windows, and Linux on System z) has an operational requisite for any of the following products:

    - 32-bit Red Hat Enterprise Linux, Version 5 (with the latest update)
    - 32-bit SUSE Linux Enterprise 10
    - 64-bit SUSE Linux on System z (SLES10x)
    - 64-bit Red Hat Linux on System z, Version 5 (with the latest update)
    - 32-bit Windows Server 2003 (with the latest service pack)
  - International Components for Unicode (ICU), a prerequisite for Text Search.

IBM Data Studio

IBM Data Studio is available at no additional cost as a *separate* download. We list it here because it provides functions that are available in previous no-charge features. It provides a complete development and testing environment for building database objects and queries and for completing data object management tasks, such as building and testing stored procedures (Java and SQL) with the interactive routine debugger. IBM Data Studio supports query tuning features for DB2 for z/OS and replaces Optimization Service Center for DB2 for z/OS and Visual Explain.

For more information, see:

`http://www.ibm.com/software/data/optim/data-studio/`

DB2 10 for z/OS chargeable feature

An optional chargeable feature with DB2 10 includes Query Management Facility™.

DB2 Query Management Facility

IBM DB2 Query Management Facility (QMF) 10 for z/OS is a separate charge product that includes a comprehensive suite of features, starting with the base product and extending into new depths for business intelligence and analytics. DB2 QMF offers the following editions:

- QMF Classic Edition

  Query, reporting, graphics, procedures, batch processing, and more. The Classic Edition is designed to deliver enterprise-wide query and reporting in an efficient and scalable architecture.

- QMF Enterprise Edition

  DB2 QMF Enterprise Edition provides the entire DB2 QMF family of technologies, enabling enterprise-wide business information across user and database platforms. DB2 QMF Enterprise Edition consists of the following components:

  – DB2 QMF Classic Edition
  – DB2 QMF High Performance Option (HPO)
  – DB2 QMF for Workstation
  – DB2 QMF for WebSphere

DB2 10 for z/OS separate charge product: IBM DB2 Utilities Suite

The IBM DB2 Utilities Suite is orderable separately as an one-time charge (OTC) product and is a comprehensive set of tools for managing all DB2 data maintenance tasks. DB2 Utilities Suite helps you to minimize downtime that is associated with routine DB2 data maintenance while ensuring the highest degree of data integrity.

Version 10 of DB2 Utilities Suite for z/OS (program number: 5655-V41) supports all major new functions and structural changes in the DB2 10 for z/OS product.

For details, see *DB2 Utilities Suite for z/OS, V10, Program Directory*, GI10-8840.

12.4 Command changes

In this section, we list a summary of DB2 commands changes. For details about command changes, see *DB2 10 for z/OS Command Reference,* SC19-2972.

- -ALTER BUFFERPOOL

  The PGSTEAL NONE option is new. It specifies that no page stealing is to take place, Data that is brought in stays resident in the buffer pool until the object is physically closed. This is “page fixing” the buffer pool.

- -ALTER UTILITY

  For REORG and REBUILD, the DATAACCESS authority and system DBADM authority can issue this command.

- BIND PACKAGE

  To specify the EXPLAIN(ONLY) and SQLERROR(CHECK) options, the binder must have the BIND, BINDAGENT, or EXPLAIN privilege. The system DBADM authority has authority to run BIND PACKAGE.

Archived

- BIND QUERY (DSN) is new.
- BIND and REBIND changes:
  - ACQUIRE(ALLOCATE) is removed, and DB2 uses ACQUIRE(USE).
  - The CONCURRENTACCESSRESOLUTION and DSNZAPRM SKIPUNCI are new.
  - For DBPROTOCOL, the only valid value is now DRDA (elimination of private protocol), and you can no longer code PRIVATE.
  - You can now use DEPLOY to also deploy non-inline SQL function.
  - The EXTENDEDINDICATOR keyword determines whether DB2 recognizes extended indicator variables when the associated package is run.
  - The LOOKUP keyword determines if a query has matching access plan hint information in the SYSIBM.SYSQUERYPLAN table.
  - The PLANMGMT option value of ON is used to regenerate and save access paths for static SQL and purge previous or original copies.
  - The PLANMGMTSCOPE keyword is used for plan stability management.
  - The system DBADM authority can issue the -CANCEL THREAD (DB2) command, which can be used to cancel accelerated threads.
- -DISPLAY DATABASE output now shows new AREOR status for pending definition changes.
- The system DBADM authority can issue the following commands:
  - -DISPLAY DDF
  - -DISPLAY FUNCTION SPECIFIC
  - -DISPLAY GROUP
  - -DISPLAY GROUPBUFFERPOOL
  - -DISPLAY LOCATION
  - -DISPLAY LOG (also displays the new setting for when/how checkpoints are taken)
  - -DISPLAY PROCEDURE
  - -DISPLAY PROFILE
  - -DISPLAY RLIMIT
  - REBIND PACKAGE (DSN)
  - REBIND PLAN (DSN)
  - REBIND TRIGGER PACKAGE (DSN)
  - -RECOVER INDOUBT (DB2)
  - -RECOVER POSTPONED (DB2)
  - -START DATABASE (DB2)
  - -START DDF (DB2)
  - -START FUNCTION SPECIFIC (DB2)
  - -START PROCEDURE (DB2)
  - -START PROFILE (DB2)
  - -START RLIMIT (DB2)
  - -STOP DATABASE (DB2)
  - -STOP DB2 (DB2)
  - -STOP DDF (DB2)
  - -STOP FUNCTION SPECIFIC (DB2)
  - -STOP PROCEDURE (DB2)
  - -STOP PROFILE (DB2)
  - -STOP RLIMIT (DB2)

Archived

- -DISPLAY THREAD, the new ACCEL() keyword, limits the list to show only threads with active accelerator processes that execute within the specified accelerator name. The system DBADM authority can issue this command. The output shows the accelerator server name and the IP address or domain of the accelerator server.
- -DISPLAY TRACE
  - The system DBADM authority and SECADM authority can issue this command.
  - The ASID() keyword filters trace-events based on the address space in which the events occurred.
  - The AUDTPLCY() keyword specifies the audit traces to display.
- The system DBADM authority can issue the following DSN subcommands:
  - FREE PACKAGE (DSN)
  - FREE PLAN (DSN)
- FREE QUERY (DSN) removes one or more queries from the access path repository.
- -MODIFY DDF (DB2) modifies information regarding the status and configuration of DDF and statistical information regarding connections or threads controlled by DDF.
- -MODIFY TRACE (DB2)

  The system DBADM authority and SECADM authority can issue this command.
- REBIND QUERY (DSN) marks one or more dynamic SQL queries for reoptimization. Those queries are reoptimized the next time they are submitted for a full prepare operation.
- -SET LOG (DSN) changes in support of the SINGLE or BOTH parameter to specify when to take a checkpoint. The -DISPLAY LOG output has also changed to show the setting.
- -SET SYSPARM (DB2)

  The System SECADM authority can issue this command.
- -START DB2 (DB2), DECP() keyword, specifies the name of the load module that contains the DB2 application parameter default.
- -START TRACE (DB2)
  - System DBADM authority and SECADM authority can issue this command.
  - The ASID() keyword to filter trace-events based on the address space in which the events occur.
  - The TDATA sub keyword STATEMENT places a statement header on the record
  - The TDATA sub keyword AUDTPLCY() lists up to eight audit policy names for which trace information is to be gathered.
- -STOP TRACE (DB2)
  - System DBADM authority and SECADM authority can issue this command.
  - The ASID() keyword filters trace-events based on the address space in which the events occurred.
  - The AUDTPLCY() keyword specifies which audit traces to display.
- -TERM UTILITY (DB2)

  System DBADM authority and DATAACCESS authority can issue this command.

Archived

12.5 Catalog changes

Figure 12-5 shows how the DB2 catalog continues to grow from release to release. These numbers are the numbers for DB2 10 as of the writing of this book and do not include the objects for XSR support.

DB2 Version	Table spaces	Tables	Indexes	Columns	Table check constraints
V1	11	25	27	269	N/A
V3	11	43	44	584	N/A
V5	12	54	62	731	46
V6	15	65	93	987	59
V7	20	84	117	1212	105
V8	21	87	128	1286	105
DB2 9	28	106	151	1668	119
DB2 10	**81 (88-7)**	**124**	**195**	**1701**	**119**

Figure 12-5 DB2 catalog evolution

In this section, we also discuss the following items that are related to the restructuring of the DB2 catalog in DB2 10:

- DB2 now SMS manage catalog and directory data sets
- Define new CLOB/BLOB columns in the catalog

  Merge records that store SQL statement text
- Reduce catalog contention
  - Removal of links
  - Change to row-level locking
- Convert some catalog and directory table spaces to partition-by-growth (with MAXPARTS = 1)
- Combine SYSUTIL and SYSUTILX into a single table SYSUTILX

The list of changes during DSNTIJTC to migrate to DB2 10 CM9 includes 25 new columns, 22 new index, two new table spaces, three new tables, and no change in check constraints or any new check constraints.

During the ENFM, DB2 creates eight additional new column, 11 new LOB table spaces, 11 new aux tables, 11 new auxiliary indexes, 47 new table spaces, four new tables, 11 new indexes. In addition, seven old table spaces are dropped.

If your applications do not use LOBs today and if you are not accustomed to LOBs, be aware that DB2 10 catalog now has LOB objects, which implies catalog recovery procedures changes and disaster recovery testing.

Archive

12.5.1 SMS-managed DB2 catalog and directory data sets

By DB2 using SMS-managed DB2 catalog and directory data sets, we obtain the following advantages:

- Minimize user's effort to maintain data sets.
  - No need to allocate data sets and extended pieces (A002, A003, and so forth).
  - No need to allocate data sets as part of the migration to the next release.

    If you are migrating to DB2 10, DSNTIJIN is needed only to allocate the DB2 10 sample libraries (prefix.RUNLIB.LOAD, prefix.DBRMLIB.DATA, and prefix.SRCLIB.DATA). DSNTIJIN is customized accordingly when you run the install CLIST in MIGRATE mode. When you are installing a new DB2, the full-blown DSNTIJIN is needed to allocate the initial data sets for the catalog and directory, and the data sets for the BSDS and active logs.
  - No need to allocate data sets for the shadow for online REORG.
  - No need to estimate the size for the data sets (had to provide space allocations in the DEFINEs).
    - DB2 uses a default for the primary and a sliding scale for secondary.
    - Minimize the chance of running out of extents before reaching the maximum data set size.
  - SMS determines which data set goes to which volume.
  - Minimize outage due to improper data set allocations.
- New fields in installation and migration panels (CLIST).
  - SMS information (data class, storage class, and management class) stored in ZPARM.
- In all modes of DB2 10, all new data sets for the catalog and directory are defined by DB2 and must be SMS managed. Data sets for existing catalog and directory table spaces and indexes do not need to be converted to SMS prior to migration and can remain non-SMS managed indefinitely in DB2 10. Such data sets are converted to SMS managed the first time you reorganize the associated table space.

12.5.2 CLOB and BLOB columns added to the catalog

The use of LOBs in the catalog relieves the limit of 64 GB per page set and overcomes the limit of 32 KB per row, as follows:

- DB2 10 relieves SPT01 package space problems. (SPT01 is still limited to 64 GB, but LOBs extend the space.)
  - When the DB2 system is in DB2 10 NFM, SPT01 is a partition-by-growth, DSSIZE 64 GB, MAXPART 1.
  - BLOB and CLOB columns are used for package data.

  Note that application developers can create packages using the following methods:
  - Using BIND PACKAGE
  - Using CREATE TRIGGER
  - Creating SQL procedures
  - Binding a package using package stability

- CLOB columns are introduced in the catalog because some catalog tables have columns that store possibly large SQL statements.
  - SYSINDEXES, SYSPACKSTMT, SYSVIEWS, SYSTRIGGERS, database descriptors (DBDs, in the directory), SPT01 (directory)
  - Not easy to scan the SQL statements
  - Statements are split into multiple records with different sequence numbers associated to each record
  - Possibly a mixture of UNICODE and EBCDIC SQL statements are stored in the existing FOR BIT DATA column in SYSPACKSTMT
- Adding CLOB columns to the catalog to store the SQL statements provides the following benefits:
  - Implicit UNICODE/EBCDIC conversion is handled by CLOB.
  - Merge all the split records that store SQL statements into the new LOB columns. The old text columns are cleared and are no longer used.
  - Easier to read the statements and perform searches using SQL.
  - Inline LOBs for SYSPKSTMT and SYSVIEWS are used.

  The old columns remain, but the data is moved to the new columns during ENFM processing.
- New BLOB columns are needed because there are catalog or directory tables that have columns that store large amount of binary structures (packages and DBDs). The BLOB columns store the packages and DBDs because LOB table spaces can hold much more data.

12.5.3 Reduced catalog contention

Prior to DB2 10, the DB2 catalog makes use of links to help with performance when following the structure of the metadata. With more concurrent updates, the links can cause the following issues:

- Contention between DDL and BIND
- Do not allow row-level locking
- Contention between BIND and utilities
- New releases require a rebind of old packages, which can affect more users

These issues generally result in the application programmer receiving deadlocks when trying to BIND or PREPARE packages in parallel. The production DBA might be asked to diagnose why an application or utility fails with a deadlock.

A system programmer can receive deadlocks when trying to run utilities at the same time as BIND or PREPAREs.

The SYSDBASE, SYSPLAN, SYSDBAUT, SYSVIEW, SYSGROUP, and DBD01 table spaces have links and use page level locking because of the links.

With DB2 10 NFM, these table spaces are removed and the tables within are moved to new table spaces that are defined as follows:

- Row level locking
- Reordered row format
- Partition-by-growth
- One table per table space

The new definitions provide the following benefits:

- Row level locking minimizes timeout or deadlock for the following concurrent catalog access:
  - DDL
  - DML
  - Grant/revoke
  - Commands (DISPLAY, BIND, REBIND, FREE, and other commands)
  - Utilities (for example RUNSTATS)
- There is less or no SQL timeout when running BIND at the same time, therefore increased availability for your applications.
- DB2 can allocate new data sets when space is required instead of failing after waiting for operator intervention.
- Reduced catalog contention in case of the following situations:
  - Concurrent binding and running of DDL
  - Multiple binds in parallel

12.5.4 Converting catalog and directory table spaces to partition-by-growth

Table 12-2 lists the catalog tables that are moved to new partition-by-growth table spaces with MAXPARTS=1. Each converted table now has its own partition-by-growth table space. During the conversion of these tables and their associated indexes, DB2 retains the tables' original OBIDs and index OBIDs/PSIDs. That is, the OBID of the tables in the new table spaces remain the same as prior to the move, and the OBIDs/PSIDs of the indexes for these tables remain the same as prior to the move.

These partition-by-growth tables spaces for the catalog are defined as UNICODE, and SYSTABLES.TSNAME is updated to reflect the table space names. All the new table spaces have the following attributes unless otherwise specified:

- Row-level locking
- Reordered-row format (RRF)
- Partition-by-growth
- DSSIZE 64G MAXPART 1
- LOG YES
- SEGSIZE 32
- CCSID UNICODE
- MEMBER CLUSTER NO
- CISIZE 4096 (or page size)
- PRIQTY and SECQTY -1 (sliding scale of secondary extent allocation)

The new table spaces inherit the following attributes from the original table spaces:

- TRACKMOD
- LOGGED
- MAXROWS
- GBPCACHE
- FREEPAGE
- PCTFREE
- CLOSE

Archived

Table 12-2 Catalog and directory tables and their partition-by-growth table spaces

Table name	Old table space	New partition-by-growth table space
SYSCOLAUTH	SYSDBASE	SYSTSFAU
SYSCOLUMNS	SYSDBASE	SYSTSCOL
SYSFIELDS	SYSDBASE	SYSTSFLD
SYSFOREIGNKEYS	SYSDBASE	SYSTSFOR
SYSINDEXES	SYSDBASE	SYSTSIXS
SYSINDEXPART	SYSDBASE	SYSTSIPT
SYSKEYS	SYSDBASE	SYSTSKEY
SYSRELS	SYSDBASE	SYSTSREL
SYSSYNONYMS	SYSDBASE	SYSTSSYN
SYSTABAUTH	SYSDBASE	SYSTSTAU
SYSTABLEPART	SYSDBASE	SYSTSTPT
SYSTABLES	SYSDBASE	SYSTSTAB
SYSTABLESPACE	SYSDBASE	SYSTSTSP
SYSDATABASE	SYSDBAUT	SYSTSDBA
SYSSTOGROUP	SYSGROUP	SYSTSSTG
SYSAUXRELS	SYSOBJ	SYSTSAUX
SYSCONSTDEP	SYSOBJ	SYSTSCON
SYSDATATYPES	SYSOBJ	SYSTSDAT
SYSDEPENDENCIES	SYSOBJ	SYSTSDEP
SYSENVIRONMENT	SYSOBJ	SYSTSENV
SYSKEYCOLUSE	SYSOBJ	SYSTSKYC
SYSPARMS	SYSOBJ	SYSTSPRM
SYSROUTINEAUTH	SYSOBJ	SYSTSRAU
SYSROUTINES	SYSOBJ	SYSTSROU
SYSSCHEMAAUTH	SYSOBJ	STSTSSCM
SYSTABCONST	SYSOBJ	SYSTSTBC
SYSSTRIGGERS	SYSOBJ	SYSTSTRG
SYSPACKAGE	SYSPACKAGE	SYSTSPKG
SYSPACKAUTH	SYSPACKAGE	SYSTSPKA
SYSPACKDEP	SYSPACKAGE	SYSTSPKD
SYSPACKLIST	SYSPACKAGE	SYSTSPKL
SYSPACKSTMT	SYSPACKAGE	SYSTSPKS

Table name	Old table space	New partition-by-growth table space
SYSPKSYSTEM	SYSPACKAGE	SYSTSPKY
SYSPLSYSTEM	SYSPKAGE	SYSTSPLY
SYSDBRM	SYSPLAN	SYSTSDBR
SYSPLAN	SYSPLAN	SYSTSPLN
SYSPLANAUTH	SYSPLAN	SYSTSPLA
SYSPLANDEP	SYSPLAN	SYSTSPLD
SYSSTMT	SYSPLAN	SYSTSSTM
SYSVOLUMES	SYSSTOGROUP	SYSTSVOL
SYSVIEWDEP	SYSVIEWS	SYSTSVWD
SYSVIEWS	SYSVIEWS	SYSTSVIW
SYSVTREE	SYSVIEWS	
SYSVLTREE	SYSVIEWS	
DBDS (directory)	DBD01	DBD01
SPTR (directory)	SPT01	SPT01
SYSUTIL (directory)	SYSUTILX	SYSUTILX
SYSUTILX (directory)	SYSUTILX	

The REORG step in job DSNTIJEN unloads tables from the following old table spaces to the new partition-by-growth table spaces:

- SYSDBASE
- SYSDBAUT
- SYSGROUP
- SYSPKAGE
- SYSPLAN
- SYSOBJ
- SYSVIEW
- DBD01
- SPT01

Tables SYSVTREE and SYSVLTREE no longer exist after SYSVIEWS completes ENFM processing. Table SYSUTILX (directory) no longer exists after SYSUTILX table space completes ENFM processing.

Figure 12-6 shows the changes made to the DB2 directory.

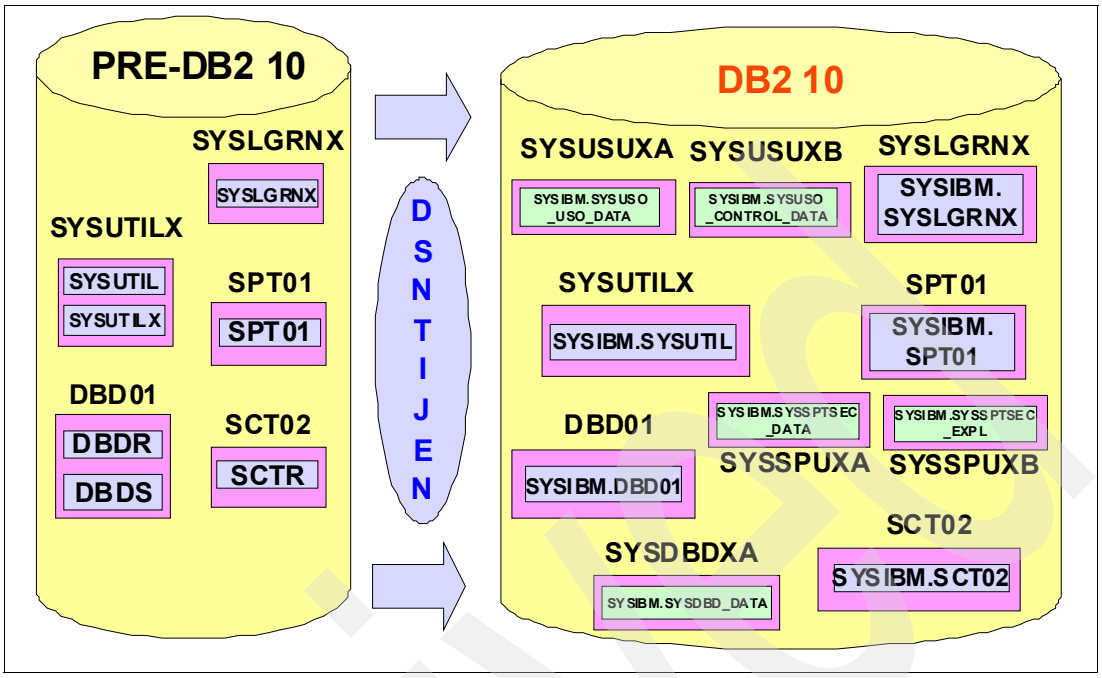


Figure 12-6 DB2 directory table changes

DB2 10 includes the following changes to the directory tables. These tables are for DB2 use only.

- SYSIBM.SYSUTIL

  The existing SYSUTIL and SYSUTILX tables in the DSNDB01.SYSUTILX table space are combined and redefined as SYSIBM.SYSUTIL, and this table is populated by the DSNTIJEN job.

- New directory table SYSIBM.SYSUSO_USO_DATA in the new DSNDB01.SYSUSUXA directory table space
- New directory table SYSIBM.SYSUSO_CONTROL_DATA in the new DSNDB01.SYSUSUXB directory table space
- SYSIBM.SCT02

  The existing table SCTR in the DSNDB01.SCT02 is redefined as SYSIBM.SCT02, and this table is populated by the DSNTIJEN job.

- SYSIBM.DBD01

  The existing tables DBDR and DBDS in the DSNDB01.DBD01 is redefined as SYSIBM.DBD01, and this table is populated by the DSNTIJEN job.

- New directory table SYSIBM.SYSDBD_DATA in the new DSNDB01.SYSDBDXA directory table space
- SYSIBM.SYSLGRNX

  The existing table SYSLGRNX in DSNDB01.SYSLGRNX is redefined as SYSIBM.SYSLGRNX, and this table is populated by the DSNTIJEN job.

- SYSIBM.SPT01

  The existing table SPT01 in DSNDB01.SPT01 is redefined as SYSIBM.SPT01, and this table is populated by the DSNTIJEN job.

Archive

- New directory table SYSIBM.SYSSPTSEC_DATA in the new DSNDB01.SYSSPUXA directory table space
- New directory table SYSIBM.SYSSPTSEC_EXPL in the new DSNDB01.SYSSPUXB directory table space

12.5.5 Added catalog objects

Catalog objects are added in DB2 10 to allow support of new functions. Tables are added for the following functions:

- Security and auditing
- Pending object changes
- BIND QUERY feature
- Added columns

In this section, we discuss these functions.

Security and auditing

In Chapter 10, “Security” on page 337, we discuss DB2 10 security and auditing functions. Figure 12-7 shows the catalog tables to support the enhancements to security and auditing in DB2 10.

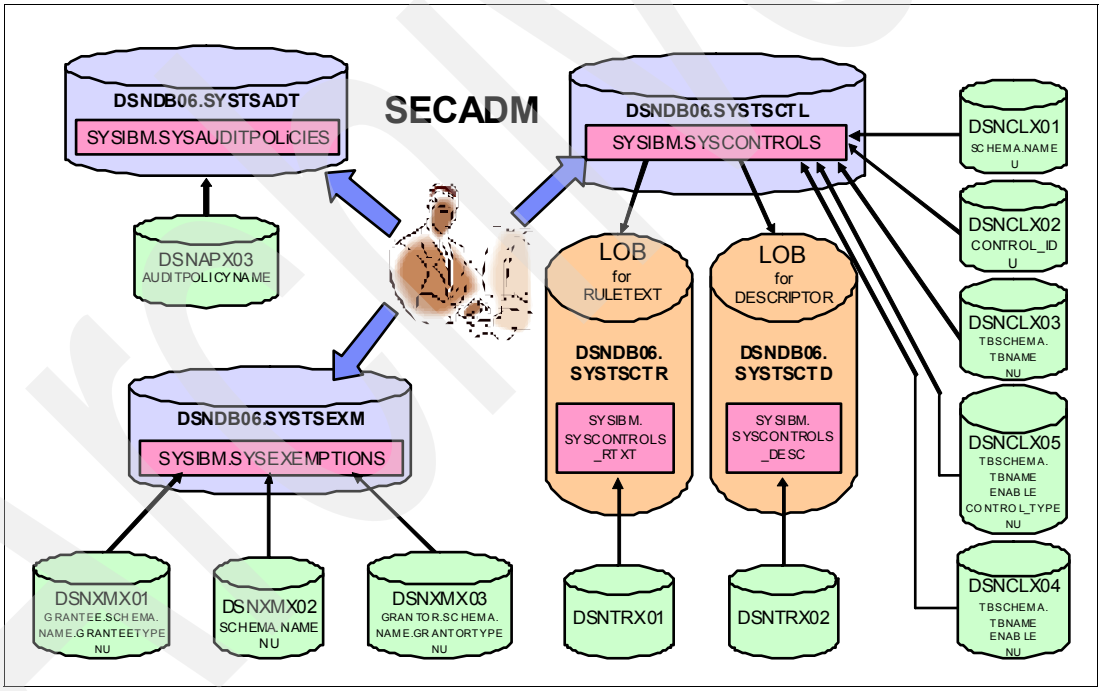


Figure 12-7 Catalog tables for security and auditing in DB2 10

SYSIBM.SYSAUDITPOLICIES in the DSNDB06.SYSTSADT catalog table space includes a row for each audit policy. A user with SECADM authority has the privilege to insert, select, update, or delete from this catalog table. A unique clustering index DSNAPX03 is created on column AUDITPOLICYNAME on this table.

SYSIBM.SYSCONTROLS in the new DSNDB06.SYSTSCTL catalog table space contains one row for each row permission and column mask. This table can be accessed only by SECADM authority (SQLSTATE 42502, SQLCODE -551). The following indexes, in ascending order, are defined:

- DSNCLX01: SCHEMA.NAME (unique)
- DSNCLX02: CONTROL_ID (unique)
- DSNCLX03: TBSCHEMA.TBNAME (non-unique)
- DSNCLX04: TBSCHEMA.TBNAME,ENABLE
- DSNCLX05: TBSCHEMA.TBNAME,ENABLE,CONTROL_TYPE

The following auxiliary tables and indexes are included for LOB table spaces:

- The SYSIBM.SYSCONTROLS_RTXT auxiliary table in the DSNDB06.SYSTSCTR catalog LOB table space is a new AUX table for the RULETEXT column of SYSIBM.SYSCONTROLS and is required to hold the LOB data. The auxiliary index is SYSIBM.DSNTRX01.
- Auxiliary table SYSIBM.SYSCONTROLS_DESC is a new AUX table that is defined in the DSNDB06.SYSTSCTD table space for the DESCRIPTOR column of SYSIBM.SYSCONTROLS and is required to hold the LOB data. The auxiliary index is SYSIBM.DSNTRX02.

Pending object changes

In 4.1, “Online schema enhancements” on page 68, we discuss the online changes and explain the technique used for their implementation requiring catalog registration. Figure 12-8 shows the catalog tables for support of pending object changes enhancement in DB2 10. All indexes are non-unique.

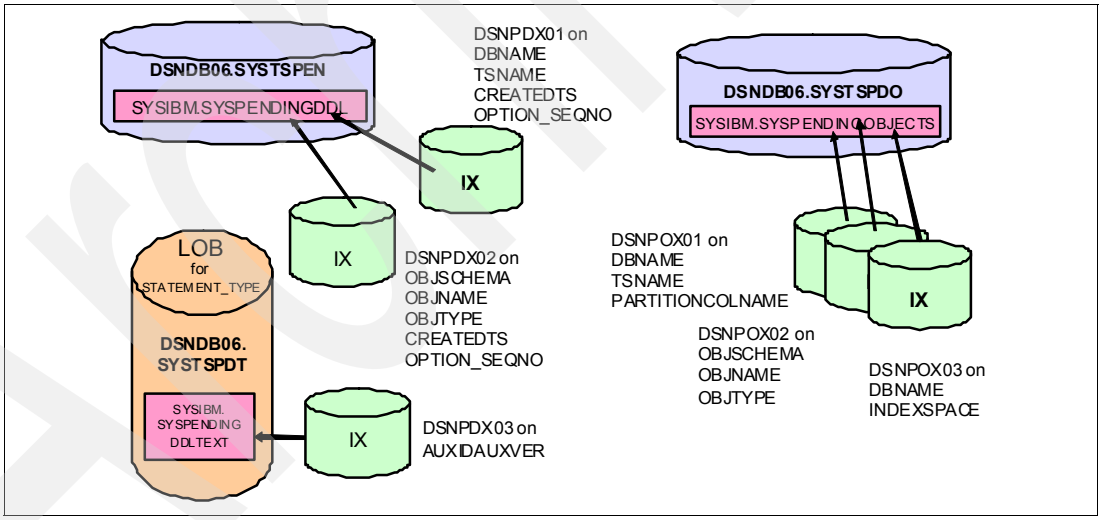


Figure 12-8 Catalog tables for pending object changes in DB2 10

SYSIBM.SYSPENDINGDDL in the DSNDB06.SYSTSPEN catalog partition-by-growth table space contains information about objects that have pending definition changes. Table 4-1 on page 72 shows the columns of the SYSIBM.SYSPENDINGDDL table. The entries exist only during the window between the time the names of the new objects are generated and the time when the catalog definition of the new objects are materialized. Thus, the names can be safeguarded from the other immediate-CREATE objects during the same window.

The SYSPENDINGDDL table has two non-unique indexes.

SYSIBM.SYSPENDINGDDLTEXT in the DSNDB06.SYSTSPDT catalog table space is a new AUX table for the STATEMENT_TYPE column of SYSIBM.SYSPENDINGDDL and is required to hold the LOB data. The SYSPENDINGDDLTEXT table has one index.

SYSIBM.SYSPENDINGOBJECTS in the DSNDB06.SYSTSPDO catalog table space contains name and OBID information about the pending-CREATE objects whose data sets are created but for whom object definition are not materialized to the real catalog.

The SYSPENDINGOBJECTS table has three non-unique indexes.

BIND QUERY feature

We discuss the BIND Query in 7.2, “Access plan stability and instance-based statement hints” on page 220. Figure 12-9 shows the catalog tables in support of the new BIND QUERY feature of DB2 10.

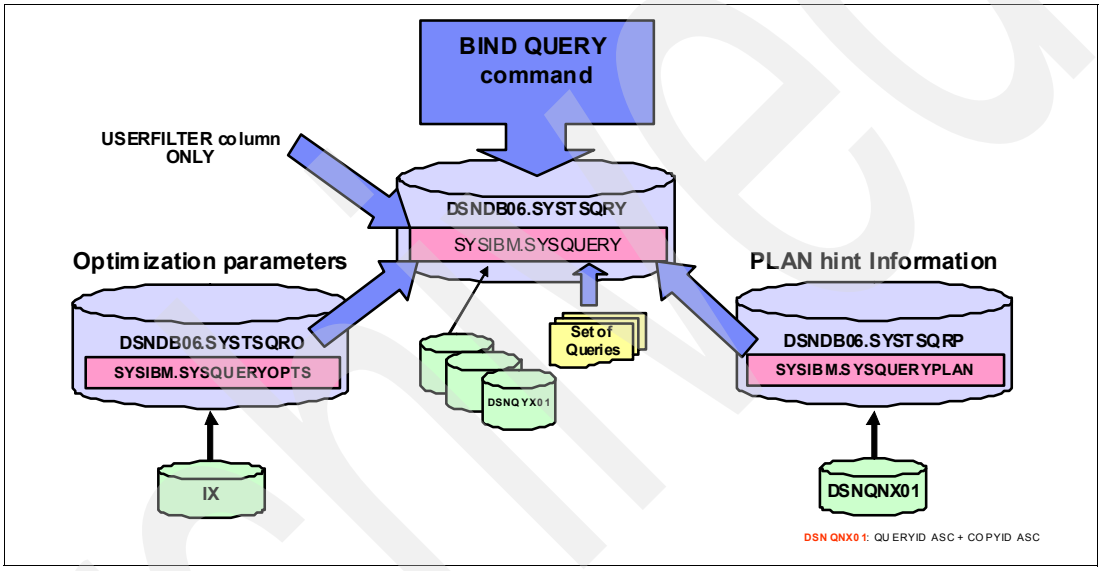


Figure 12-9 Catalog tables for BIND QUERY support in DB2 10

SYSIBM.SYSQUERY in the DSNDB06.SYSTSQRY catalog table space contains a set of queries. The SYSIBM.SYSQUERY table cannot be updated by users except for the USERFILTER column. The SYSIBM.SYSQUERY is populated using a BIND QUERY DB2 command.

The SYSQUERY table has the following indexes:

- DSNQYX01 on QUERY_HASH, SCHEMA, SOURCE, QUERY_SEC_HASH
- DSNQYX02 on QUERYID (unique index)
- DSNQYX03 on LOCATION, COLLECTION, PACKAGE, VERSION, SECTNO

SYSIBM.SYSQUERYPLAN in the DSNDB06.SYSTSQRP catalog table space contains the plan hint information for the queries in the SYSIBM.SYSQUERY table. It correlates to the SYSIBM.SYSQUERY table by QUERYID. For a query, there can be up to three copies of plan hints stored in the SYSQUERYPLAN table for it, which are distinguished by different COPYID values.

The SYSQUERYPLAN table has one unique index DSNQNX01, QUERYID ASC + COPYID ASC.

SYSIBM.SYSQUERYOPTS in the DSNDB06.SYSTSQRO catalog table space contains optimization parameters for the queries in SYSIBM.SYSQUERY. The optimization parameters are enforced on a query at BIND/PREPARE time when PLAN_VALID is blank in SYSQUERY. The SYSQUERYOPTS table has one index.

Added columns

The following tables have added columns to support DB2 10 new features, such as member cluster, hash access, inline lobs, index enhancements, and package related enhancements:

- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSDATATYPES
- SYSIBM.SYSINDEXES
- SYSIBM.SYSINDEXSPACESTATS
- SYSIBM.SYSPACKAGE
- SYSIBM.SYSPACKDEP
- SYSIBM.SYSPLAN
- SYSIBM.SYSROUTINES
- SYSIBM.SYSTABLEPART
- SYSIBM.SYSTABLES
- SYSIBM.SYSTABLESPACE
- SYSIBM.SYSTABLESPACESTATS
- SYSIBM.SYSTRIGGERS
- SYSIBM.SYSUSERAUTH

There are also added values for existing columns, new or changed indexes, and referential constraints.

New MEMBER CLUSTER column

Prior to DB2 10, a “K” or an “I” in the TYPE column of the SYSTABLESPACE catalog table indicated that the table space had MEMBER CLUSTER structure. In DB2 10, a new MEMBER_CLUSTER column on the SYSTABLESPACE catalog table is populated during the enabling-new-function mode (from both DB2 V8 and DB2 9) migration process. For existing MEMBER CLUSTER table spaces, values of “K” in the TYPE column of SYSTABLESPACE are replaced with “L” and values of “I” are replaced with blank.

The new MEMBER_CLUSTER column is populated with “Y” if MEMBER CLUSTER is enabled and is blank if not. After migration to ENFM (from DB2 V8 or DB2 9), applications that query “K” or “I” in the TYPE column must query the new MEMBER_CLUSTER column instead.

Table 12-3 lists additional new tables and table spaces in DB2 10.

Table 12-3 Catalog tables and table spaces in DB2 10

New catalog table	Table space name
SYSINDEXES_RTSECT	SYSTSIXR
SYSINDEXES_TREE	SYSTSIXT
SYSPACKSTMT_STMT	SYSTSPKX
SYSVIEWS_STMT	SYSTSVWT
SYSVIEWS_TREE	SYSTSVTR
SYSTRIGGERS_STMT	SYSTSTRT

New catalog table	Table space name
SYSDUMMYU	SYSTSUNI
SYSDUMMYA	SYSTSASC
SYSDUMMYE	SYSEBCDC (existing)

For a complete description of the columns of the new and changed catalog tables, see *DB2 10 for z/OS SQL Reference,* SC19-2983.

This catalog update includes the following changes:

- Add column DRIVETYPE to table SYSIBM.SYSTABLESPACESTATS
- Add column LPFACILITY to table SYSIBM.SYSTABLESPACESTATS
- Add column STATS01 to table SYSIBM.SYSTABLESPACESTATS
- Add column DRIVETYPE to table SYSIBM.SYSINDEXSPACESTATS
- Add column STATS101 to table SYSIBM.SYSINDEXSPACESTATS
- Alter LOCKSIZE to ROW for SYSCOPY,SYSDDF,SYSGPAUT,SYSSTATS,SYSSTR and SYSUSER
- Convert DSNDB01.SPT01 from 8 KB to 32 KB page size
- Alter index DSNKDX01 back to the original key definition (DLOCATION,DCOLLID,DNAME,DCONTOKEN)

12.6 Implications of DB2 catalog restructure

A summary of the justifications and benefits provided by the restructuring of the DB2 catalog and directory follows:

- Bypass the 64 GB limit for catalog and directory table spaces (for example SPT01)
- Benefit of having DB2 catalog tables in a partition-by-growth table space
- DB2 managed data sets for the catalog and directory
- Catalog contention reduction for the removal of links and the introduction of row level locking
- Permitting use of SQL to locate words in bound packages including the new query tables
- Scanning SQL statements stored in the catalog using SQL
- Multiple binds running concurrently
- No need to run DSN1CHKR on a catalog to check the links, because they are replaced by referential integrity during DSNTIJEN execution
- DB2 can update the referential integrity (RI) information and catalog pages without getting a deadlock as frequently.
- Less contention for automatic REBIND on invalid packages
- DB2 can update the statistics and catalog pages without getting a deadlock as frequently.

For migrated systems, the catalog and directory restructuring happens during the DSNTIJEN job.

For new installations, the catalog and directory is NFM ready.

The structure of the DSNTIJEN job is similar to the past. It is started with CATENFM START to enter ENFM. followed by CATENFM CONVERT and REORG for every catalog and directory table space.

The DSNTIJEN job can be halted at any time during execution, and there is no need to modify the job before resubmitting it. It automatically skips the steps that were previously completed.

A special REORG is run in the DSNTIJEN job, and the 'CONVERTV10' keyword is used to indicate it is the special ENFM REORG.

Important: Catalog and directory objects must be recovered in a particular order because RECOVER needs information from some tables to recover others.

- Use the DSNTIJIC installation job to create backup copies of catalog and directory objects. If you are migrating from DB2 9 or earlier and if DB2 10 is in CM, the COPY SHRLEVEL(CHANGE) utility in the DSNTIJIC job automatically skips new or obsolete catalog and directory objects and issues message DSNU1530I with RC0 for each skipped object.
- Use the sequence provided in the *DB2 10 for z/OS Utility Guide and Reference,* SC19-2984. For DB2 10 CM, the sequence is the same as for DB2 9. For DB2 10 NFM, the sequence is different because of the changes in the catalog contents. You need to include LOB table spaces.

12.7 DSNZPARM change summary

Table 12-4 lists the system parameters that are eliminated in DB2 10 and must be removed from executing DSNTIJUZ jobs before reassembling the DSNZPARM module.

Table 12-4 Removed system parameters

Macro	DSNZPARM	Description/Comment
DSN6ENV		An environment macro that specifies whether the operating system is MVS/370 or MVS/XA. This macro is removed from SDSNMACS and from all invocation macros.
DSN6SYSP	LOGAPSTG	The maximum DBM1 storage that can be used by the fast log-apply process. DB2 10 currently assumes 100 MB.[a]
DSN6SPRM	SJMISSKY	Enables star join query performance enhancements.
DSN6SPRM	PARTKEYU	Specifies if values of partitioning key columns can be updated. The default is yes.
DSN6SPRM	MAX_UTIL_PARTS	Number of partitions of a compressed table space that REORG and LOAD process without limits.
DSN6SPRM	EDMBFIT	Specifies how free space is used for EDM pools larger than 40 MB in size.

a. The default value of LOGAPSTG is changed to 500 MB with APAR PM24808.

Table 12-5 lists the system parameters that are deprecated in DB2 10.

Table 12-5 Deprecated system parameters

Macro	DSNZPARM	Description/Comment
DSN6SPRM	RETVLCFK	Specifies whether the VARCHAR column is to be retrieved from a padded index.
DSN6SPRM	DISABSCL	Specifies whether SQLWARN1 and SQLWARN5 are set for non-scrollable cursors after the OPEN CURSOR or ALLOCATE CURSOR statement.
DSN6SPRM	OJPERFEH	Specifies whether to disable performance enhancements for outer join operations.
DSN6SPRM	OPTIOWGT	Controls how DB2 balances the I/O cost and CPU estimates when selecting access paths.
DSN6SPRM	OPTIXIO[a]	Provides stable I/O costing with less sensitivity to buffer pool sizes.
DSN6SPRM	PTCDIO[b]	Enables an optimizer enhancement to improve inefficient index access path for a single-table query.
DSN6SPRM	SMSDCFL[c]	Specifies a DFSMS data class for table spaces.
DSN6SPRM	SMSDCIX	Specifies a DFSMS data class for indexes.
DSN6SPRM	STATCLUS	Specifies the type of clustering statistics to be collected by the RUNSTATS utility.

a. Opaque parameter
b. Hidden parameter
c. As of DB2 9 NFM, you can specify the DATACLAS, MGMTCLAS, and STORCLAS parameters on the CREATE STOGROUP statement, which is the preferred method over SMSDCFL and SMSDCIX.

Table 12-6 lists the system parameters that have new default values in DB2 10.

Table 12-6 System parameters that are deprecated and have new default values

Macro	DSNZPARM	New default	Description/Comment
DSN6SPRM	SEQCACH	from BYPASS to SEQ	Specifies whether to use the sequential mode to read cached data from a 3990 controller.
DSN6SPRM	SEQPRES	from NO to YES	Specifies whether DB2 utilities that do a scan of a non-partitioning index followed by an update of a subset of the pages in the index allow data to remain in cache longer when reading data.
DSN6SPRM	DB2SORT	from DISABLE to ENABLE	Specifies whether DB2 utilities use DB2SORT if present.

The values for system parameter DLDFREQ (specifies how often, in the number of checkpoints, the level ID of a set or partition is to be updated) in macro DSN6SYSP are changed. DB2 10 changes the values of 0-32767 with a default value of 5 to the new values of ON and OFF with a default value of ON. This change provides the same default behavior, but the meaning of this parameter changes slightly. This parameter now means whether the level ID of a set or partition is to be updated at DB2-determined checkpoint intervals. When in MIGRATE mode, the DB2 10 installation CLIST automatically converts the migration input setting for DLDFREQ. If the parameter is set to 0, then it is set to OFF; if the parameter is set to a value between 1 and 32767, then it is set to ON.

As shown in Table 12-7, the improvements in virtual storage constraint relief delivered in DB2 10 allow the increase of the upper limits of these parameters to 10 times their previous maximums. Refer to 3.1, “Virtual storage relief” on page 52 for additional information regarding this topic. The upper limit of MAXOFILR[3] is the current CTHREAD setting. Because the CTHREAD maximum was raised in DB2 10, the MAXOFILR setting was raised also.

Table 12-7 System parameters with new maximums

Macro	DSNZPARM	New maximum	Description/Comment
DSN6SYSP	CTHREAD	from 2000 to 20000	Specifies the maximum number of allied threads (threads started at the local subsystem) that can be allocated concurrently.
DSN6SYSP	MAXDBAT	from 1999 to 19999	Specifies the maximum number of database access threads (DBATs) that can be active concurrently.
DSN6SYSP	IDFORE	from 2000 to 20000	Specifies the maximum number of allied threads (threads started at the local subsystem) that can be allocated concurrently.
DSN6SYSP	IDBACK	from 2000 to 20000	Specifies the maximum number of concurrent connections that are identified to DB2 from batch.
DSN6SYSP	MAXOFILR	from 2000 to 20000	Specifies the maximum number of data sets that can be open concurrently for processing of LOB file references (same as MAX USERS).

Table 12-8 lists parameters with changed defaults. The majority of these defaults are changed to be more in line with current best practices.

Table 12-8 System parameters with new default settings

Macro	DSNZPARM	New default value	Description/Comment
DSN6SYSP	STATIME	from 5 to 1	Specifies the number of minutes between gathering statistics[a].
DSN6SYSP	MONSIZE	Lower limit 1 MB Upper limit 64 MB	The upper limit is raised from 16,777,216 bytes (16 MB) to 67,108,864 bytes (64 MB) The lower limit (default) is raised from 262,144 bytes (256 KB) to 1,048,576 bytes (1 MB) On the DSNTIPN installation panel, the MONITOR SIZE field is modified to have a default setting of 1 MB and to accept values from 1 MB to 64 MB.
DSN6SPRM	DSMAX	from 10000 to 20000	Specifies the maximum number of data sets that can be open at one time.
DSN6SPRM	IRLMRWT	from 60 to 30	Specifies the number of seconds before a timeout is detected.
DSN6SYSP	CHKFREQ	from 500000 records to 5 minutes	Specifies the number of minutes or log records between log checkpoints.
DSN6SYSP	URLGWTH	from 0 to 10000	Specifies the number of log records that are to be written by an uncommitted unit of recovery before DB2 issues a warning message to the console.

[3] The MAXOFILR subsystem parameter specifies the maximum number of data sets that can be open concurrently for the processing of LOB file references.

Macro	DSNZPARM	New default value	Description/Comment
DSN6SYSP	URCHKTH	from 0 to 5	Specifies the number of checkpoint cycles that are to complete before DB2 issues a warning message to the console and instrumentation for an uncommitted unit of recovery.
DSN6SPRM	LRDRTHLD	from 0 to 10	Specifies the number of minutes that a read claim can be held by an agent before DB2 writes a trace record to report it as a long-running reader.
DSN6SPRM	CONSTOR	from NO to YES	Specifies whether DB2 is to periodically compact each thread's working storage area.
DSN6SPRM	MINSTOR	from YES to NO	Specifies whether DB2 is to use storage management algorithms that minimize the amount of working storage consumed by individual threads.
DSN6SPRM	IRLMSWT	from 300 to 120	Specifies the number of seconds that DB2 waits for the IRLM to start during autostart.
DSN6SPRM	MAXRBLK	from 8000 to 400000	Specifies the KB of storage needed for the RID pool.
DSN6SPRM	SRTPOOL	from 2000 to 10000	Specifies the KB of storage needed for the SORT pool.
DSN6SPRM	EN_PJSJ	from OFF to ON	Specifies whether dynamic index ANDing for star join, also known as *pair-wise join*, can be enabled when star join processing is enabled.
DSN6SPRM	UTSORTAL	from NO to YES	Specifies whether DB2 uses real-time statistics to determine the sort work data set sizes if real-time statistics data is available.
DSN6SPRM	SPRMPCWH [b]	from 5 to 1	Specifies the number of buffers per hash anchor (if positive) or the number of hash anchors per buffer (if negative).
DSN6SYSP	PCLOSEN	from 5 to 10	Specifies the number of consecutive DB2 checkpoints since a set or partition was last updated, after which DB2 converts the set or partition from read-write to read-only.
DSN6SPRM	RRULOCK	from NO to YES	Specifies whether to use the U (UPDATE) lock when using repeatable read (RR) or read stability (RS) isolation to access a table.
DSN6SPRM	NUMLKTS	from 1000 to 2000	Specifies the default value for the maximum number of page, row, or LOB locks that a single application can hold simultaneously in a single table or table space before lock escalation occurs.
DSN6SPRM	EDMDBDC	from 11700 to 23400	Specifies the minimum size (in KB) of the DBD cache that can be used by the EDM.
DSN6SPRM	EDM_ SKELETON_ POOL	from 5120 to 10240	Specifies the minimum size (in KB) of the EDM skeleton pool.
DSN6SPRM	EDMSTMTC	from 56693 to 113386	Specifies the size (in KB) of the statement cache that can be used by the EDM.

Macro	DSNZPARM	New default value	Description/Comment
DSN6SPRM	STATROLL	from NO to YES	Specifies whether the RUNSTATS utility aggregates the partition-level statistics, even though some parts might not contain data.
	BP8K0	from 1000 to 2000	Specifies the initial installation size in pages of the BP8K0 buffer pool.
	NUMCONDB	from 100 to 200	Specifies the estimated number of concurrently-open databases.

a. SMF type 101 statistics trace (IFCIDs 0001, 0002, 0202, 0217, 0225, and 0230) interval is at one minute interval, no matter what you specify in the STATIME DSNZPARM parameter.
b. Hidden parameter.

Table 12-9 lists the new DSNZPARMs for DB2 10. Of particular importance is INDEX_IO_PARALLELISM, which by default is set to YES. This parameter enables index insert parallelism. *Index insert parallelism* means that if you have more than one index defined on a table, DB2 10 updates the indexes in parallel instead of serially. There are also several new parameters associated with controlling with more granularity the use of FLASHCOPY.

Table 12-9 DB2 10 new system parameters

Macro, module, or panel	DSNZPARM	Default value	Description/Comment
DSN6SPRM	CHECK_FASTREPLICATION	REQUIRED	Specifies the FASTREPLICATION setting for the copy command of the CHECK utility. CHECK utilities using SHRLEVEL CHANGE invoke DSSCOPY to copy shadow data sets on which the CHECK is performed.
DSN6SPRM	DIRECTORY AND CATALOG DATA (CATDDACL, CATDMGCL, CATDSTCL)	blank blank blank	Specifies the explicit SMS classes that are used for defining VSAM data sets for the DB2 catalog and directory. Your SMS storage administrator defines these SMS classes. To use ACS routines for defining these data sets, leave this field blank.
DSN6SPRM	FCCOPYDDN	HLQ.&DB..&SN ..N&DSNUM..& UQ.	Defines the default value that is to be used for the FCCOPYDDN parameter of the FLASHCOPY clause of the DB2 utility control statement.
DSN6SPRM	FLASHCOPY_COPY	YES	Specifies whether the FLASHCOPY clause of the COPY utility is used by default.
DSN6SPRM	FLASHCOPY_LOAD	YES	Specifies whether the FLASHCOPY clause of the LOAD utility is used by default.
DSN6SPRM	FLASHCOPY_PPRC	REQUIRED	Specifies the behavior for DFSMSdss FlashCopy requests by DB2 Utilities when the target disk storage volume is in a peer to peer remote copy (PPRC) relationship.
DSN6SPRM	INDEX_IO_PARALLELISM	YES	Enables index I/O parallelism. The indexes can be updated in parallel. DB2 can use index I/O parallelism only on UTS and partitioned table spaces.
DSN6SPRM	MAXTEMPS_RID	NOLIMIT	Determines the maximum amount of temporary storage in the work file database that a single RID list can use at any given time.

Archive

Macro, module, or panel	DSNZPARM	Default value	Description/Comment
DSN6SPRM	PARA_EFF (parallelism efficiency)	50	Controls the efficiency that DB2 assumes for parallelism when DB2 chooses an access path. The integer value that is used for this parameter represents a percentage efficiency.
DSN6SPRM	PLANMGMT	EXTENDED	Specifies the default plan management policy to use when the PLANMGMT option is not explicitly specified for the bind or rebind of a package.
DSN6SPRM	PLANMGMTSCOPE	STATIC	Specifies the default plan management scope to use when the PLANMGMTSCOPE option is not explicitly specified for the bind or rebind of a package.
DSN6SPRM	REC_FASTREPLICATION	PREFERRED	Specifies whether recovery from a FlashCopy image copy should use FlashCopy.
DSN6SPRM	RRF	ENABLE	Specifies whether most newly created table spaces are to store data in reordered row format (RRF) or basic row format (BRF) by default.
DSN6SPRM	SECADM1	SECADM	Specifies the first of two authorization IDs or roles that are to have DB2 security administrator authority.
DSN6SPRM	SECADM1_TYPE	AUTHID	Specifies whether the entry in the SECURITY ADMIN 1 field is an authorization ID or a role.
DSN6SPRM	SECADM2	SECADM	Specifies the second of two authorization IDs or roles that are to have DB2 security administrator authority.
DSN6SPRM	SECADM2_TYPE	AUTHID	Specifies whether the entry in the SECURITY ADMIN 2 field is an authorization ID or a role.
DSN6SPRM	SEPARATE_SECURITY	NO	Specifies whether DB2 security administrator duties are separated from system administrator duties for this subsystem.
DSN6SPRM	WFDBSEP	NO	Specifies whether DB2 provides an unconditional separation of table spaces in the work file database based on the allocation attributes of the table spaces.
DSN6SYSP	ACCESS_CNTL_MODULE	DSNX@XAC	Specifies the member name of the load module that is to be used for the DB2 access control exit routine.
DSN6SYSP	DEL_CFSTRUCTS_ON_RESTART	NO	Specifies whether a DB2 restart attempts to delete CF structures (SCA, IRLM lock structure, and group buffers pools) and to rebuild them during restart.
DSN6SYSP	DPSEGSZ	32	Specifies the default segment size used for a partitioned table space when the CREATE TABLESPACE statement does not include the SEGSIZE parameter.
DSN6SYSP	IDAUTH_MODULE	DSN3@ATH	Specifies the member name of the load module that is used for the DB2 connection authorization exit routine.
DSN6SYSP	LOB_INLINE_LENGTH	0	Specifies the default length (in bytes) that is used for inline LOB columns.
DSN6SYSP	SIGNON_MODULE	DSN3@SGN	Specifies the member name of the load module that is used for the DB2 sign-on exit routine.
DSN6SYSP	SMFCOMP	OFF	Specifies the compress trace records that are destined for SMF.

Macro, module, or panel	DSNZPARM	Default value	Description/Comment
DSNHDECP	IMPLICIT_TIMEZONE	CURRENT	Determines the implicit time zone that is to be used when a time zone is not provided. This parameter applies to DB2 table columns and routing parameters that are declared with TIMESTAMP WITH TIME ZONE data types.
DSNTIPB	FLASHCOPY_REBUILD_INDEX	DSNTIP6	Specifies whether the FLASHCOPY clause of the REBUILD INDEX utility is used by default.
DSNTIPB	FLASHCOPY_REORG_INDEX	DSNTIP6	Specifies whether the FLASHCOPY clause of the REORG INDEX utility is used by default.
DSNTIPB	FLASHCOPY_REORG_TS	DSNTIP6	Specifies whether the FLASHCOPY clause of the REORG TABLESPACE utility is used by default.

Be aware that if you accept the default value of 32 for DSNZPARM DPSEGSZ, the behavior of the existing DDL to create classic partitioned table spaces now results in a UTS partitioned by range table space. If you are not prepared for this behavior to change, you need to set DSNZPARM DPSEGSZ=0.

As shown in Table 12-10, setting DPSEGSZ=0 and specifying the NUMPARTS parameter in a CREATE statement causes the classic partitioned table space to be created.

Table 12-10 DPSEGSZ and type of table space created

DPSEGZ setting	MAXPARTITIONS clause	NUMPARTS clause	Type of table space
0	Not specified	Not specified	Segmented table space with SEGSIZE 4
		Specified	Classic partitioned table space
	Specified	Not specified	Partition by growth table space with SEGSIZE = 32
		Specified	Partition by growth table space with SEGSIZE = 32
n > 0	Not specified	Not specified	Segmented table space with SEGSIZE 4
		Specified	Partition by range table space with SEGSIZE = n
	Specified	Not specified	Partition by growth table space with SEGSIZE = n
		Specified	Partition by growth table space with SEGSIZE = n

12.8 EXPLAIN tables in DB2 10

Typically, each new release of DB2 introduces new columns or modifies existing columns in the EXPLAIN tables. DB2 has traditionally honored EXPLAIN tables in previous release formats, but it is best practice to use the current-release format to obtain maximum benefit from the EXPLAIN data.

In addition, EXPLAIN tables were traditionally created in the EBCDIC encoding scheme because data in the DB2 catalog was EBCDIC encoded. Beginning with DB2 V8, almost all catalog data is Unicode encoded; therefore, it is increasingly beneficial to store EXPLAIN data in Unicode tables.

In DB2 10, the use of EBCDIC EXPLAIN tables and the use of EXPLAIN tables in previous release formats are deprecated. Thus, support for such tables might be removed in a future release of DB2. Begin to identify non-compliant tables, and eliminate or convert them.

APAR PK85068 can help you migrate to the DB2 10 Unicode format of EXPLAIN tables from the V8 or DB2 9 format.

12.8.1 EXPLAIN table changes

With DB2 10, the EXPLAIN tables have the following changes, which we discuss further in the sections that follow:

- Added EXPLAIN tables
- Added columns and column values
- Conversion to current format and Unicode

Added EXPLAIN tables

DB2 10 adds the following EXPLAIN tables:

- DSN_COLDIST_TABLE

  The column distribution table contains non-uniform column group statistics that are obtained dynamically by the DB2 optimizer.

- DSN_KEYTGTDIST_TABLE

  The key-target distribution table contains non-uniform index expression statistics that are obtained dynamically by the DB2 optimizer.

Added columns and column values

DB2 10 adds several columns and column values.

One of the changes is that the PLAN_TABLE column ACCESSTYPE has new values to identify hash access. The ACCESSTYPE column in the plan table has a value of 'H', 'HN', or 'MH', when the table that is accessed is organized by hash, and hash access is used to access the data. Columns ACCESSCREATOR and ACCESSNAME can now contain the hash overflow index name when hash access is used.

ACCESSTYPE column also has new values for range-list (ACCESSTYPE=’NR’) and IN-list workfile (ACCESSTYPE ‘IN’ or ‘I’).

Conversion to current format and Unicode

DB2 10 for z/OS deprecates the use of EBCDIC-encoded explain tables and disallows use of explain tables in formats prior to those delivered with DB2 V8. During migration to DB2 10 CM, there are tools to convert the explain tables to Unicode and to the latest version.

To facilitate this task, DB2 10 provides queries and jobs as follows:

- Queries to discover non-compliant tables. In DB2 V8 and DB2 9, these queries are added to job DSNTIJPM. In DB2 V8 only, they are added to job DSNTIJP9.
- A DSNTIJXA sample job that drives a DB2 REXX executable called DSNTXTA to alter EXPLAIN tables into the current-version format. You can specify to convert all EXPLAIN tables in DB2, or you can limit the conversion to a particular schema (creator).
- A DSNTIJXB sample job that drives a DB2 REXX executable called DSNTXTB to generate statements for migrating data from EBCDIC EXPLAIN tables with a specified schema. DSNTXTB produces the following types of statements:

Archived

- Control statements for processing by the DB2 utilities cross loader (EXEC SQL)
- SQL statements for processing by SPUFI or a similar dynamic SQL tool

These statements should not be processed before they are inspected and validated by the user.

- A DSNTIJXC sample job that shows how to process the statements that are generated by running DSNTIJXB.

In general, use the following process to convert from the current format to Unicode. We provide details about available tools in the sections that follow.

1. Identify non-compliant explain tables. Reports are provided in job DSNTIJPM (and in DSNTIJP9 in V8 only) that identify all schemas having one or more such tables.
2. Bring all EXPLAIN tables into the current release format. You can use job DSNTIJXA to bring all EXPLAIN tables in a specified schema into current release format. To convert all non-compliant EXPLAIN tables regardless of the schema, specify an asterisk (*) as the schema.
3. Migrate all EBCDIC-encoded explain tables to Unicode, which is a two-step process:
   a. Use job DSNTIJXB to generate DB2 cross loader control statements (and equivalent SQL statements) that can be used to copy all EBCDIC explain tables in a specified schema ID to Unicode equivalents. All EXPLAIN tables that belong to the specified schema must first be in the current release format.
   b. After examining the control statements generated by DSNTIJXB, use job DSNTIJXC to process them. DSNTIJXC assumes that you have purchased the DB2 Utilities Suite. If you have not, you can use the equivalent SQL generated by DSNTIJXB. The process works as follows for each table:
      i. RENAME TABLE to append '_EBCDIC' to the EBCDIC EXPLAIN table. Renaming the EBCDIC table makes its original name available to its Unicode replacement.
      ii. CREATE TABLE to create the explain table as a Unicode table.
      iii. CREATE AUX TABLE (if the table is a LOB table).
      iv. CREATE INDEX for each index on the EBCDIC table. To avoid name conflicts, the new indexes are prefixed with DSNU. If the name already starts with DSN, then a “U” is inserted at the fourth position, as shown in the following example:

```
PLAN_TABLE_IDX1 -> DSNU_PLAN_TABLE_IDX1
DSN_STATEMNT_TABLE_IDX1 -> DSNU_STATEMNT_TABLE_IDX1
```

      v. DECLARE CURSOR on the EBCDIC table to extract the data.
      vi. LOAD DATA INTO the Unicode table, using the cursor on the _EBCDIC table.

Repeat these steps until all explain tables are Unicode encoded and in the current release format.

12.8.2 Tools to help you convert to new format and Unicode

In this section, we provide details about the EXPLAIN tables conversion tools.

Queries for discovering non-compliant tables

The following queries are added to the V8 premigration checkout job (shipped as DSNTIJPM in V8) and the DB2 9 premigration checkout jobs (shipped as DSNTIJPM in DB2 9 and as DSNTIJP9 in V8):

- The first query identifies EBCDIC-encoded EXPLAIN tables that need to be migrated to Unicode.
- The second query identifies EXPLAIN tables that are either not in V8 format (if you are preparing to migrate to DB2 V8) or not in DB2 9 format (if you are preparing to migrate to DB2 9).

Both queries are lengthy and release-dependent; therefore, we do not discuss them in this book.

Before you migrate to DB2 10, use DSNTIJPM to identify non-compliant EXPLAIN tables. Then use DSNTIJXA/DSNTXTA to put these tables in the current release (V8 or DB2 9) format. Use DSNTIJXB/DSNTXTB and DSNTIJXC to migrate EBCDIC EXPLAIN tables to Unicode. After you migrate to DB2 10, wait until NFM before you rerun DSNTIJXA/DSNTXTA to add new-for-DB2 10 columns to your EXPLAIN tables.

Note that you can use these queries anytime after migration to identify non-compliant EXPLAIN tables.

Job DSNTIJXA and REXX exec DSNTXTA

DSNTXTA alters all EXPLAIN tables under a specified schema name into current-release format. The job prolog of DSNTIJXA explains how to customize it for use on your system.

If you identify non-compliant tables during preparations to migrate to a new release of DB2, note them, but wait to correct them until after you complete the migration and stabilize DB2 in NFM.

Job DSNTIJXA contains a single job step that calls DB2 REXX exec DSNTXTA, which accepts the following parameters:

- The DB2 SSID
- An authorization ID
- A schema name

If the schema name is an asterisk (*), DSNTXTA locates and converts all EXPLAIN tables under all schemas. Example 12-7 shows a sample call to DSNXTA.

Example 12-7 Sample call to DSNXTA

```
DSNTXTA +
    DSN SYSADM DSN8810
```

DSNTXTA analyzes each EXPLAIN table in a specified schema. If no tables are found, processing terminates with the following message:

```
DSNT090I DSNTXTA REQUEST CANCELLED BECAUSE THERE ARE NO EXPLAIN TABLES IN
SCHEMA schema-name
```

When a candidate table is found, the addition and alteration of columns is summarized in the following message:

```
DSNT091I DSNTXTA HAS ALTERED EXPLAIN TABLE schema-name.- table-name.
         COLUMN column-name-1 WAS description-of-change-1.
         COLUMN column-name-2 WAS description-of-change-2.
         ...
         COLUMN column-name-n WAS description-of-change-n.
```

When alteration of a table cannot be completed because of an unknown or unmanageable format, the following message is written and processing continues to the next table:

```
DSNT092I DSNTXTA DID NOT COMPLETE PROCESSING OF
schema-name.table-name BECAUSE reason.
```

When all tables in the schema have been processed, the following summary message is written:

```
DSNT093I DSNTXTA PROCESSING COMPLETE.
- SCHEMAS EXAMINED : total-schemas
- TABLES EXAMINED  : total-tables
- TABLES ALTERED   : total-altered
- TOTAL WARNINGS   : total-warnings
- TOTAL ERRORS     : total-errors
```

Note that in the DB2 V8 format of DSN_QUERY_TABLE, HASHKEY and HAS_PRED are columns 8 and 9, whereas in the DB2 9 format they are columns 9 and 10. Therefore, DSNTXTA cannot alter DSN_QUERY_TABLE from V8 format to a newer format. For each such case, DSNTXTA indicates with warning message DSNT092I that you need to use the DSNTXTB process to migrate the indicated DSN_QUERY_TABLE to a 10-column (DB2 9) format, as shown in Example 12-8.

Example 12-8 Warning message DSNT0921

```
DSNT092I DSNTXTA DID NOT COMPLETE PROCESSING OF
schema-name.DSN_QUERY_TABLE BECAUSE IT CANNOT BE
ALTERED TO A 10-COLUMN TABLE
```

In DB2 V8 and DB2 9, prior to APAR PK65772, the names of columns 13-16 of DSN_PTASK_TABLE contained the hashmark symbol, which is a variant EBCDIC character. In DB2 9 NFM, DSNTXTA checks for and renames these columns if necessary. In V8 and DB2 9 CM, RENAME COLUMN is not available so DSNTXTA indicates with warning message DSNT092I that you need to use the DSNTXTB process to migrate the indicated DSN_PTASK_TABLE to a DSN_PTASK_TABLE that uses the column name corrections introduced by PK65772:

```
DSNT092I DSNTXTA DID NOT COMPLETE PROCESSING OF
schema-name.DSN_PTASK_TABLE BECAUSE ONE OR MORE
COLUMN NAMES CONTAINS A VARIANT EBCDIC CHARACTER
```

Job DSNTIJXB and REXX exec DSNTXTB

DSNTXTB generates DB2 utilities cross loader control statements for creating, indexing, and loading Unicode-encoded copies of all EBCDIC EXPLAIN tables in a specified schema. A parallel set of DDL statements is generated for use by customers who do not have a license for the DB2 Utilities Suite. The job prolog of DSNTIJXB explains how to customize it for use on your system.

If you identify non-compliant tables during preparations to migrate to a new release of DB2, note them, but wait to correct them until after you complete migration and stabilize DB2 in NFM.

Job DSNTIJXB contains two job steps. The first deletes data sets created by the job. The second job step calls DB2 REXX DSNTXTB, which accepts these parameters:

- The DB2 SSID to which to connect
- The authorization ID to use
- The schema name of the explain tables to be converted
- The name of the target database
- The name of the target 4 KB page table space
- The name of the target 8 KB page table space

- The name of the target 16 KB page table space
- The name of the target 8 KB page LOB table space
- The name of the target 32 KB page LOB table space

Example 12-9 shows a sample of the call.

Example 12-9 Sample call to DSNTXTB

```
DSNTXTB +
    DSN +
    SYSADM +
    SCHEMA +
    TARGETDB +
    TS4K +
    TS8K +
    TS16K +
    LOBTS8K +
    LOBTS32K
```

DSNTXTB allocates the following output DDs:

- XLDOUT: Destination for generated DB2 cross loader control statements (RECFM=FB,LRECL=80)
- DDLOUT: Destination for generated SQL statements (RECFM=FB,LRECL=80)

Job DSNTIJXB does not process the generated control statements. These statements need to be inspected by the user and then processed by the DB2 cross-loader function (for example by using sample job DSNTIJCX) to migrate the tables to Unicode.

All explain tables must be in current release format, except that in DB2 9, DSN_QUERY_TABLE can be in V8 format.

DSNTXTB also generates a control statement to rename each EBCDIC table so that the name can be reused by the Unicode table. The renaming consists of adding the suffix _EBCDIC to the current table name.

DSNTXTB also generates control statements for creating the same indexes on Unicode tables as are defined on the EBCDIC tables. To avoid name conflicts, the new indexes are prefixed with DSNU.

When no candidate tables exist under the specified schema, DSNTXTB terminates with the following warning message:

```
DSNT094I: DSNTXTB REQUEST CANCELLED BECAUSE THERE ARE NO
EBCDIC EXPLAIN TABLES IN SCHEMA schema-name
```

When a candidate table is not in current release format, DSNTXTB terminates with the following message:

```
DSNT095I: DSNTXTB REQUEST CANCELLED BECAUSE TABLE schema-name.table-name
IS NOT IN THE REQUIRED FORMAT AT COLUMN NUMBER column-number.
- EXPECTED: expected-format - FOUND   : actual-format.
```

When the target database does not exist, DSNTXTB generates a statement to create it. If it exists but is not a default Unicode database, DSNTXTB writes the following warning if the target database:

```
DSNT096I: WARNING: THE SPECIFIED TARGET DATABASE, database-name,
IS NOT A UNICODE DATABASE
```

When a target table space exists but is not of the required type, processing terminates with the following message:

```
DSNT097I: DSNTXTB REQUEST CANCELLED BECAUSE THE SPECIFIED TARGET required-type
TABLE SPACE, database-name.- table-space-name, IS AN EXISTING actual-type TABLE
SPACE
```

When a target table space of the required type already exists, DSNTXTB writes the following message once, only when the first candidate EXPLAIN table is discovered that will reside in it:

```
DSNT098I: DSNTXTB HAS GENERATED STATEMENTS TO CREATE UNICODE EXPLAIN TABLES FOR
THE SPECIFIED SCHEMA IN AN EXISTING TARGET TABLE SPACE, database-name.-
table-space-name
```

For each candidate table, the following message is written:

```
DSNT099I: DSNTXTB HAS GENERATED STATEMENTS FOR MIGRATING TABLE
schema-name.table-name TO conversion-type
```

In addition, the following statements are generated as DB2 cross loader control statements and written to the XLDOUT DD:

- RENAME TABLE to append '_EBCDIC' to the table name. The EBCDIC table is renamed to make its original name available to its Unicode replacement.
- CREATE TABLE to create the table as a Unicode table
- CREATE AUX TABLE (if the table is a LOB table)
- CREATE INDEX for each index on the EBCDIC table
- DECLARE CURSOR on the EBCDIC table to extract the data there
- LOAD DATA INTO the Unicode table, using the cursor on the _EBCDIC table

Parallel SQL statements are written to the DDLOUT DD, except that each DECLARE CURSOR and LOAD DATA INTO statement pair is replaced by INSERT INTO unicode-explain-table-name SELECT FROM ebcdic-explain-table name.

Job DSNTIJXC

Job DSNTIJXC contains a single job step that invokes the DB2 Utilities cross loader. It can be used to process statements generated by running job DSNTIJXB.

The job prolog of DSNTIJXC explains how to customize it for use on your system.

If you identify non-compliant tables during preparations to migrate to a new release of DB2, note them, but wait to correct them until after you complete migration and stabilize DB2 in NFM.

The cross loader is a function of the EXEC SQL utility component of the licensed DB2 Utilities Suite. If the Utilities Suite is not installed on your DB2, use SPUFI or a similar dynamic SQL tool to process the SQL statements generated by running job DSNTIJXB.

Archive

12.8.3 Converting the EXPLAIN table to new format and Unicode

As we mention earlier, DB2 10 for z/OS deprecates use of EBCDIC-encoded EXPLAIN tables and disallows use of EXPLAIN tables in formats prior to those delivered with DB2 V8. There are tools to help with migration. In this section, we look at the steps that you need to take to convert EXPLAIN tables to UNICODE and into the DB2 10 format:

1. Run job sample job DSNTIJXA. Job DSNTIJXA calls REXX exec DSNTXTA, which alters all EXPLAIN tables or EXPLAIN tables that belong to the specified creator ID to the DB2 10 format.
2. If you did not convert your EXPLAIN tables to UNICODE encoding before you migrated to DB2 10, complete the following steps:
   a. Customize job DSNTIJXB with the following values:
      - A DB2 subsystem name
      - An authorization ID
      - The schema name of the EXPLAIN tables to convert
      - The name of the target database
      - The name of the target 4 KB page table space
      - The name of the target 8 KB page table space
      - The name of the target 16 KB page table space
      - The name of the target 8 KB page LOB table space
      - The name of the target 32 KB page LOB table space
   b. Run job DSNTIJXB. Job DSNTIJXB calls REXX exec DSNTXTB, which generates DB2 cross-loader control statements and equivalent SQL statements that can be used to copy all EBCDIC EXPLAIN tables under a specified creator ID to Unicode equivalents. All EXPLAIN tables that belong to the specified creator ID must be in DB2 10 format (see step 1).
   c. If you have the DB2 Utilities Suite, run job DSNTIJXC to process the cross loader control statements that were generated by DSNTIJXB. Otherwise, use the equivalent SQL statements that were generated by DSNTIJXB to convert the tables to Unicode.
   d. Repeat steps a, b, and c for each creator that has at least one EBCDIC EXPLAIN table

Example 12-10 shows the typical output from the execution of job DSNTIJXA. Note that at the end of the output, a summary is provided that shows what was found and what was changed to bring the table to DB2 10 format. Messages DSNT091I are completely successful changes, and other messages such as DSNT092I describe warnings and errors received.

Example 12-10 Output of job DSNTIJXA program DSNTXTA to convert all schemas

```
READY
  DSNTXTA DB2DBA DB2R6 *

DSNTXTA entry:
  Subsystem ID ..........................................  DB8A
  Authorization ID ......................................  DB2DBA
  Schema ................................................  *
****** DB2 signature: DSN10010 ******

DSNT091I: DSNTXTA HAS ALTERED EXPLAIN TABLE JEFF.PLAN_TABLE.
          COLUMN APPLNAME WAS CHANGED FROM CHAR(8) TO VARCHAR(24)
          COLUMN VERSION WAS CHANGED FROM VARCHAR(64) TO VARCHAR(122)
          COLUMN GROUP_MEMBER WAS CHANGED FROM CHAR(8) TO VARCHAR(24)
          COLUMN PARENT_PLANNO WAS ADDED AS SMALLINT NOT NULL WITH DEFAULT
          COLUMN BIND_EXPLAIN_ONLY WAS ADDED AS CHAR(1) NOT NULL WITH DEFAULT
```

```
          COLUMN SECTNOI WAS ADDED AS INTEGER NOT NULL WITH DEFAULT
          COLUMN EXPLAIN_TIME WAS ADDED AS TIMESTAMP NOT NULL WITH DEFAULT
          COLUMN MERGC WAS ADDED AS CHAR(1) NOT NULL WITH DEFAULT
          COLUMN MERGN WAS ADDED AS CHAR(1) NOT NULL WITH DEFAULT

. . .

DSNT092I: DSNTXTA DID NOT COMPLETE PROCESSING OF JOHN.DSN_PTASK_TABLE
          BECAUSE ONE OR MORE COLUMN NAMES CONTAINS A VARIANT EBCDIC CHARACTER
          COLUMN GROUP_MEMBER WAS ADDED AS VARCHAR(24) NOT NULL WITH DEFAULT
          COLUMN SECTNOI WAS ADDED AS INTEGER NOT NULL WITH DEFAULT
          COLUMN COLLID WAS ADDED AS VARCHAR(128) NOT NULL WITH DEFAULT
          COLUMN VERSION WAS ADDED AS VARCHAR(122) NOT NULL WITH DEFAULT

. . .

DSNT093I: DSNTXTA PROCESSING COMPLETE:
          - SCHEMAS EXAMINED : 7
          - TABLES EXAMINED  : 90
          - TABLES ALTERED   : 87
          - INDEXES EXAMINED : 0
          - INDEXES RECREATED: 0
          - TOTAL WARNINGS   : 9
          - TOTAL ERRORS     : 1

*** End   RC=8
READY
END
```

Following the execution of job DSNTIJXA, all altered table spaces are placed in advisory REORG-pending (AREO*) state. To see all the objects affected, issue the following command:

```
-DISPLAY DB(*) SPACENAM(*) ADVISORY(AREO*)
```

During the next steps, job DSNTIJXB converts your explain tables to UNICODE encoding by creating new tables. This step is then followed by either running the cross loader (job DSNTIJXC) or processing the generated SQL to populate these new tables. For any tables that are already using Unicode encoding but that are not in DB2 10 format prior to the start of this process, you need to reorganize them to remove the AREO* status. After you convert the EBCDIC tables to UNICODE and copy the data to the UNICODE tables, you might want to drop the EBCDIC tables.

When migrating from DB2 V8 to DB2 10 and converting the EXPLAIN tables to DB2 10 format and to UNICODE, you can convert the DSN_STATEMENT_CACHE_TABLE to DB2 10 format. You can convert this table to DB2 10 format only when DB2 10 enters NFM. Instead, during the conversion process in DB2 CM, the DSN_STATEMENT_CACHE_TABLE is converted to DB2 9 format.

The reason for this is that DSN_STATEMENT_CACHE_TABLE contains BIGINT columns in DB2 10, and BIGINT is not a supported column type in DB2 V8. If this conversion were allowed when in DB2 10 CM and if you had to fall back to DB2 V8, DSN_STATATEMENT_CACHE_TABLE becomes a frozen object. However, DB2 10 tolerates DSN_STATEMENT_CACHE_TABLE in DB2 V8 and DB2 9 formats. When DB2 10 is in NFM, rerun job DSNTIJXA to convert DSN_STATEMENT_CACHE_TABLE to DB2 10 format. Be

Archive

aware that at this time this table is placed in AREO* again, and you need to follow up and run a REORG to remove this state.

Another issue that is worth mentioning here is a possible problem with the conversion of the the DSN_PTASK_TABLE table. It is possible for the names of four columns of this table to contain a variant EBCDIC character (meaning that it does not reside at a consistent code point across all code pages). The names of these four columns can contain code point x'7B', which corresponds to a hashmark symbol (#) in CCSID 37 but to a different characters in many other EBCDIC code pages.

If the DDL for creating DSN_PTASK_TABLE was processed with an application coding scheme other than CCSID 37, the SYSIBM.SYSCOLUMNS NAME entry for these columns might be corrupted and cause performance tools such as the Optim Query Tuner to issue error messages because DSN_PTASK_TABLE is not in the correct format.

If you encounter this issue, sample job DSNTIJOC and APAR PK65772 contain additional details and the instructions that you need to convert tables to the new format. Table 12-11 lists the columns in question.

Table 12-11 DSN_PTASK_TABLE column name corrections

Old name	New name
LPTLOPG#	LPTLOPG
LPTHIPG#	LPTHIPG
LPTLOPT#	LPTLOPT
LPTHIPT#	LPTHIPT

PK70423 update: PK70423 updates Optim Query Tuner and DB2 optimizer to ignore DSN_PTASK_TABLE tables that use the old column names.

12.9 SMS-managed DB2 catalog

Prior to DB2 10, there was no hard requirement for SMS managed catalog and directory objects. You could explicitly define these objects or use SMS to manage them. When migrating to DB2 10 CM, all new or changed indexes and new or changed table spaces for the catalog and directory must be SMS managed.

Prior to migration, you must define an SMS source control data set (SCDS) for storing the base configuration, an SMS storage group, an SMS Storage class with volumes, an SMS data class for allocating data sets in extended format (EF) and using the extended addressability (EA) attribute and an SMS automatic class selection (ACS) routine for managing the DB2 catalog and directory. Installation job DSNTIJSS does all this for you. Note that this job builds a brand new SMS environment to be used for DB2.

If you already have an existing SMS environment defined, further customization of this job is required to incorporate the definition changes into the existing control data sets. It might be simpler to make the definition changes through Interactive Storage Management Facility (ISMF). If this is the case, you need to define a data class with the EXT attribute in the DATA SET NAME TYPE field of the DATA SET CLASS DEFINE panel of ISMF. Then, you need to associate an ACS routine to the DB2 catalog and directory with this data class. It is best to

coordinate with your DASD management group to make these changes and these changes must be in place prior to migration. If you fail to have this change defined prior to migration, the migration job fails with the message shown in Example 12-11.

Example 12-11 Failure if no SMS definitions for DB2 catalog and directory are defined

```
DSNU000I    231 20:25:46.89 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = RELODCAT
DSNU1044I   231 20:25:46.91 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    231 20:25:46.98 DSNUGUTC -  CATMAINT UPDATE
DSNU750I    231 20:25:47.04 DSNUECM0 - CATMAINT UPDATE PHASE 1 STARTED
DSNU777I    231 20:25:47.04 DSNUECM0 - CATMAINT UPDATE STATUS - VERIFYING CATALOG IS AT CORRECT LEVEL FOR MIGRATION.
DSNU777I    231 20:25:47.12 DSNUECM0 - CATMAINT UPDATE STATUS - BEGINNING MIGRATION SQL PROCESSING PHASE.
DSNU778I  -DB8A 231 20:25:49.82 DSNUEXDL - ERROR PROCESSING SQL STATEMENT -
             SQL CODE IS: -904
             SQL MESSAGE TEXT: UNSUCCESSFUL EXECUTION CAUSED BY AN UNAVAILABLE RESOURCE. REASON 00D70025, TYPE OF
RESOURCE 00000220, AND RESOURCE NAME DB8AU.DSNDBC.DSNDB06.DSNADH02.I0001.A001
             SQL STATEMENT:
CREATE INDEX SYSIBM.DSNADH02 ON SYSIBM.SYSDBAUTH (NAME) USING STOGROUP
"00000001" FREEPAGE 0 PCTFREE 10 BUFFERPOOL BP0 NOT PADDED CLOSE NO
DSNU017I    231 20:25:52.86 DSNUGBAC - UTILITY DATA BASE SERVICES MEMORY EXECUTION ABENDED, REASON=X'00E40601'
CAUSE=X'25EC7C69'
```

User-defined catalog indexes and other DB2 data sets, such as active logs and BSDS, are not required to be SMS managed and can continue as before. The remainder of the catalog and directory objects not changed to SMS managed during the migration process are changed to be SMS managed the next time that they are reorganized.

12.10 Skip level migration

DB2 10 for z/OS provides the capability to migrate from DB2 9 NFM and also directly from DB2 V8 NFM. The process of migrating directly from DB2 V8 NFM to DB2 10 CM is called *skip level migration*. This capability was also available when migrating from DB2 V5 or DB2 V6 to DB2 V7.

Although this capability affords greater flexibility as to how and when you can migrate to DB2 10, making the decision to do a skip level migration requires careful planning to make the correct decision. Be aware that preparing for a skip level migration is more complicated than migrating from DB2 9. The actual skip level migration process itself is rather simple and allows you to jump directly from DB2 V8 NFM to DB2 10 CM without any intermediate steps. See Example 12-12 for sample job output when running job DSNTIJTC to migrate directly from DB2 V8 NFM to DB2 10 CM.

Example 12-12 Output of CATMAINT job DSNTIJTC DB2 V8 NFM to DB2 10 CM

```
DSNU000I    232 20:26:22.39 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = RELODCAT
DSNU1044I   232 20:26:22.42 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    232 20:26:22.50 DSNUGUTC -  CATMAINT UPDATE
DSNU750I    232 20:26:22.54 DSNUECM0 - CATMAINT UPDATE PHASE 1 STARTED
DSNU777I    232 20:26:22.54 DSNUECM0 - CATMAINT UPDATE STATUS - VERIFYING CATALOG IS AT CORRECT LEVEL FOR MIGRATION.
DSNU777I    232 20:26:22.58 DSNUECM0 - CATMAINT UPDATE STATUS - BEGINNING MIGRATION SQL PROCESSING PHASE.
DSNU777I    232 20:26:51.88 DSNUEXUP - CATMAINT UPDATE STATUS - BEGINNING ADDITIONAL CATALOG UPDATES PROCESSING.
DSNU777I    232 20:26:51.89 DSNUEXUP - CATMAINT UPDATE2 STATUS - BEGINNING SYSCOPY TABLE SPACE MIGRATION PROCESSING.
DSNU777I    232 20:26:51.89 DSNUECM0 - CATMAINT UPDATE STATUS - UPDATING DIRECTORY WITH NEW RELEASE MARKER.
DSNU752I    232 20:26:51.90 DSNUECM0 - CATMAINT UPDATE PHASE 1 COMPLETED
DSNU010I    232 20:26:51.97 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

The testing and rollout phases of your skip level migration project might be only slightly longer than a “normal” migration. However, the planning, education, and remediation work that you

need to perform for a skip level migration is slightly larger than double that of a normal migration. You need to take into account *all* the release changes between DB2 V8 and DB2 9 as well as those between DB2 9 and DB2 10.

For information about all the release changes, refer to *DB2 10 for z/OS Installation and Migration Guide,* GC19-2974. It is also worth reviewing all other DB2 9 documentation, including the DB2 9 for z/OS announcement material that is available at:

`http://www.ibm.com/common/ssi/rep_ca/8/897/ENUS206-098/ENUS206-098.PDF`

This material can helps you in sizing and planning skip level migration and ensures that all changes are taken into account. Pay close attention to those items that were deprecated in DB2 V8 or deprecated in DB2 9, which are eliminated in DB2 9 or eliminated in DB2 10 for z/OS. Furthermore, coordinate with any third-party software vendors to ensure that their code can handle a skip level migration and is ready to tolerate changes introduced in DB2 9 and DB2 10.

Make sure that you allocate the appropriate amount of time in your project plan to identify and take appropriate action to handle all of these changes, the remediation work to be done, and the education and learning curve to move forward two releases. The total preparation time that is required for you to get systems in shape for a skip level migration might save you only 20-30% of the overall preparation time that it would have taken to first go to DB2 9 and then later go to DB2 10.

Also consider the risk factor that is involved in introducing two releases worth of changes into a stable working environment. How soon after GA do you normally roll out a new release? Can you wait that long to do a skip level release to DB2 10, and can you go that long without the new functions provided by DB2 9? The major enhancements that you will be waiting longer for include:

- CPU reductions for utilities are substantial in DB2 9.
- DB2 9 allows more online schema changes and utilities that allow concurrent access.
- The BACKUP SYSTEM and RESTORE SYSTEM capabilities are improved.
- Improved granularity for security (including roles and trusted contexts) and better network security come in DB2 9.
- XML has become pervasive, and delivery in DB2 improves productivity while it avoids more data integration projects for the future.
- DB2 9 offers warehouse delivery in DB2 for z/OS with improved SQL, optimization, and surrounding products.
- Index compression can reduce the disk space for indexes by half.

Keep in mind that end of service for DB2 V8 is the end of April 2012. Plan to migrate before running on out of service code. Are you better off migrating to DB2 9 first? DB2 9 has been available for several years. Migrating to DB2 9 is might be faster, easier, and smoother than a skip level migration.

Before migrating to DB2 10, expand the BSDS (using the DSNJCNVB job that is supplied in the V8 DSNTIJUZ procedure) to support 10,000 data sets per copy of the archive logs and 93 data sets per copy of the active logs.

Be aware when migrating directly from DB2 V8 to DB2 10 that no DB2 9 or DB2 10 new functions are available until you reach DB2 10 NFM.

See *DB2 10 for z/OS Installation and Migration Guide,* GC19-2974, for details about how these changes might impact you.

Other interesting facts to consider for a skip level migration:

- After you start the migration process for a V8 system to DB2 10, you can fall back only to V8.
- After you start the migration process for a V8 system to DB2 10, if you fall back to V8, you cannot migrate to DB2 9.
- If you migrate a V8 system to DB2 10, you cannot use DB2 9 new functions until DB2 10 NFM is reached.
- A data sharing group that started migrating from V8 to DB2 10 cannot have any DB2 9 members.
- A DB2 9 system that has started the migration to DB2 10 can only fall back to DB2 9.
- A data sharing group that started migration from DB2 9 NFM to DB2 10 cannot have any V8 members.
- When migrating DB2 V8 to DB2 10 and converting EXPLAIN tables to Unicode, you cannot convert DSN_STATEMENT_CACHE_TABLE to DB2 10 format until DB2 enters DB2 10 NFM (BIGINT is not supported in V8). However, DB2 10 tolerates DSN_STATEMENT_CACHE_TABLE in V8 format. When DB2 is in DB2 10 NFM, rerun job DSNTIJXA to convert DSN_STATEMENT_CACHE_TABLE to DB2 10 format.
- Use the DB2 10 early code when migrating DB2 V8 to DB2 10 so that you can use the -REFRESH DB2,EARLY command if necessary. DB2 10 early code also provides some additional VSCR because, while the DB2 V8 and DB2 9 early code require some storage to remain below the bar, the DB2 10 early code allows it to be above the bar.

12.11 Fallback

When migrating to DB2 10, you can fall back to the previous phase of the migration process. Here, we discuss briefly a few considerations for the following fall back scenarios:

- From CM8 or CM9 to the previous version of DB2
- From ENFM8 to CM8* or from ENFM9 to CM9*
- From NFM to ENFM8* or ENFM9*
- From Fallback from NFM to CM8* or CM9*

For views of fallback statuses, see Figure 12-2 on page 475 and Figure 12-3 on page 476. For a complete list of fallback considerations, refer to the *DB2 10 for z/OS Installation and Migration Guide,* GC19-2974.

12.11.1 Implication to catalog image copy job

When you fall back to DB2 V8, you must update your old catalog image copy job to account for the following table spaces that were created during the migration to DB2 10 CM:

- DSNDB06.SYSCONTX, this table space contains tables SYSCONTEXT, SYSCTXTTRUSTATTRS and SYSCONTEXTAUTHIDS
- DSNDB06.SYSPLUXA, this table space contains table SYSROUTINESTEXT
- DSNDB06.SYSROLES, this table space contains tables SYSOBJROLEDEP and SYSROLES
- DSNDB06.SYSRTSTS, this table space contains tables SYSTABLESPACESTATS and SYSINDEXSPACESTATS
- DSNDB06.SYSTARG, this table space contains table SYSKEYTARGETS

- DSNDB06.SYSTSASC, this table space contains table SYSDUMMYA
- DSNDB06.SYSTSUNI, this table space contains table SYSDUMMYU
- DSNDB06.SYSXML, this table space contains tables SYSXMLRELS and SYSXMLSTRINGS

When you fall back to DB2 9, you must update the old catalog image copy job to account for the following table spaces that were created during the migration to DB2 10 CM:

- DSNDB06.SYSTSASC, this table space contains table SYSDUMMYA
- DSNDB06.SYSTSUNI, this table space contains table SYSDUMMYU

12.11.2 Frozen objects

Falling back does not undo changes that the migration process made to the catalog. Objects that were affected by DB2 10 function might become frozen after fallback. In particular, be aware of any plans, packages or views that use any new syntax, objects, or bind options.

12.12 Improvements to DB2 installation and samples

In this section, we discuss improvements to the DB2 installation process and include a sample for XML.

New SECADM authority: The SECADM authority can now also execute work while DB2 is in MAINT mode just like SYSADM and SYSOPR.

12.12.1 Installation pop-up panel DSNTIPSV

When migrating to DB2 10, a panel allows you to save changes in the output member that is specified on panel DSNTIPA. You can do this save at any point while you are navigating through the panels. To save your changes, enter SAVE in the command line of any installation panel, and choose option 1. Figure 12-10 shows the new DSNTIPSV panel.

```
DSNTIPSV
  Save changes in progress:

  Select one.
       1.  Save and continue
       2.  Save and exit
       3.  Continue without saving

  PRESS:  ENTER to continue
          RETURN to exit
```

Figure 12-10 Installation pop-up panel DSNTIPSV

12.12.2 Job DSNTIJXZ

When using the installation CLIST to install or migrate to a new version of DB2, you have always been able to save the configuration options (which consists of DSNZPARM settings, buffer pool settings, and other install related settings) into an output member often referred to

as DSNTIDxx member. The saved DSNTIDxx member can subsequently be used as input to the CLIST when migrating to a new version. This member is used by the install CLIST to generate your installation and migration jobs.

One of the jobs generated by the CLIST is DSNTIJUZ which assembles the systems parameter module DSNZAPRM. Although the CLIST provides an update function for maintaining the settings in the DSNTIDxx member and regenerating this job, you have the capability of manually updating private copies of this job. In the process, the values in the current DSNTIDxx member are outdated and might make subsequent migration processes time consuming, error-prone and cumbersome as you would have to manually verify each panel entry that corresponds to each entry in DSNTIJUZ.

DB2 10 for z/OS provides a new job DSNTIJXZ.DSNTIJXZ executes a batch program called DSNTXAZP that can be used to update a CLIST defaults input member (DSNTIDxx) with the current DSNZPARM and buffer pool settings that are defined in the specified DB2 subsystem. This tool can help ease your migration process by eliminating the need to manually compare current DB2 settings with the values defined in the CLIST defaults input member to be used for migration. Furthermore, this tool provides a report that maps install CLIST parameters to actual DSNZPARM parameters, which can be helpful because there are a number of DSNZPARM parameters that map to a different name in the installation CLIST.

The DSNTXAZP program requires the following arguments:

```
//STEP01 EXEC PGM=DSNTXAZP,PARM='ssid action'
```

Where:

- *ssid* is the subsystem ID.
- *action* is one of the following values:

`UPDATE_ALL`	Creates a new CLIST defaults input member and updates it with the current subsystem parameter and buffer pool settings.
`UPDATE_BPOOL`	Creates a new CLIST defaults input member and updates it with the current buffer pool settings only (no subsystem parameter changes).
`UPDATE_ZPARM`	Creates a new CLIST defaults input member and updates it with the current subsystem parameter settings only (no buffer pool changes).

Usage note: This job is made available to DB2 V8 and DB2 9.

Example 12-13 Execution of DSNTRIN with PREVIEW option in INSTALL mode

```
//*
//DSNTRIN EXEC PGM=DSNTRIN,COND=(4,LT),
//            PARM=('DB2SSN(VA1A) MODE(INSTALL-PREVIEW) AUTHID(ADMF001-
//            ) SECDEFID(ADMF001)')
//DBRMLIB  DD  DISP=SHR,DSN=DSN!!0.SDSNDBRM
//SYSUT1   DD  UNIT=SYSDA,SPACE=(27930,(10,5)),
//             DCB=(RECFM=FB,LRECL=133)
//SYSPRINT DD  SYSOUT=*,DCB=(RECFM=FB,LRECL=133)
//SYSTSPRT DD  SYSOUT=*
//CFIGOUT  DD  SYSOUT=*
//SQLOUT   DD  SYSOUT=*
//BINDOUT  DD  SYSOUT=*
//JCLOUT   DD  DSN=USER.JCLLIB(DB2INST),DISP=SHR
```

```
//DB2OPT   DD  *
 STOGROUP(DB2_ROUTINES_GROUP)
 BP4K(BP0)
 BP8K(BP8K0)
 BP16K(BP16K0)
 BP32K(BP32K)
 LOBBP8K(BP8K0)
 LOBBP16K(BP16K0)
/*
//CFIGIN   DD  *
...
   The configuration control statements go here
...
```

12.12.3 Installation verification procedure

When you complete the installation verification procedure (IVP), save the verification objects. You need them when you migrate to the next release of DB2.

Table 12-12 shows the new or revised programs and jobs that are used to run the installation or migration verification procedure. These jobs are designed to run with minimal interaction on your part. Refer to *DB2 10 for z/OS Installation and Migration Guide,* GC19-2974, for details about running the IVP and required tailoring of these jobs.

Table 12-12 New or revised installation or migration verification jobs

Job	C language program	Description
DSNTEJ6O	DSN8DTS1	Shows how to use the DB2-supplied stored procedures SYSPROC.SYSTS_START, SYSPROC.SYSTS_CREATE, and SYSPROC.SYSTS_UPDATE to create and initialize a text search index.
DSNTEJ6O	DSN8DTS2	Shows how to use the DB2-supplied stored procedures SYSPROC.SYSTS_START and SYSPROC.SYSTS_DROP to drop a text search index.

12.12.4 Sample for XML

Table 12-13 shows the revised sample job DSNTEJ2H. This job tests the XML program preparation procedures.

Table 12-13 Revised sample job DSNTEJ2H: Programs

Program	Description
DSN8EDXR	Sample application to format and display an XML record
DSN8EDXU	Sample application to format and update an existing XML record
DSN8EDXI	Sample application to insert an XML record
DSN8EDXD	Sample application to delete an XML record

Archived

Before running the sample stored procedures, you need to make sure that you have run the DSNTIJRT job to configure and install the stored procedures. You need to be aware that sample stored procedures have new names if you used DSNTIJRT to install them.

For detailed information, refer to *DB2 10 for z/OS Installation and Migration Guide,* GC19-2974.

12.13 Simplified installation and configuration of DB2-supplied routines

DB2 10 introduces changes to simplify installation, configuration, validation, and service of DB2-supplied stored procedures and user-defined functions that are provided with the DB2 for z/ OS base product (that is, those that ship an external module in the DB2 for z/OS base FMID). We refer to these stored procedures and functions hereinafter simply as *the DB2-supplied routines*.

The number of DB2-supplied routines has grown dramatically in recent releases. DB2 10 for z/OS, for example, currently provides 110+ stored procedures and UDFs. Most routines entail supporting objects such as databases, created global temporary tables, and packages. All are fenced and, therefore, require a WLM environment, often with specific parameter settings and address space properties.

Table 12-14 summarizes the stored procedures and their WLM environment setup in DB2 10.

Table 12-14 DB2-supplied stored procedures and WLM environment setup

DB2-supplied routine	Schema name	NUMTCB	Core WLM environment	APF authorized	Program controlled
ADMIN_COMMAND_DSN	SYSPROC	1	DSNWLM_REXX	No	No
ADMIN_COMMAND_UNIX	SYSPROC	40 - 60	DSNWLM_PGM_CONTROL	No	Yes
ADMIN_DS_BROWSE	SYSPROC	40 - 60	DSNWLM_GENERAL	Yes	No
ADMIN_DS_DELETE	SYSPROC	40 - 60	DSNWLM_GENERAL	Yes	No
ADMIN_DS_LIST	SYSPROC	40 - 60	DSNWLM_GENERAL	Yes	No
ADMIN_DS_RENAME	SYSPROC	40 - 60	DSNWLM_GENERAL	Yes	No
ADMIN_DS_SEARCH	SYSPROC	40 - 60	DSNWLM_GENERAL	Yes	No
ADMIN_DS_WRITE	SYSPROC	40 - 60	DSNWLM_GENERAL	Yes	No
ADMIN_INFO_HOST	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
ADMIN_INFO_SMS	SYSPROC	40 - 60	DSNWLM_GENERAL	Yes	No
ADMIN_INFO_SQL	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
ADMIN_INFO_SYSPARM	SYSPROC	1	DSNWLM_NUMTCB1	No	N0
ADMIN_JOB_CANCEL	SYSPROC	40 - 60	DSNWLM_PGM_CONTROL	Yes	Yes
ADMIN_JOB_FETCH	SYSPROC	40 - 60	DSNWLM_PGM_CONTROL	Yes	Yes
ADMIN_JOB_QUERY	SYSPROC	40 - 60	DSNWLM_PGM_CONTROL	Yes	Yes
ADMIN_JOB_SUBMIT	SYSPROC	40 - 60	DSNWLM_PGM_CONTROL	No	Yes
ADMIN_TASK_ADD	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No

DB2-supplied routine	Schema name	NUMTCB	Core WLM environment	APF authorized	Program controlled
ADMIN_TASK_CANCEL	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
ADMIN_TASK_REMOVE	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
ADMIN_TASK_UPDATE	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
ADMIN_UTL_EXECUTE	SYSPROC	40 - 60	DSNWLM_PGM_CONTROL	Yes	Yes
ADMIN_UTL_MODIFY	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
ADMIN_UTL_MONITOR	SYSPROC	40 - 60	DSNWLM_PGM_CONTROL	Yes	Yes
ADMIN_UTL_SCHEDULE	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
ADMIN_UTL_SORT	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
ALTER_JAVA_PATH	SQLJ	40 - 60	DSNWLM_GENERAL	No	No
CREATE_SESSION	DB2DEBUG	5 - 20	DSNWLM_DEBUGGER	No	No
DB2_INSTALL_JAR	SQLJ	40 - 60	DSNWLM_GENERAL	No	No
DB2_REMOVE_JAR	SQLJ	40 - 60	DSNWLM_GENERAL	No	No
DB2_REPLACE_JAR	SQLJ	40 - 60	DSNWLM_GENERAL	No	No
DB2_UPDATEJARINFO	SQLJ	40 - 60	DSNWLM_GENERAL	No	No
DBG_ENDSESSIONMANAGER	SYSPROC	5 - 20	DSNWLM_DEBUGGER	No	No
DBG_INITIALIZECLIENT	SYSPROC	5 - 20	DSNWLM_DEBUGGER	No	No
DBG_LOOKUPSESSIONMANAGER	SYSPROC	5 - 20	DSNWLM_DEBUGGER	No	No
DBG_PINGSESSIONMANAGER	SYSPROC	5 - 20	DSNWLM_DEBUGGER	No	No
DBG_RECVCLIENTREPORTS	SYSPROC	5 - 20	DSNWLM_DEBUGGER	No	No
DBG_RUNSESSIONMANAGER	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
DBG_SENDCLIENTCOMMANDS	SYSPROC	5 - 20	DSNWLM_DEBUGGER	No	No
DBG_SENDCLIENTREQUESTS	SYSPROC	5 - 20	DSNWLM_DEBUGGER	No	No
DBG_TERMINATECLIENT	SYSPROC	5 - 20	DSNWLM_DEBUGGER	No	No
DEBUGGERLEVEL	DB2DEBUG	5 - 20	DSNWLM_DEBUGGER	No	No
DESTROY_SESSION	DB2DEBUG	5 - 20	DSNWLM_DEBUGGER	No	No
DSN_WLM_APPLENV	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
DSNAHVPM	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
DSNAIMS2	SYSPROC	40	DSNWLM_GENERAL	Yes	No
GET_CONFIG	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
GET_MESSAGE	SYSPROC	40 - 60	DSNWLM_GENERAL	No	No
GET_REPORT	DB2DEBUG	5 - 20	DSNWLM_DEBUGGER	No	No
GET_SYSTEM_INFO	SYSPROC	1	DSNWLM_NUMTCB1	Yes	No
INSTALL_JAR	SQLJ	40 - 60	DSNWLM_GENERAL	No	No
LIST_SESSION	DB2DEBUG	5 - 20	DSNWLM_DEBUGGER	No	No

DB2-supplied routine	Schema name	NUMTCB	Core WLM environment	APF authorized	Program controlled
MQPUBLISH	DB2MQ1C	? 10	DSNWLM_MQSERIES	No	No
MQPUBLISH	DB2MQ2C	? 10	DSNWLM_MQSERIES	No	No
MQSUBSCRIBE	DB2MQ1C	? 10	DSNWLM_MQSERIES	No	No
MQSUBSCRIBE	DB2MQ2C	? 10	DSNWLM_MQSERIES	No	No
MQUNSUBSCRIBE	DB2MQ1C	? 10	DSNWLM_MQSERIES	No	No
MQUNSUBSCRIBE	DB2MQ2C	? 10	DSNWLM_MQSERIES	No	No
PUT_COMMAND	DB2DEBUG	5 - 20	DSNWLM_DEBUGGER	No	No
QUERY_SESSION	DB2DEBUG	5 - 20	DSNWLM_DEBUGGER	No	No
REMOVE_JAR	SQLJ	40 - 60	DSNWLM_GENERAL	No	No
REPLACE_JAR	SQLJ	40 - 60	DSNWLM_GENERAL	No	No
SOAPHTTPC	DB2XML	? 10	DSNWLM_WEBSERVICES	No	No
SOAPHTTPNC	DB2XML	? 10	DSNWLM_WEBSERVICES	No	No
SOAPHTTPNV	DB2XML	? 10	DSNWLM_WEBSERVICES	No	No
SOAPHTTPV	DB2XML	? 10	DSNWLM_WEBSERVICES	No	No
SQLCAMESSAGE	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLCOLPRIVILEGES	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLCOLUMNS	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLFOREIGNKEYS	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLFUNCTIONCOLS	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLFUNCTIONS	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLGETTYPEINFO	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLPRIMARYKEYS	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLPROCEDURECOLS	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLPROCEDURES	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLSPECIALCOLUMNS	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLSTATISTICS	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLTABLEPRIVILEGES	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLTABLES	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SQLUDTS	SYSIBM	40 - 60	DSNWLM_GENERAL	No	No
SYSTS_CREATE	SYSPROC	10 - 40	DSNWLM_GENERAL	No	No
SYSTS_DROP	SYSPROC	10 - 40	DSNWLM_GENERAL	No	No
SYSTS_ENCRYPT	SYSFUN	2 - 8	DSNWLM_JAVA	No	No
SYSTS_RESTORE	SYSPROC	10 - 40	DSNWLM_GENERAL	No	No
SYSTS_START	SYSPROC	10 - 40	DSNWLM_GENERAL	No	No
SYSTS_STOP	SYSPROC	10 - 40	DSNWLM_GENERAL	No	No

DB2-supplied routine	Schema name	NUMTCB	Core WLM environment	APF authorized	Program controlled
SYSTS_TAKEOVER	SYSPROC	10 - 40	DSNWLM_GENERAL	No	No
SYSTS_UPDATE	SYSPROC	10 - 40	DSNWLM_GENERAL	No	No
WLM_SET_CLIENT_INFO	SYSPROC	> 1	DSNWLM_GENERAL	No	No

Installation, configuration, validation, and servicing of these various objects was largely a user-managed effort with DB2 9 that can be summarized as follows:

1. Create a new WLM environment or identify an existing one that is suitable for running the routine.
2. Define a system security environment, if necessary, for the routine.
3. Create and grant access to the routine.
4. Create, populate, and grant access (as applicable) to supporting objects, such as created global temporary tables, databases, table spaces, tables, indexes, views, and aliases.
5. Bind a package, if necessary, for the routine.

Configuration of a routine refers to associating it with the correct WLM environment, granting access to the appropriate authorization IDs, and assigning the package to the appropriate owner.

Validation of a routine refers to a checkout process to determine whether the routine is installed and configured successfully, and ready for use. Otherwise, validation provides diagnostics to help isolate the problem.

Servicing refers to specific actions required after applying a PTF that affects a routine: Such actions can vary from binding a fresh package for an existing routine to installing and configuring a new routine that is being added in the service stream.

For more information, refer to *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7604.

With DB2 10, the installation process takes over most of the task by providing several new install functions:

- A new program for installing and servicing DB2-supplied routines according to user-specified configuration parameters
- A new program for validating that a DB2-supplied routine is correctly installed and configured
- New jobs for calling these programs
- A description of a core set of WLM environments, collectively suitable for running all DB2-supplied routines
- DB2-supplied address space procedures for these environments
- New installation panels for specifying routine configuration parameters
- Premigration queries and reports for identifying the current WLM environment, execute access list, and package owner (if any) for each routine

For more information about DB2-supplied routines and a summary chart that cross references each DB2-supplied routine to one of a minimal set of WLM environments, refer to *DB2 10 for z/OS Installation and Migration Guide,* GC19-2974.

Archived

Some routines have a runtime dependency on other products and features such as DB2 JDBC/SQLJ or IBM WebSphere MQ. Successful installation and configuration of such software dependencies, if required, is assumed by this function.

DB2 10 provides 11 core WLM environments for running DB2-supplied routines. The default names are listed in Table 12-15.

Table 12-15 DB2 default WLM environments

WLM environment name	Purpose
DSNWLM_GENERAL	For most DB2-supplied routines
DSNWLM_PGM_CONTROL	For DB2-supplied routines that require program control and do not run on systems where BPX.DAEMON.HFSCTL is defined
DSNWLM_NUMTCB1	For DB2-supplied routines that need to run serially. A routine that must run serially cannot share a task control block with another routine.
DSNWLM_UTILS	For DB2-supplied routines for utilities
DSNWLM_XML	For most DB2-supplied XML processing routines
DSNWLM_JAVA	For DB2-supplied Java routines
DSNWLM_REXX	For DB2-supplied REXX stored procedures
DSNWLM_DEBUGGER	For DB2-supplied routines that are for the Unified Debugger tool
DSNWLM_DSNACICS	For the DB2 CICS transaction processor routine SYSPROC.DSNACICS
DSNWLM_MQSERIES	For DB2 MQSeries functions
DSNWLM_WEBSERVICES	For DB2 Web Services functions

Example 12-14 shows the output of step DSNWLMU of job DSNTIJRW. This step defines the WLM environment for utilities stored procedures. The job defines a total of 11 standard WLM environments. You can customize this job for your particular requirements.

Example 12-14 Sample output from DSNTIJRW

```
DSNT023I DSNTWLMB ADD WLM APPLICATION ENVIRONMENT DSNWLMDB8A_UTILS SUCCESSFUL

 APPLICATION ENVIRONMENT NAME : DSNWLMDB8A_UTILS
 DESCRIPTION                  : DB2-SUPPLIED WLM ENVIRONMENT
 SUBSYSTEM TYPE               : DB2
 PROCEDURE NAME               : DB8AWLMU
 START PARAMETERS             : DB2SSN=&IWMSSNM,APPLENV='DSNWLMDB8A_UTILS'

 STARTING OF SERVER ADDRESS SPACES FOR A SUBSYSTEM INSTANCE:
 (x) MANAGED BY WLM
 ( ) LIMITED TO A SINGLE ADDRESS SPACE PER SYSTEM
 ( ) LIMITED TO A SINGLE ADDRESS SPACE PER SYSPLEX
```

The last step of this job activates the WLM environment changes. If the program is not in an APF authorized library, the step fails with a return code of 8 and a message as indicated in Example 12-15.

Example 12-15 Program DSNTWLMB needs to be APF-authorized to activate WLM changes

```
DSNT024I DSNTWLMB QUERY SERVICE POLICY FAILED - PROGRAM IS NOT APF-AUTHORIZED
```

Two jobs install and validate the installation of DB2-supplied routines with DB2 10. These jobs are configured with the options that you specify on the installation panels.

- The first job is DSNTIJRT, described in 12.13.1, “Deploying the DB2-supplied routines when installing DB2 10 for z/OS”.
- The second job is DSNTIJRV, described in 12.13.2, “Validating deployment of DB2-supplied routines” on page 529.

12.13.1 Deploying the DB2-supplied routines when installing DB2 10 for z/OS

DB2-supplied routines are deployed as part of installing DB2. A newly-installed DB2 10 for z/OS starts in NFM, which means that all DB2 10 for z/OS-routines supplied can be deployed together. The following tasks are included:

- Create the address space start procedures for all WLM environments that are used by DB2-supplied routines
- Create environment option files
- Create routines and supporting databases
- Bind packages for the routines
- Grant access to the routines
- Validate that the routines are working successfully

In DB2 10, installation job DSNTIJMV is enhanced to create the startup procedures for the WLM environment address spaces, in addition to the usual role of creating the DB2 address space startup procedures. DSNTIJMV is also enhanced to create the environment option files. Routines are created and bound after DB2 is started.

In DB2 10, the DSNTIJRT job handles all DB2-supplied routines. This job provides a configuration control statement for each DB2-supplied routine and processes those statements. The configuration control statements specify the WLM environment, a list of authorization IDs that are allowed to execute the routine, and optionally the package owner for the routine. The control statements are validated and used to create and customize the routines and supporting objects.

In DB2 10, DSNTIJRT is identically configured, regardless of whether DB2 is freshly installed or migrated and regardless of the DB2 mode. DSNTIJRT is always run or rerun from the top, and has the same basic process for each routine that is for use in the current DB2 mode:

- Optionally drop the routine and any supporting objects.
- Creates the routine if it does not exist.
- Creates any user-managed databases for it that do not already exist.
- Binds packages as needed, that is if a package is missing or downlevel (according to the consistency token of its DBRM), or has been invalidated.
- Grants access to it.

If a routine requires a higher DB2 mode, these steps are bypassed automatically for that routine until DSNTIJRT is rerun after DB2 enters NFM.

DSNTIJRT processing by default does not drop an existing routine (and any supporting database) prior to create, but offers the option to do so globally (all routines and databases

are dropped up front). Attempting to drop an object that does not exist is reported only as informational.

12.13.2 Validating deployment of DB2-supplied routines

After a DB2-supplied routine is deployed, validate it to ensure that it is ready for use. In DB2 10, a common validation process is driven by a the DSNTIJRV installation job. This job validates the DB2-supplied routines by executing program DSNTRVFY. Validate the routines after DB2 installation and after each phase of migration. You can use DSNTRVFY to validate a single routine, multiple routines, or all DB2-supplied routines. DSNTRVFY produces a report that identifies routines that passed the validation and possible problem areas for routines that failed.

DSNTIJRV is run after successful completion of DSNTIJRT, whether DSNTIJRT is run for a fresh DB2 installation, for migration to conversion mode, or after DB2 enters NFM.

After the systems programmer defines all needed WLM environments, the security administrator creates all needed security environments, the systems programmer successfully runs DSNTIJRT, and DSNTIJRV is customized by the installation CLIST, the following actions validate the deployment of the DB2-supplied routines:

1. The systems programer runs DSNTIJRV to validate installation of the DB2-supplied routines.

   DSNTIJRV performs the same basic process for each routine:

   – If the routine requires a higher DB2 mode, an informational message to that effect is printed and the job moves to the next routine.
   – If a non-critical routine is not registered to DB2, a warning message to that effect is printed and the job moves to the next routine.
   – If a critical routine is not registered to DB2, an error message to that effect is printed and the job move to the next routine.
   – If a routine's parameters are incorrect, an error message to that effect is printed and the job moves to the next routine.
   – If a non-critical routine cannot be called successfully, a warning message to that effect is printed and the job moves to the next routine.
   – If a critical routine cannot be called successfully, an error message to that effect is printed and the job moves to the next routine.
   – The return code is set accordingly to whether only warnings were received, errors were received or everything was successful.

   DB2 starts WLM address spaces for routines that are validated and returns diagnostics.
2. The systems programmer reviews the results, corrects an problems, and optionally reruns DSNTIJRV.

12.14 Eliminating DDF private protocol

Since DB2 V5, private protocol has not been enhanced and the DRDA functions provided are equivalent or better. For details about function and migration steps, see *DB2 9 for z/OS: Distributed Functions*, SG24-6952.

Here is a list of enhancements to DRDA during the last two releases of DB2:

- DRDA enhancements implemented with DB2 9:
  - DB2 can use IPv6 only with DRDA.
  - DDF supports running the address space in AMODE 64, which relieves the storage constraints that exist in Version 8. The performance of TCP/IP has improved because the communications buffer is in 64-bit shared memory. This change means that you do not have to copy the buffer into your own address space or into the DBM1 address space. Reply communications are built and defined in 64-bit shared memory and can be pointed to instead of copied into your own address space or into the DBM1 address space.
  - DB2 can use trusted application servers with DRDA.
  - The performance of LOB retrieval is improved.
  - VTAM® independence allows you to set up a DB2 DDF to support TCP/IP and DRDA access.
  - DB2 has a new DRDA secure port, which allows use of SSL authentication methods. SSL authentication methods support secure communications between a client and DB2.
- DRDA enhancements that come with DB2 10 for z/OS:
  - DB2 uses new DRDA capabilities to encode DRDA metadata in Unicode instead of EBCDIC, thereby eliminating the need for character conversion from EBCDIC to Unicode.
  - New monitoring capabilities in DRDA enable monitoring tools to track the execution of statements and to track the sequence of transactions in the distributed environment.
  - DB2 supports the following new DRDA user data type capabilities:
    - TIMESTAMP data (SQLTYPE 392/393) with extended timestamp precision
    - TIMESTAMP WITH TIME ZONE data type (SQLTYPE 2448/2449)
  - DB2 implements new DRDA capabilities that support various new DB2 10 SQL functions. If new DRDA capabilities are required to support the new DB2 10 SQL functionality, then the requester system must be at the correct DRDA level to use the new SQL function. These new capabilities include support for the PREPARE ATTRIBUTES clause CONCENTRATE STATEMENTS WITH LITERALS.
  - DRDA always resolves aliases during a PREPARE of a dynamic SQL statement or binding of a static statement into a package at a remote location.

Thus, if you are using private protocol, you are not taking advantage of these enhancements and the many more that are implemented since DB2 V4.

DB2 V8 and DB2 9 tools can help you identify plans and packages that are using private protocol and allow you the opportunity to convert them to DRDA. Private protocol was officially deprecated in DB2 9 and is eliminated in DB2 10.

If you have not yet converted your plans and packages to DRDA and are preparing to migrate to DB2 10, there are additional capabilities to further ease the work required to convert to DRDA. We recommend that you use the tools provided in DB2 V8 and DB2 9 to convert to DRDA. It is best if you explicitly rebind all plans and packages that were bound with DBPROTOCOL(PRIVATE) to use DBPROTOCOL(DRDA), define the required aliases, execute the required remote binds, and cleanup hard coded applications. This method provides the cleanest environment for you to run with and eliminates any issues related to a conversion from private protocol to DRDA prior to rolling out a new release of DB2 (in this case, DB2 10).

Archived

DB2 10 has the built in capability that as plans and packages are loaded to identify plans and packages that were bound with DBPROTOCOL(PRIVATE) but are not dependant on a remote location (do not actually run SQL at a remote location), to be auto-rebound with DBPROTOCOL(DRDA). However, those plans that are identified as truly needing private protocol, or that DB2 cannot accurately determine if they run SQL at a remote location, fail execution.

Any package that was last bound in a release of DB2 prior to DB2 10 with the DBPROTOCOL(PRIVATE) option is now allocated and allowed to run in DB2 10. If the application executing one of these packages only uses sections of the package which access objects local to the DB2 subsystem and there are no issues with accessing the local objects, the application should run successfully without any SQLCODE errors.

When an application attempts to either execute a statement section which either might require an incremental bind or execute a prepare/execute immediate of a statement, and it is determined that the statement must access a remote location and the package was last bound in a release of DB2 prior to DB2 10 with the DBPROTOCOL(PRIVATE) option, the application receives SQLCODE -904 with tokens: reason code 00E3001E, resource type x'801', and resource name LOCATION.COLLECTION.PACKAGE.CONTOKEN.

If a PLAN with member DBRMs is run in DB2 10, it undergoes an automatic rebind to convert it to a plan having a PKLIST only and the member DBRMs are bound into packages. Because DB2 10 supports DRDA only, the autobind uses the DBPROTOCOL(DRDA) bind option regardless of the DBPROTOCOL option value used during the previous BIND/REBIND of the PLAN. This means that the PLAN is rebound with DBPROTOCOL(DRDA) and the created packages are all bound with DBPROTOCOL(DRDA). If the PLAN does not have member DBRMs, then the packages referenced by the PLAN run as described previously.

Finally, DSNTP2DP and its DSNTIJPD installation job are added to DB2 10. DSNTP2DP now only generates the commands to convert any package or plans to use DRDA protocol that were last bound with DBPROTOCOL(PRIVATE) and have sections with a remote location dependency, have dynamic SQL support, or the package was bound from a remote requester.

This addition allows users with large numbers of packages bound with DBPROTOCOL(PRIVATE) to delay from having to immediately convert these to DRDA. The goal here is to let those packages that do not actually try to run SQL at a remote location to continue to work without an explicit BIND/REBIND being done.

In DB2 10, private protocol is no longer supported. Plans and packages that were bound with DBPROTOCOL(PRIVATE) and access remote locations can exist in DB2 10, but attempts to load or run those plans or packages will cause the local section to run, but any remote reference will fail. DB2 provides a REXX tool (DSNTP2DP) for preparing a subsystem to use DRDA access only. For details about how to run this tool, refer to *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974.

12.15 Precompiler NEWFUN option

The precompiler NEWFUN option indicates whether the DB2 precompiler accepts SQL that requires NFM in the current release. The traditional settings are YES or NO: NEWFUN(YES) means it accepts SQL that requires new function, while NEWFUN(NO) means it does not. If the NEWFUN option is not specified explicitly, the precompiler uses the setting of the NEWFUN parameter in the DSNHDECP module it has loaded. In DSNHDECP, the valid settings are the same, YES or NO and the default is YES.

In consideration of skip-release migration to DB2 10, SQL processing options NEWFUN(YES) and NEWFUN(NO) options are deprecated, DB2 10 introduces version-specific (absolute) settings of Vnn (that is NEWFUN(V10), NEWFUN(V9), NEWFUN(V8)) for the NEWFUN precompiler option.

Starting with DB2 10, the absolute settings are preferred; however, NO and YES are still accepted. In DB2 10, NEWFUN(NO) means the same as NEWFUN(DB2 9), which means that the precompiler accepts all SQL that is valid in DB2 9 NFM but not SQL that requires DB2 10 NFM. NEWFUN(YES) means the same as NEWFUN(DB2 10), which means that the precompiler accepts all SQL that is valid in DB2 10 NFM.

Beginning in DB2 10, the DSNHDECP module also uses (and prefer) absolute settings for the NEWFUN parameter. DSNHDECP remains the same in DB2 V8 and DB2 9 because there is no skip-release migration consideration for those releases.

In job DSNTIJUZ DSNHDECP.NEWFUN is set by default to the current version, Vnn. In DB2 10, if the installation CLIST is run in MIGRATE mode, the NEWFUN setting is changed to V8 or V9 accordingly to indicated migration-from release. In other words, if you are migrating form DB2 V8, NEWFUN is set to V8 and if you are migrating from DB2 9 NEWFUN is set to V9. In job DSNTIJNG, DSNHDECP.NEWFUN is always set to the current version, Vnn.

The DSNH CLIST, precompiler and DSNTPSMP external SQL procedure processor accept and recognize the new version-specific settings from DSNHDECP.NEWFUN.

Archive

13

Performance

DB2 10 for z/OS delivers value by improving performance and efficiency. Compared to previous DB2 versions, users might achieve a 5% to 10% CPU savings for traditional workloads, with up to 20% CPU savings for non-traditional workloads. Synergy with IBM System z platform components reduces CPU use by taking advantage of the latest processor improvements, larger amounts of memory, solid-state disk, and z/OS enhancements.

In this chapter, we describe performance enhancements in DB2 10. Many of these improvements are available by migrating to DB2 10 and rebinding. More information is provided in the planned *DB2 10 for z/OS Performance Topics*, SG24-7942.

In this chapter, we discuss the following topics:

- Performance expectations
- Improved optimization techniques
- Dynamic prefetch enhancements
- DDF enhancements
- Miscellaneous DDF performance improvements
- Dynamic statement cache enhancements
- INSERT performance improvement
- Referential integrity checking improvement
- Buffer pool enhancements
- Work file enhancements
- Support for z/OS enqueue management
- LOB enhancements
- Utility BSAM enhancements for extended format data sets
- Performance enhancements for local Java and ODBC applications
- Logging enhancements
- Hash access
- Additional non-key columns in a unique index
- DB2 support for solid state drive
- Extended support for the SQL procedural language
- Preemptable backout
- Eliminate mass delete locks for universal table spaces
- Parallelism enhancements
- Online performance buffers in 64-bit common
- Enhanced instrumentation

© Copyright IBM Corp. 2010. All rights reserved.

- Enhanced monitoring support

Archived

13.1 Performance expectations

DB2 10 builds on the solid foundation provided by Version 8 and Version 9 of DB2 for z/OS by continuing to focus on performance, availability, and scalability while reducing total cost of ownership. Historically, new releases of DB2 since DB2 Version 2 have shown possible CPU regression of up to 5% from release to release, which is necessary to change the infrastructure to support new features and functions. The degree of regression varies from workload to workload. Such regression is often mitigated or overcome when packages are rebound or when a group restart is done and sometimes requires new-function mode (NFM).

CPU regression in DB2 Version 8 was higher because Version 8 moved from 32-bit architecture to 64-bit architecture and Unicode and introduced support for long names (up to 128 bytes) to improve the portability of applications to and from other platforms.

DB2 9 reduced CPU regression through significant reduction in utility CPU usage and through improvements in virtual storage constraint relief.

Figure 13-1 shows the historical trend of CPU performance regression goals for transaction workloads since DB2 Version 3. The figure illustrates how the goals for DB2 10 10 are higher than other versions.

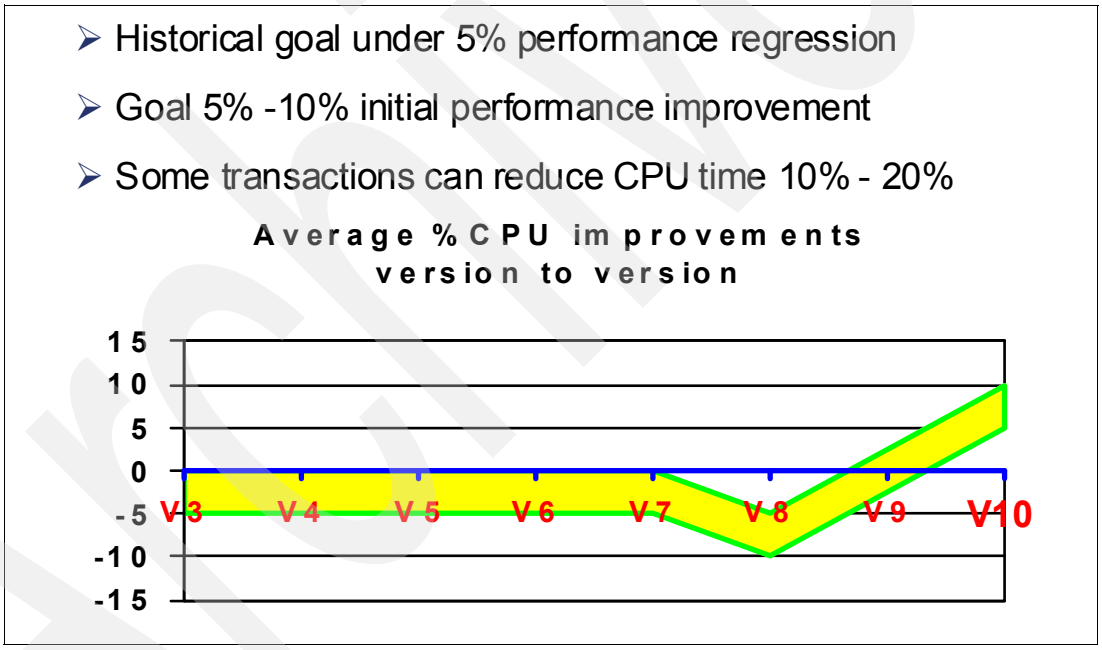


Figure 13-1 DB2 10 performance objective

DB2 10 offers as much as a 5% to 10% CPU reduction in conversion mode (CM) with a simple rebind or bind. These CPU savings are expected for traditional OLTP-type applications. Applications that use functions such as distributed processing, ODBC, and SQL procedures can see a greater CPU reduction.

Workloads using native SQL procedures can see a CPU reduction of up to 20% after you DROP/CREATE or REGENERATE the SQL procedures. Concurrent sequential insert workloads can see a 5% to 40% CPU reduction, depending on the type of table space. Some query workloads can see as much as a 20% CPU reduction without access path changes. You might also expect to see even more CPU reduction with positive access path change.

The CPU reduction is due to several product enhancements that apply to DB2 10 CM and that rebind or bind applications and regenerate SQL procedure or procedures. The following performance-oriented enhancements are available in DB2 10 CM:

- Runtime access path enhancements
- Index I/O parallelism and general sequential insert enhancement
- Improved dynamic prefetch and list prefetch used for indexes
- More query parallelism and, therefore, more query zIIP exploitation
- More streamlined code and general path length reduction
- Buffer pool system paging enhancements
- DDF optimization and RELEASE(DEALLOCATE) use
- Logging enhancements in terms of I/O reduction and buffer fixing
- Synergy with new hardware
- Exploit new hardware instructions (z10, z196)

The biggest performance improvements in DB210 are for those workloads with high-volume concurrent transactions with high logical resource contention, such as latch.

In addition, all DB2 prefetch engines and deferred write SRBs are now zIIP eligible, as is the Runstats utility. XML schema validation can redirect to both zIIP and zAAP specialty engines.

Changes to the storage mapping in the DBM1 address space reduce the storage requirements below the bar by 75% or more. Virtual storage constraint within the DBM1 address space is no longer an issue. In fact, DB2 10 can process as much as 5 to 10 times the current thread workload. There might be opportunity for you to consolidate current workloads into fewer DB2 data sharing members or DB2 subsystems that are subject to ECSA storage consumption verification.

The following performance improvements require more setup, such as the setting of new parameters in NFM:

- SQL/PL improvements
- Additional columns in index
- MEMBER CLUSTER universal table spaces
- Hash access
- Inline LOBs
- Utility improvements
- Access to currently committed data

DB2 10 also improves support for performance monitoring and problem diagnostic tooling and profile table support for system level monitoring of threads and connections. In our testing, we used primarily IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS V5.1(OMEGAMON PE V5.1). DB2 provides enhanced problem determination and performance monitoring and increased granularity of monitoring for system-level activities.

13.2 Improved optimization techniques

Each release of DB2 brings advances in optimizer technology. In this section, we discuss the following enhancements to DB2 10:

- Safe query optimization
- RID pool enhancements
- Range-list index scan
- IN-LIST enhancements
- Aggressive merge for views and table expressions
- Improvements to predicate processing

- Sort enhancements

13.2.1 Safe query optimization

A purely cost-based optimization process might not generate the optimal access plan due to a number of reasons, for example:

- Invalid query optimization assumptions, like RANGE predicate default selectivity
- Unpredictable runtime environment, such as concurrent RID pool usage

The DB2 optimizer can sometimes overestimate the filtering of a predicate, which can occur if the literal value is not known, for example when the predicate is a range predicate, when data is distributed non-uniformly, or host variables or parameter markers are used.

The following value can qualify 0-100% of a table, depending on the literal provided:

```
WHERE BIRTHDATE < ?
```

In addition, the following value is 99% Y and 1% N, depending on which value you use at run time:

```
WHERE STATUS = ?
```

Prior to DB2 10, the recommended solution was to use the REOPT BIND parameter to allow DB2 to reevaluate access paths based on the actual host variable values. However, this solution is not desirable in all cases because of the reoptimization overhead on each SQL execution.

DB2 10 quantifies the cost uncertainty and incorporates it into the cost-based optimizer during access path selection, especially index selection. When two indexes have a close cost estimate, the DB2 optimization process considers the uncertainty of matching and screening predicates when choosing the most efficient index. The DB2 optimization process might choose an index that has a slightly higher cost estimate if that index has a lower cost certainty.

You can take advantage of these safer optimization techniques by rebinding or binding applications in DB2 10 CM.

13.2.2 RID pool enhancements

DB2 uses the RID pool for a variety of SQL access plans, including list prefetch and hybrid join. However when the RID pool becomes full, SQL performance is severely degraded. RID access falls back to a table space scan, and all the work done up to that time is lost, including index filtering. To mitigate these failures, DB2 performs RID pool threshold checking at bind time; however, it cannot completely avoid these types of RID pool failures.

For example, DB2 at bind time does not choose to use any SQL access plan that uses list prefetch processing if it estimates the query returns more than 25% of the rows in a table. However, if the optimizer estimates that fewer than 25% of the rows would match the predicate and then at run time the number of rows returned exceeds 25%, list prefetch is abandoned. This threshold is known as *TERMINATED - EXCEEDED RDS LIMIT*.

Archived

Running RUNSTATS and REBIND: It is always good practice, in DB2 10 and in prior versions of DB2, to regularly run RUNSTATS and REBIND to ensure that statistical information in plans and packages is up to date and to avoid unnecessary RID pool failures. For example, the RDS threshold (25% of the rows in a table) is calculated at BIND time and is stored in the skeleton plan table and skeleton cursor table.
If the number of rows in a table increases significantly, and RUNSTATS and REBIND are not run to reflect these changes, the RDS threshold stored in the skeleton plan table and skeleton cursor table can easily be exceeded in cases where far less than 25% of the rows would be accessed. In those cases DB2 9 reverts to a table space scan unnecessarily.

DB2 10 provides new techniques to better manage the RID pool. DB2 10 allows an access plan to use the RID pool as long as it is the cheapest plan, which allows the optimizer to consider RID list access plans in more cases.

At run time, RIDs overflow to the work file and continue processing with 32 KB sized records, where each record holds the RIDs from one 32 KB RID block. Thus, RID access is less likely to fall back to a table space scan (as with DB2 9). In some cases, you might see work file use increase as a result. Large memory with a large work file buffer pool avoids I/O for the RID blocks. Even if there is not enough memory to avoid work file I/O, work file I/O is far better than paging I/O for an oversized RID pool.

In DB2 9, access reverts to a table space scan if the maximum supported number of RIDs (approximately 26,552,680 RIDS) is exceeded. DB2 10 lifts this restriction but only for those RID lists that are stored in a work file. Thus, DB2 first attempts to use work file storage and falls back to table space scan only if DB2 cannot use work file storage *and* the RID list in the work file successfully.

Although this enhancement is available in CM and although you do not need to rebind or bind applications to have the RIDs overflow to the work files, we advise that you rebind or bind applications to reset the RID thresholds that are stored with the package.

To allow concurrent threads to use this RID pool and more queries to choose RID access where appropriate, the default RID pool size is increased from 8,000 KB in DB 9 to 400,000 KB in DB2 10.

You can use the MAXTEMPS DSNZPARM parameter to set the maximum amount of temporary storage in a work file database that a single agent can use at any given time.

DB2 10 introduces the new MAXTEMPS_RID DSNZPARM parameter specifically to manage work file resources for RID processing.

MAXTEMPS applies to all types of work file usage, such as sort, result set materialization, and declared or created temporary global tables. MAXTEMPS_RID is used specifically to manage RID usage of work file. You can use it to completely disable the work file use for RID list processing, enabling you to use pre-DB2 10 behavior.

The MAX TEMP RID field is on the DSNTIP9 installation panel and specifies the maximum number of RIDs (measured in RID *blocks*) that the subsystem is allowed to store in the work file. If there are more RIDs than what is allowed by this subsystem parameter, RID processing falls back to a table space scan. Each RID block stored in the work file occupies 32 KB of work file storage and contains 6524 RIDs. For example, specifying a value of 10,000 for the MAXTEMPS_RID limits the number of RIDs that are allowed to be stored in the work file to 6,524,000 (about 312.5 MB).

Note that MAXTEMPS controls the maximum amount temporary storage in the work file database that a single agent can use at any given time for any type of usage. This use includes the work file use for storing the RIDs. So, RID list processing can revert back to a table space scan even if the MAXTEMPS_RID value *is not* exceeded but the MAXTEMPS value for that agent *is* exceeded.

The following IFCIDs are enhanced to monitor RID pool processing:

- IFCID 002
- IFCID 003
- IFCID 106
- IFCID 125
- IFCID 148
- IFCID 316
- IFCID 401

13.2.3 Range-list index scan

Consider the case where you have an application that submits a query to DB2 that returns a result set containing 100 rows: however, the application wants to scroll forward or backward based upon the current position on the screen or list. This case is typical for cursor scrolling type applications where the returned result set is only part of the complete result set. These types of queries are typically built from complex OR predicates, such as the following example:

```
(C1 op ? AND C2 op ? AND C3 op ?) OR
(C1 op ? AND C2 op ?) OR
(C1 op ?)
```

For example, scrolling through a list of names, as shown:

```
SELECT ...
FROM EMPLOYEE
WHERE (LASTNAME = ‘Beeblebrox’ AND FIRSTNAME > ‘Zaphod') OR
(LASTNAME > ‘Beeblebrox’)
ORDER BY LASTNAME, FIRSTNAME
FETCH FIRST 21 ROWS ONLY;
```

These types of queries (with OR predicates) can suffer from poor performance because DB2 cannot use OR predicates as matching predicates with single index access. The alternative method is to use multi-index access (index ORing), which is not as efficient as single index access. Multi-index access retrieves all RIDs that qualify from each OR condition and then unions the result.

In our example here, given the current position of Beeblebrox, that means a union with all rows until the remainder of the table. This process fails the RID pool because it is probably 90+% of the table.

The other issue is that the query only wants the next 21 rows, and multi-index access cannot do that filtering. Single-index access allows you to scroll through the index and read only the next 21 rows without requiring RID processing. Prior to DB2 10, DB2 could not use single matching index access for these complex OR conditions.

DB2 10 can process these types of queries with a single index access, improving the performance of these types of queries. This type of processing is known as a *range list index scan*, although some documentation also refer to it as *SQL pagination*.

Returning to the example. assume there is a single index available defined on columns (LASTNAME, FIRSTNAME). To process the query, DB2 logically breaks down the OR predicate into a range list of two ranges. DB2 then performs two index probes:

- The first probe is for (LASTNAME = 'Beeblebrox' AND FIRSTNAME > 'Zaphod'). All remaining Beeblebrox rows are scanned until no more rows qualify the first probe predicate or until the application stops fetching rows. There is no need to sort the result set because the rows are retrieved in the required order through the index.
- If the requirement to fetch 21 rows is not satisfied by the first probe, then the second probe of the index is executed with (LASTNAME > 'Beeblebrox'). A new position is established in the index and scanning continues until either all rows are processed or until the required number of rows are returned to the application.

This access method is shown in PLAN_TABLE by the column ACCESSTYPE='NR' as shown in Table 13-1.

Table 13-1 PLAN_TABLE extract for range list access

QBNO	PLANNO	METHOD	TBNAME	ACTYPE	MC	ACNAME	QBTYPE	MIXOPSEQ
1	1	0	EMPLOYEE	NR	2	INDEX1	SELECT	1
1	1	0	EMPLOYEE	NR	1	INDEX1	SELECT	2

The order of the PLAN_TABLE entries is determined by the coding sequence. However, the determination of execution sequence is deferred to run time when all the host variables or parameter markers are resolved.

Range list index scan is available after you rebind or bind applications in CM. However, DB2 considers a range-list index scan only when the SELECT statement meets the following requirements:

- Every OR predicate refers to the same table.
- Every OR predicate has at least one matching predicate.
- Every OR predicate is mapped to the same index.

However, range list index scan is not supported under the following conditions:

- Residual OR predicate (Stage 2 predicate)
- When the OR predicate includes an in-list
- On the inner table (except the inner table of a sort merge join)
- Within a correlated subquery
- When the index key is being updated
- Index on expression
- Dynamic scrollable cursor
- Rowset cursor (multi-row select)
- Hash access supported

Single index access for complex OR predicates is available in CM after a rebind or bind is complete for static SQL.

13.2.4 IN-LIST enhancements

DB2 10 delivers improvements to IN-list predicate processing using a single index access.

Two of these improvements include accessing the IN-list as an in-memory table if list prefetch is used or if DB2 can match on multiple IN-lists.

This use in shown in the EXPLAIN output in the PLAN_TABLE with a new table type (TABLE_TYPE) of *I* and a new access type (ACCESSTYPE) of *IN*. For example, assuming there is a single index on C1, the EXPLAIN output for the following example returns two rows:

```
SELECT *
FROM T1
WHERE T1.C1 IN (?, ?, ?):
```

In this example, the first row is to access the IN-list table, and the second row is to access the base table, T1. Table 13-2 shows an extract of the PLAN_TABLE.

Table 13-2 PLAN_TABLE extract for IN-list

QBNO	PLANNO	METHOD	TBNAME	ACTYPE	MC	ACNAME	TBTYPE	PFETCH
1	1	0	DSNIN001(01)	IN	0		I	
1	2	1	T1	I	1	T1_IX-C1	T	L

If IN-list predicate is selected as the matching predicate and list prefetch is chosen, the IN-list predicates are accessed as an in-memory table. In the EXPLAIN output in the PLAN_TABLE, this access to the in-memory table is associated with a new table type *I*, and new access type *IN*. DSNIN001(01) is the in-memory table name named by the DB2 optimizer. The IN-list table name uses a simple convention:

- *DSNIN* indicates that it relates to IN-list.
- *001* indicates the IN-list predicate number.
- *(01)* in the middle indicates the query block number.

DB2 further extends the use of IN-list in-memory tables to support multiple IN-list predicates, as shown by expanding our example as follows:

```
SELECT *
FROM T1
WHERE T1.C1 IN (1, 2, 3)
AND T1.C2 IN ('A', 'B', 'C'):
```

If there is an available index on T1 as (C1, C2), then DB2 can choose both IN-list predicates as matching index predicates. The decision is made on cost. Assume that DB2 does choose both of the IN-list predicates as matching index predicates. Then, the IN-list in-memory table can be populated with (1, 'A'), (1. 'B'), (1, 'C'), (2, 'A'), (2, 'B'), (2, 'C'), (3, 'A'), (3, 'B'), (3, 'C').

There is no limit to the number of IN-list predicates that DB2 builds into an in-memory table for processing.

Existing IN-list access (ACCESSTYPE='N') are used if a single IN-list matching is chosen without list prefetch.

Predicate transitive closure support for IN-list predicates is also introduced. DB2 10 generates transitive closure predicates for IN-list predicates, so more access path options can be considered.

For example, the following query:

```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.C1 IN (? ? ?):
```

Becomes:

```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.C1 IN (?, ?, ?):
AND T2.C1 in (?, ?, ?):
```

Now, an index on T2.C1 provides DB2 with more access plan options.

Predicate transitive closure already applies to the following values:

- COL1 op value
  - op is =, <>, >, >=, <, or <=
  - Value is a constant, host variable, or special register
- COL1 (NOT) BETWEEN value1 AND value2
- COL1=COL3

13.2.5 Aggressive merge for views and table expressions

In the context of the following examples, the word *materialize* is defined as "the process of putting rows from a view or nested table expression into a work file for additional processing by a query."

Physical materialization itself is an overhead. In addition, it limits the number of join sequences that can be considered and can limit the ability to apply predicates early in the processing sequence. The join predicates on materialization work files are also not indexable. So, normally avoiding materialization is desirable.

In DB2 10, there are additional areas where materialization can be avoided, particularly for views and table expressions involved in outer joins:

- When there is a IFNULL, NULLIF, CASE, VALUE, or COALESCE expression on the preserved side of an outer join. The exception is when a CASE expression will merge to become a join predicate. For example:

```
SELECT A.C1, B.C1, A.C2, B.C2
FROM T1 ,(SELECT COALESCE(C1, 0) as C1 ,C2
            FROM T2 ) A <--table expression 'A' is merged
          LEFT OUTER JOIN
          (SELECT COALESCE(C1, 0) as C1 ,C2
            FROM T3 ) B <-- B is materialized
          ON A.C2 = B.C2
WHERE T1.C2 = A.C2;
```

  In this case, the subquery becomes:

```
SELECT COALESCE(A.C1, 0) as C1, B.C1, A.C2, B.C2
FROM T1, T2 A
        LEFT OUTER JOIN
               (SELECT COALESCE(C1, 0) as C1 ,C2 FROM T3 ) B
                ON A.C2 = B.C2
WHERE T1.C2 = A.C2;
```

- When there is a subquery in the view or table expression that is on the preserved side of an outer join, DB2 does not materialize so that selective predicates can be applied earlier to reduce the join size. For example:

```
CREATE VIEW V1
AS SELECT C1, C2
FROM T3
WHERE T3.C1 IN (SELECT T4.C1
                FROM T4
                WHERE T4.C2 = T3.C2
                GROUP BY T4.C1);

SELECT T1.C1, T1.C2, T2.C1, T2.C2
FROM V1 AS T1
   LEFT OUTER JOIN
   T2 ON T1.C1 = T2. C1
WHERE (T1.C2 IN ('712', '713', '714'));
```

  After merge, the query becomes:

```
SELECT T3.C1, T3.C2, T2.C1, T2.C2
FROM T3
LEFT OUTER JOIN T2
ON T3.C1 = T2.C1
WHERE T3.C2 IN ('712', '713', '714')
AND  T3.C1 IN
   (SELECT T4.C1
    FROM T4
    WHERE T4.C2 = T3.C2
    GROUP BY T4.C1);
```

- If there is a single table view or table expression on the null padded side that contains a subquery, materialization can be avoided. For example:

```
SELECT *
FROM T1
    LEFT OUTER JOIN
       (SELECT *    <-- table expression contains subquery
        FROM T2
        WHERE T2.C1 = (SELECT MAX(T3.C1) FROM T3 ) <--subquery
        ) TE
ON T1.C1 = TE.C1;
```

  The table expression becomes:

```
SELECT *
FROM T1 LEFT OUTER JOIN T2              <-- table expression is merged
ON T2.C1 = (SELECT MAX(T3.C1) FROM T3)   <-- subquery ON-predicate
AND T1.C1 = TT.C1;
```

  For left or right outer joins, after query transformation, DB2 allows subquery predicates in the on-clause for LEFT or RIGHT outer join. Thus DB2 could merge certain single table views or table expressions on null-padded side which contains a subquery. These views and table expressions must only contain a reference to a single table. DB2 does so by converting the subquery predicate to a *before join* predicate. When the table in the table expression is very large, performance is improved due to lack of materialization.

- When the query contains a correlated table expressions without a GROUP BY, DISTINCT, or column function, DB2 does not materialize the table expression. Instead, the table expression is merged into the parent query block with the query being rewritten.

Archived

For example:

```
SELECT *
FROM T1,
   TABLE(
         SELECT *
         FROM T3 AS T2
         WHERE T1.C1= T2.C1
         ) AS X; <-- table expression X is materialized in V9
```

Is rewritten to:

```
SELECT T1.* , T2.C2
FROM T1, T3 AS T2
WHERE T1.C1 = T2.C2;
```

13.2.6 Improvements to predicate processing

The processing of stage 1 and non-index matching predicates is enhanced. Using path length reduction and technology similar to SPROCs, which were first introduced in DB2 Version 2, DB2 10 processes non-boolean predicates more efficiently when accessing an index and stage 1 data access predicates.

This enhancement does provide significant CPU savings for index access and is available in CM. You do not need to rebind or bind static applications to take advantage of some of these optimization improvements. However, a rebind or bind is required to take full advantage.

All workloads benefit from this enhancement; however, more complex queries with many predicates show higher improvement. Queries that scan large amounts of data also show a higher saving in CPU. Preliminary measurements show an average of 18% CPU reduction (1% through 70%) from a TPC-H-like workload using 140 queries.

13.2.7 Sort enhancements

Assume that you have a large table from which you want to sort only the first 2,000 rows in a particular order. To perform this sort, you can code a SELECT statement using the FETCH FIRST and ORDER BY clauses. However, this sort is done before the fetch, which causes a large sort for no reason. The workaround is to code this sort using a temporary table, which is a lot more work than coding a simple select.

DB2 9 provides relief with the support for FETCH FIRST N ROWS with ORDER BY (also in subselects), where the tournament tree sort is avoided for small N. With the small N, you do not need to sort the entire answer set. Only the top N needs to be sorted. ORDER BY now exploits FETCH FIRST n ROWS so that work files are not created (less I/O). This DB2 9 enhancement is conditional on (N * (sort key + data)) < 32 KB.

DB2 10 further extends this support to (N * (sort key + data)) < 128 KB.

DB2 9 also avoids allocating work files for small sorts for final sorts only, provided that no greater than 255 rows are to be sorted and the result is < 32 KB (sort key + data). DB2 10 extends sort avoidance for small sorts to intermediate sorts, except for parallelism or SET functions.

Finally, DB2 10 introduces a hashing technique for managing large sorts, which potentially reduces the number of merge passes that are needed to complete these sorts. Hashing can be used to identify duplicates on input to sort if functions such as DISTINCT or GROUP BY are specified.

13.3 Dynamic prefetch enhancements

DB2 Version 2.3 introduced dynamic prefetch. DB2 9 included the following changes to prefetch:

- Sequential prefetch is enabled only for table space scans.
- For SQL access, the maximum prefetch quantity increased from 128 KB (32 pages) with DB2 V8 to 256 KB (64 pages) with DB2 9 for sequential prefetch and LOB-related list prefetch for table spaces and indexes that are assigned to 4 KB buffer pools. In DB2 10, the maximum prefetch quantity is still 32 pages for dynamic prefetch and non-LOB list prefetch.
- For utility access to objects assigned to 4 KB buffer pools, the prefetch quantity increased from 256 KB (64 pages) with DB2 V8 to 512 KB (128 pages) with DB2 9 in a utility. These values also do not change in DB2 10.

The following types of DB2 for z/OS prefetch are available:

Sequential	Selected at SQL statement bind time. At statement execution time, as soon as the target table space is accessed, two prefetch quantities of sequential pages are read into the buffer pool. Additional requests for a prefetch quantity of pages are issued each time a *trigger page* is accessed by the executing SQL statement. A trigger page is one that is a multiple of the prefetch quantity. DB2 therefore tries to stay at least one prefetch quantity of pages ahead of the application process.
List	RIDs for qualifying rows (per a query's predicates) are obtained from one or more indexes. In most cases, DB2 sorts the RIDs, then issues asynchronous multi-page read requests against the base table space.
Dynamic	DB2 determines at query run time that the pattern of page access for a target table or a utilized index is *sequential enough* to warrant activation of prefetch processing. If the page access pattern subsequently strays from sequential enough, prefetch processing is turned off. It is turned on again if sequential enough access resumes.

DB2 10 further enhances prefetch processing, particularly with index access, as follows:

- Index scans using list prefetch
- Row level sequential detection
- Progressive prefetch quantity

These enhancements are available in DB2 10 CM without the need for a rebind or bind.

13.3.1 Index scans using list prefetch

An index scan reads the leaf pages of the index in key sequence order. DB2's sequential detection algorithm detects whether the leaf pages are physically organized well enough to kick off dynamic prefetch. If the sequential detection algorithm discovers that the leaf pages are disorganized, then DB2 9 performs synchronous I/Os. When doing sequential I/O, dynamic prefetch reads 128 KB per I/O, that is 32 pages if the page size is 4 KB. Thus, there is a big incentive with DB2 9 to keep indexes organized.

However, DB2 10 greatly mitigates any performance issues of a disorganized index.

DB2 10 can use list prefetch I/O for the leaf pages. How does DB2 know which leaf pages to read? DB2 10 uses the N-1 level of the index to locate the leaf pages, whereas DB2 9 did not issue getpages for the non-leaf pages when doing an index scan. In a typical DB2 10 index

scan, there are a few synchronous I/Os executed while DB2 discovers that the index is disorganized. Thereafter, there is an occasional synchronous I/O for a non-leaf page, but the leaf pages are read using list prefetch. Periodically, DB2 might also revert back to its sequential detection algorithm in case that only a subset of the index is disorganized.

So, an index scan can begin with an index probe to position someplace in a leaf page, but then it scans forward (or backwards) using a combination of dynamic and list prefetch. The enhancement to use list prefetch for disorganized indexes applies to index-to-data access the same as it does for index-only access. List prefetch on index leaf pages can greatly reduce the synchronous I/O waits for long running queries accessing disorganized indexes.

Utilities that need to access index leaf pages to extract key values, including REORG INDEX, CHECK INDEX, and RUNSTATS, can also benefit. Partition-level LOAD and REORG benefit also. Dynamic prefetch and list prefetch are also used for updating the nonpartitioning index (NPI), which happens to perform an index scan.

Finally, with this enhancement, there might be less of a need to run REORG, because your application performance might be able to tolerate accessing data that is more disorganized than previous versions of DB2.

Preliminary performance results show a 2 to 6 times elapsed time improvement with simple SQL statements and a small key size using list prefetch compared to synchronous I/Os.

13.3.2 Row level sequential detection

First, let us review the sequential detection algorithm used by dynamic prefetch in DB2 9.

When the second page in the target table space or index is accessed, DB2 checks to determine whether it is within half of the prefetch quantity forward or backwards of the first page (that is, 16 pages if the prefetch quantity is 32 pages, which is an arbitrary definition that has not changed in DB2 10). If it is within half, that second page is noted as being page-sequential relative to the first. When the third page is accessed, DB2 checks to see that it is within half a prefetch quantity forward the second page. If it is, the third page is noted as being page-sequential with respect to the second page.

DB2 keeps a running counter that is the number of page-sequential getpages among the last eight getpages. If five out of the last eight getpages are page-sequential, a getpage can trigger prefetch, but only if the following conditions are true:

- The getpage is in the *prefetch window*.
- The counter is greater than 4.
- The getpage is page-sequential.

Archive

Normally the prefetch window is the last set of pages to be prefetched, see Figure 13-2. Thus, the last prefetch I/O read pages N through N+P[1]-1, then a getpage for one of those pages can trigger a prefetch I/O starting at page N+P. Only the first prefetch I/O is an exception. Because the first prefetch of pages N through N+P-1 is triggered by the getpage of page N, the next prefetch is triggered by a getpage in the range N+P/2 to N+P-1.

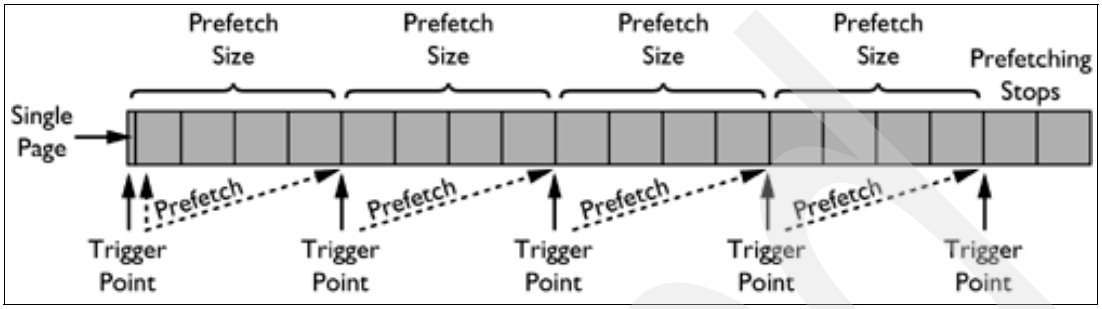


Figure 13-2 Prefetch window

Now let us consider a typical sequence of page numbers for successive RIDs:

100,100,100,***800***,***100***,100,***500***,***100***,100,***101***,101,***300***,***101***,101,101,101,101,101,***300***,***101***,
101,***102***,***400***,***102***,102,102,102,102,102,102,102,***103***,103,103,***200***,***103***,103,103,103,103

The page numbers that are highlighted in bold font are page-sequential and the page numbers highlighted in italic font are non-page-sequential. The prefetch detection counter is incremented when a getpage is page-sequential and is decremented when a getpage is not page-sequential. DB2 9 remembers only the previous page number and not the previous page number. Thus, when DB2 9 returns from page 800 back to page 100, the getpage for page 100 decrements the counter (unless it is already zero).

The other page numbers do not cause a getpage because those RIDs are on the same page as the previous RID.

Each time that DB2 bounces from a clustered page to a non-clustered page, the prefetch counter is decremented not just once, but twice. Therefore, if there is at least one random RID for each clustered page, the prefetch counter never rises over one and prefetch is never triggered, as shown in the sequence of RIDs.

DB2 10 solves this issue by switching from page level sequential detection (PLSD) to row level sequential detection (RLSD).

Consider the same sequence of page numbers for successive RIDs:

100,***100,100***,800,***100,100***,500,***100,100,101,101***,300,***101,101,101,101,101,101***,300,***101,***
101,102,400,***102,102,102,102,102,102,102,102,103,103,103***,200,***103,103,103,103,103***

Now, when the next RID is on the same page as the previous RID, the prefetch counter is incremented. When the flow returns back to a clustered page, the prefetch counter is incremented again. Thus, with RLSD, it is easy to trigger prefetch. However, prefetch is never triggered when advancing to the next row in the same page, because only a getpage can trigger a prefetch. Using the same sequence of RIDs, prefetch is triggered when DB2 issues a getpage for page 101.

Thus, RLSD encompasses the following distinct changes:

- DB2 remembers the previous two page numbers, not just the previous page number.
- Rows are counted, not just getpages.

[1] P is the prefetch quantity.

13.3.3 Progressive prefetch quantity

Because RLSD is based on rows, not pages, it is possible to trigger prefetch more quickly, which can cause DB2 to use more buffers for a query that accesses just a few pages. DB2 10 introduces the concept of a *progressive dynamic prefetch quantity*. The initial prefetch quantity is 32 KB, with the exception of 32 KB pages that prefetch 64 KB. The second prefetch I/O reads 64 KB, with the exception of 32 KB pages that uses 128 KB. Subsequent prefetch I/Os read 128 KB, as in DB2 9.

The definition of a *page-sequential* remains unchanged. If a getpage accesses a page that is within 64 KB of the previous page, it is counted as page-sequential, even if the prefetch quantity starts out at 32 KB.

The progressive prefetch quantity applies to indexes and data, even though RLSD does not affect indexes. So, any type of query can benefit from this enhancement.

13.4 DDF enhancements

DB2 10 includes a number of enhancements to DDF processing to improve the overall performance of distributed workload:

- High performance DBATs are introduced to make it more palatable to bind distributed packages with the option RELEASE(DEALLOCATE).
- Together with other optimizations between the DIST address space and the DBM1 address space, DB2 10 performs an implicit close between the DIST address space and the DBM1 address space when only one row is to be returned.

For more information, see Chapter 9, “Connectivity and administration routines” on page 309.

In this section we describe:

- The RELEASE(DEALLOCATE) BIND option
- Miscellaneous DDF performance improvements

13.4.1 The RELEASE(DEALLOCATE) BIND option

A significant expense in distributed pooled threads is found to be in package allocation and deallocation. In addition, excessive lock timeouts occur with DDF threads holding locks that have been used to execute packages that are bound with RELEASE(DEALLOCATE) or KEEPDYNAMIC(YES). These page set and package locks make it difficult for other processes, such as BINDs and DDL, to break in.

DB2 V6 changed to the behavior of packages bound with RELEASE(DEALLOCATE) so that all package allocations for DRDA DBATs become RELEASE(COMMIT) regardless of the package’s RELEASE option. This change allows other tasks to break in to perform DDL, utilities, or BIND actions, because it was difficult to remove pooled DBATs that might be holding table space intent and package locks. However, this change imposed the extra CPU for package allocation and deallocation on these pooled DBATs.

DB2 10 in CM allows DB2 DRDA DBAT threads to run accessing data under the RELEASE bind option of the package. Thus, if the package is bound with RELEASE(DEALLOCATE), the copy of the package is now allocated to the DBAT until the DBAT is terminated. For a discussion of high performance DBATs, see 9.4, “High performance DBAT” on page 324.

Unlike normal inactive thread processing, DDF does not pool the DBAT threads bound with RELEASE(DEALLOCATE) and disassociate them from their connection after the unit of work is ended. DDF causes the DBATs to remain active against the connection but also to end the enclave and cut an accounting record, much like normal inactive connection processing. After the DBAT is used to perform 200 units of work, it is terminated, and another DBAT is created or a already pooled DBAT allocated to process the next unit of work from a connection.

Normal idle thread timeout detection is applied to these DBATs. If the DBAT is in flight processing a unit of work and it has not received the next message from a client, DDF cancels the DBAT after the IDTHTOIN value has expired. However, if the DBAT is sitting idle having completed a unit of work for the connection and if it has not received a request from the client, then the DBAT is terminated (not cancelled) after POOLINAC time has expired.

If the CMTSTAT subsystem parameter is set to ACTIVE, the existing package processing is forced to be RELEASE(COMMIT), the same as in DB2 9. Only INACTIVE connection processing is enhanced. These inactive eligible DBATs, which are bound with RELEASE(DEALLOCATE), are called *high performance DBATs*.

DBATs now can hold package allocation locks and possibly table space level locks even while they are not being used for client unit-of-work processing, making it difficult to perform DDL, utilities, or BIND actions if required.

So, DB2 10 also provides the -MODIFY DDF command (listed in Figure 13-3) to allow you to dynamically change the behavior of the package release processing mode of DDF's inactive connection support. You can explicitly bind all distributed packages to use RELEASE(DEALLOCATE) to minimize the CPU overhead of package allocation and deallocation. Then you can change the behavior with the -MODIFY DDF PKGREL COMMIT command to RELEASE(COMMIT) when you need to perform activities such as BIND or DDL, which might be blocked by locks held by the distributed threads. When the DBA work is complete, you can issue the command again with the BINDOPT option to return the inactive thread behavior back to how the DBAT packages were originally bound.

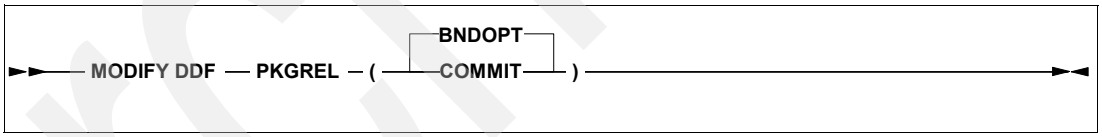


Figure 13-3 MODIFY DDF command

The MODIFY DDF command changes the behavior of any inactive threads (CMTSTAT is INACTIVE). If CMTSTAT is ACTIVE, then regardless of whether the command is issued, the packages use RELEASE(COMMIT).

The effect of switching from one value to another is not immediate. The change is essentially made when the threads begin to process new units of work. However, any DBAT that is left active against a connection due to its use of RELEASE(DEALLOCATE) packages and that is still waiting for a new unit of work request from its client might need to wait for a few minutes before being told to terminate by DDF.

DB2 10 maintains the system statistical values to help you to monitor the effectiveness of high performance DBATs. These values are externalized in the DDF section of OMEGAMON PE Statistics detail reports in the following counters:

QDSTNARD FIXED(32)	Number of currently active DBATs due to RELEASE(DEALLOCATE) packages
QDSTMARD FIXED(32)	Maximum number of active DBATs due to RELEASE(DEALLOCATE) packages (QDSTNARD high water mark)

The OMEGAMON PE statistics report of Example 13-1 shows these counters.

Example 13-1 OMEGAMON PE statistics report sample on DBATs

```
GLOBAL DDF ACTIVITY          QUANTITY
--------------------------  --------
DBAT/CONN QUEUED-MAX ACTIVE      0.00
CONN REJECTED-MAX CONNECTED      0.00
CONN CLOSED - MAX QUEUED         0.00

COLD START CONNECTIONS           0.00
WARM START CONNECTIONS           0.00
RESYNCHRONIZATION ATTEMPTED      0.00
RESYNCHRONIZATION SUCCEEDED      0.00

CUR TYPE 1 INACTIVE DBATS        0.00
HWM TYPE 1 INACTIVE DBATS        1.00
TYPE 1 CONNECTIONS TERMINAT      0.00

CUR INACTIVE CONNS (TYPE 2)      0.07
HWM INACTIVE CONNS (TYPE 2)     10.00
ACC QU INACT CONNS (TYPE 2)   3509.00
CUR QU INACT CONNS (TYPE 2)      0.00
HWM QU INACT CONNS (TYPE 2)      3.00

CUR ACTIVE AND DISCON DBATS     11.28
HWM ACTIVE AND DISCON DBATS     12.00
HWM TOTL REMOTE CONNECTIONS     10.00

CUR DISCON DBATS NOT IN USE      1.35
HWM DISCON DBATS NOT IN USE     10.00
DBATS CREATED                    1.00
DISCON (POOL) DBATS REUSED    3508.00

CUR ACTIVE DBATS-BND DEALLC      2.43
HWM ACTIVE DBATS-BND DEALLC     10.00
```

Preliminary measurements have shown up to an additional 10% CPU reduction from DB2 10 in NFM using RELEASE(DEALLOCATE) compared to RELEASE(COMMIT) bound distributed workloads. Bigger CPU reductions can be realized by short transactions.

The MODIFY command is available in CM and applies only to distributed threads. This command is useful because it causes the table space locks to be removed, allowing utilities and DDL to be executed.

Archived

13.4.2 Miscellaneous DDF performance improvements

This section summarizes the performance improvements that are available in CM and that do not require external changes to enable.

Fast implicit close for cursors declared FETCH FIRST FOR 1 ROW ONLY and WITH HOLD

In DB2 9, cursors with cursors declared FETCH FIRST FOR 1 ROW ONLY and WITH HOLD would not be eligible for implicit fast close because of the WITH HOLD clause. But, if the only reason that a FETCH FIRST FOR 1 ROW ONLY cursor cannot be fast closed is the WITH HOLD clause, then DB2 10 allows implicit fast close to occur. This implicit fast close improves overall DRDA application performance by avoiding a network flow from the application to the server to CLOSE the cursor.

For example, this feature can have benefits for some JDBC or CLI applications. When JDBC or CLI applications call the respective driver API to execute a query with the FETCH FIRST 1 ROW ONLY clause, the default driver settings declare the cursor as WITH HOLD, if a cursor declaration is not explicitly specified.

More efficient communication between DDF and DBM1

DB2 10 improves the performance of various interactions between the DDF address space and the DBM1 address space, improving the overall performance of distributed applications.

For example, in DB2 9, when DDF processes an OPEN request for a cursor from which data can be prefetched according to DRDA rules (this is known as DRDA block prefetch), DDF requires multiple interactions with the DBM1 address space to OPEN the cursor and then to perform DRDA block prefetch of the data.

In DB2 10, DDF, in many cases, might be able to reduce the number of interactions, so that both the OPEN and the DRDA block prefetch of the data can occur in the same interaction. If the prefetch of the data ultimately results in fetching the last row in the result table and if the cursor is also eligible to be fast closed, then the cursor will be fast closed implicitly in the same interaction.

Thus, in the case of a cursor where the full result table can be prefetched at OPEN time, just one network flow is required to send the OPEN request, drive a DRDA block prefetch of the data, and cause an implicit fast close of the cursor. In addition, with this feature, only one interaction between DDF and DBM1 is required to OPEN the cursor, to fetch the data, and to close the cursor. In particular, this is the behavior for eligible FETCH FIRST FOR 1 ROW ONLY cursors.

DB2 trace records show whether DDF was able to bundle the OPEN of the cursor and the DRDA block prefetch of data in the same interaction with DBM1. For example, Figure 13-4 shows a cursor for which DDF was not able to bundle the DRDA block prefetch with the OPEN.

Figure 13-4 shows the IFCID 58 record for the completion of the OPEN request at the server, along with a separate IFCID 59 record for the start of the DRDA block prefetch, followed by another IFCID 58 record for the completion of the prefetch. The SEQD portion of the second IFCID 58 record shows the number of rows fetched.

```
0000002075 16-003A-     IFCID  58-End SQL      1656     +0 DSNTEP3            DSNTEP3                                18.37.13.624920
J=VA1BDIST,ASCB=00FA7E80,ACE=156C0930,SQ=00000077,A=SYSADM  ,C=TEP3    ,CN=BATCH   ,P=DSNTEP3 ,EB=        ,            ,XU=
 SEC 2 R  1   +0000  E2E3D3C5 C3F1C240 40404040 40404040  C4E2D5E3 C5D7F340 40404040 40404040  *STLEC1B         DSNTEP3         *
              +0020  4040C4E2 D5E3C5D7 F3404040 40404040  40404040 18B5D296 1EE3F167 4040E2D8  *  DSNTEP3           ..Ko.T1.  SQ*
              +0040  D3C3C140 40400000 00880000 00000000  40404040 40404040 40404040 40404040  *LCA   ...h......                *
              +0060  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  *                                *
              +0080  40404040 40404040 40404040 40404040  40404040 4040C4E2 D5404040 40400000  *                     DSN     ..*
              +00A0  00000000 00000000 0000FFFF FFFF0000  00000000 000040D5 404040F1 40404040  *...................... N   1    *
              +00C0  40F0F0F0 F0F00000 00000678 00000000  00000000 0228FE6B 66D80000 00000000  * 00000.................,.Q......*
              +00E0  0000                                                                     *..                              *

0000002095 16-003B-     IFCID  59-FETCH   01   1656 C_NORMAL                                                                  18.37.13.626362
J=VA1BDIST,ASCB=00FA7E80,ACE=156C0930,SQ=0000006E,A=SYSADM  ,C=TEP3    ,CN=BATCH   ,P=DSNTEP3 ,EB=        ,            ,XU=
 SEC 2 R  1   +0000  E2E3D3C5 C3F1C240 40404040 40404040  C4E2D5E3 C5D7F340 40404040 40404040  *STLEC1B         DSNTEP3         *
              +0020  4040C4E2 D5E3C5D7 F3404040 40404040  40404040 18B5D296 1EE3F167 40400100  *  DSNTEP3           ..Ko.T1.  ..*
              +0040  12C36DD5 D6D9D4C1 D3404040 40404040  40404040 00000678 40D55900 00000000  *.C_NORMAL          .... N......*
              +0060  00000000 00000228 FE6B66D8 00000000  00000000                           *.........,..Q........           *
0000002098 16-003A-     IFCID  58-End SQL      1656     +0 DSNTEP3            DSNTEP3                                18.37.13.627825
J=VA1BDIST,ASCB=00FA7E80,ACE=156C0930,SQ=00000078,A=SYSADM  ,C=TEP3    ,CN=BATCH   ,P=DSNTEP3 ,EB=        ,            ,XU=
 SEC 2 R  1   +0000  E2E3D3C5 C3F1C240 40404040 40404040  C4E2D5E3 C5D7F340 40404040 40404040  *STLEC1B         DSNTEP3         *
              +0020  4040C4E2 D5E3C5D7 F3404040 40404040  40404040 18B5D296 1EE3F167 4040E2D8  *  DSNTEP3           ..Ko.T1.  SQ*
              +0040  D3C3C140 40400000 00880000 00000000  40404040 40404040 40404040 40404040  *LCA   ...h......                *
              +0060  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  *                                *
              +0080  40404040 40404040 40404040 40404040  40404040 4040C4E2 D5404040 40400000  *                     DSN     ..*
              +00A0  00000000 00000000 0000FFFF FFFF0000  00000000 00004040 40404040 40404040  *......................          *
              +00C0  40F0F0F0 F0F00000 00000678 00000000  00000000 0228FE6B 66D80000 00000000  * 00000.................,.Q......*
              +00E0  0000                                                                     *..                              *
 SEC 3 R  1   +0000  E2C5D8C4 00000000 00000000 00000074  00000000 00000074 00000000 00000074  *SEQD............................*
              +0020  00000000 00000074 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
              +0040  00000009 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
```

Figure 13-4 DB2 trace records IFCID 58 and 59 for OPEN and unbundled DRDA block prefetch

Alternatively, Figure 13-5 shows a case where DDF bundled the OPEN and the DRDA block prefetch into a single interaction with DBM1. There is a single IFCID 58 record for the completion of the OPEN, along with a SEQD record showing the number of rows prefetched with the OPEN in the same interaction.

```
0000000351 16-003A-     IFCID  58-End SQL      1656     +0 DSNTEP3            DSNTEP3                                18.36.22.580294
J=VA1BDIST,ASCB=00FA7E80,ACE=156C0930,SQ=00000072,A=SYSADM  ,C=TEP3    ,CN=BATCH   ,P=DSNTEP3 ,EB=        ,            ,XU=
 SEC 2 R  1 +0000  E2E3D3C5 C3F1C240 40404040 40404040  C4E2D5E3 C5D7F340 40404040 40404040  *STLEC1B         DSNTEP3         *
            +0020  4040C4E2 D5E3C5D7 F3404040 40404040  40404040 18B5D296 1EE3F167 4040E2D8  *  DSNTEP3           ..Ko.T1.  SQ*
            +0040  D3C3C140 40400000 00880000 00000000  40404040 40404040 40404040 40404040  *LCA   ...h......                *
            +0060  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  *                                *
            +0080  40404040 40404040 40404040 40404040  40404040 4040C4E2 D5404040 40400000  *                     DSN     ..*
            +00A0  00000000 00000000 00000000 00000000  00000000 00004040 40404040 40404040  *......................          *
            +00C0  40F0F0F0 F0F00000 00000678 00000000  00000000 0228FE6B 66D80000 00000000  * 00000.................,.Q......*
            +00E0  0000                                                                     *..                              *
 SEC 3 R  1 +0000  E2C5D8C4 00000000 00000000 00000074  00000000 00000074 00000000 00000074  *SEQD............................*
            +0020  00000000 00000074 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
            +0040  00000009 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
            +0060  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
```

Figure 13-5 DB2 trace records IFCID 58 and 59 for OPEN and bundled DRDA block prefetch

In addition, the effect of bundled interactions can be seen in DB2 statistics reports, such as those produced by OMEGAMON PE. In Figure 13-6, the extract from an OMEGAMON PE report shows the statistics of an extreme example of at a server where 497,232 OPEN cursor requests were processed and where every cursor was eligible for both bundled DRDA block prefetch and for implicit fast close.

```
DESCRIBE      0.00         0
DESC.TBL      0.00         0
PREPARE       0.00         0
OPEN          2.60    497232
FETCH         0.00         0
 ROWS        14.44   2762863
CLOSE         0.00         0

DML-ALL       4.20    803164
```

Figure 13-6 Statistics report

The FETCH count is zero, indicating that DDF did not process any remote FETCH requests from the application and did not perform any unbundled DRDA block prefetch. The CLOSE count is also zero, indicating that DDF did not process any remote CLOSE requests from the application. This extract shows all 2,762,863 rows were DRDA block prefetched in the same interactions with their respective OPEN requests.

13.5 Dynamic statement cache enhancements

Prior to DB2 V5, many application developers did not like to use dynamic SQL because every dynamic SQL had to undergo the PREPARE processing, a process similar to a BIND, for DB2 to assess the access path. The introduction of the dynamic statement cache meant that the access path was calculated the first time the SQL was executed, but every later time, the same SQL was executed by the same user a PREPARE could be avoided as the access path can be reused. Thus, using dynamic SQL can be almost as efficient as static SQL, which is good news for companies running Customer Relationship Management type of applications.

To take advantage of reuse in the cache, the SQL string had to be identical and the SQL had to be executed by the same user. This requirement effectively meant programs had to be coded with parameter markers (“?”). These values are similar to using host variables in static SQL. SQL then would always be identical, even if the values in the parameter markers changed.

However, some applications, such as Ruby on Rails, generates dynamic SQL with literals rather than using parameter markers. Because the literals are likely to be different with every execution of the SQL, little reuse of the SQL in the cache can take place. A prepare must take place for each unique piece of SQL so that the whole application can run slower than a similar application using parameter markers.

In DB2 10, more SQL can be reused in the cache across users. Dynamic SQL statements can now be shared with a already cached dynamic SQL statements, if the only difference between the two statements is literal values.

In the dynamic statement cache, literals are replaces with an ampersand (&) that behaves similar to parameter markers. For example:

```
SELECT BALANCE FROM ACCOUNT WHERE ACCOUNT_NUMBER = 123456
```

Is replaced in the SQL cache by:

```
SELECT BALANCE FROM ACCOUNT WHERE ACCOUNT_NUMBER = &
```

If a mixture of literal constants and parameter markers are used in the SQL statement, DB2 does not perform literal translation.

To enable this function, use one of the following methods:

- On the client, change the PREPARE statement to include the new ATTRIBUTES clause, specifying CONCENTRATE STATEMENTS WITH LITERALS.
- On the client side, change the JCC Driver to include the ("statementConcentrator", "2") keyword, which is specified in the data source or connection property.
- Set LITERALREPLACEMENT in the ODBC initialization file in z/OS, which enables all SQL coming into DB2 through ODBC to have literal replacement enabled.

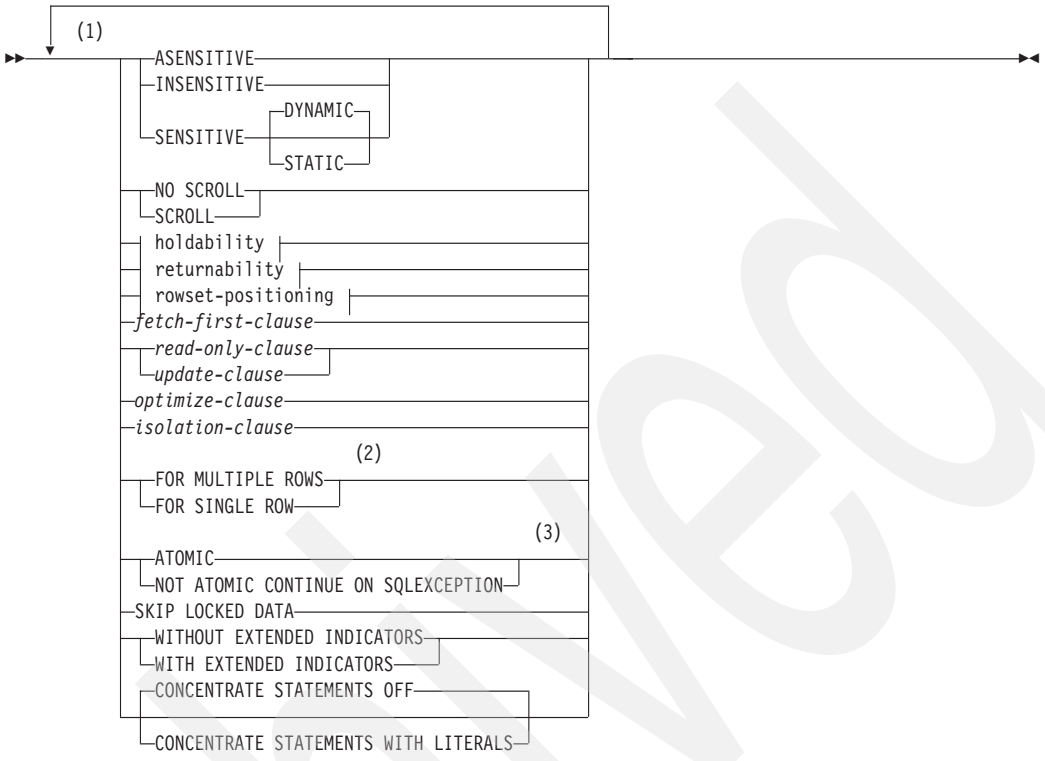
Thus, the lookup sequence in the dynamic statement cache is now as follows:

- Original SQL with literals is looked up in the cache, which is pre-DB2 10 behavior.
- If not found, literals are replaced and the new SQL is looked up in the cache. DB2 can match only with SQL that is stored with same attribute. So, SQL in the cache with parameter markers is not matched.
- If not found, new SQL is prepared and stored in the cache.

Together with the actual SQL statement, DB2 stores in the dynamic statement cache information about the actual literals, including the literal values and the context in which the literals are used. For literal reusability, the reusability criteria includes but is not limited to the immediate usage context, literal data type, and data type size of both the new literal instance and the cached literal instance. This literal reusability data must also be compatible with the actual SQL statement, if the cached SQL is to be used. If the literal reusability does not match then the new SQL must undergo full prepare and a new version of the statement is also stored in the dynamic statement cache.

Archived

Figure 13-7 shows the attributes of the PREPARE STATEMENT with the CONCENTRATE STATEMENTS WITH LITERALS clause.



Notes:

1 The same clause must not be specified more than one time. If the options are not specified, their defaults are whatever was specified for the corresponding option in an associated statement.

2 The FOR SINGLE ROW or FOR MULTIPLE ROWS clause must only be specified for an INSERT or a MERGE statement.

3 The ATOMIC or NOT ATOMIC CONTINUE ON SQLEXCEPTION clause must only be specified for an INSERT statement.

Figure 13-7 Attributes clause of the PREPARE statement

Here is an example on how to use the new ATTRIBUTES clause for SELECT:

```
EXEC SQL DECLARE C1 CURSOR FOR DYNSQL_WITH_LITERAL;
DYNSQL_SELECT = 'SELECT X, Y, Z FROM TABLE1 WHERE X < 9';
attrstring = 'CONCENTRATE STATEMENTS WITH LITERALS';
EXEC SQL PREPARE DYNSQL_WITH_LITERAL ATTRIBUTES :attrstring FROM
:DYNSQL_SELECT;
EXEC SQL OPEN C1;
```

Here is an example on how to use the new ATTRIBUTES clause for INSERT:

```
DYNSQL_INSERT = 'INSERT INTO TABLE1 (X, Y, Z) VALUES (1, 2, 3)';
attrstring = 'CONCENTRATE STATEME
NTS WITH LITERALS';
EXEC SQL PREPARE DYNSQL_WITH_LITERAL ATTRIBUTES :attrstring FROM
:DYNSQL_INSERT;
EXEC SQL EXECUTE SYNSQL_WITH_LITERAL;
```

Archived

Using parameter markers can provide higher dynamic statement cache hit ratio. However, using literals can provide the better access path. Using this method is a trade-off, depending on the performance characteristics of a dynamic SQL call. Using CONCENTRATE STATEMENTS WITH LITERALS can at times result in a degraded execution-only performance. However, the biggest performance gain is for small SQL statements with literals that have a cache hit now, but did not before.

The REOPT bind option is enhanced to address this possibility in degraded performance. Normally, REOPT(AUTO) is applicable only to dynamic SQL statements that reference parameter markers (“?”). Now, if the REOPT(AUTO) bind option is specified and the PREPARE for a dynamic SQL statement matches SQL in the dynamic statement cache with the literal replacement character (“&”), then for each OPEN or EXECUTE of that dynamic statement DB2 reevaluates that access path using the current instance of literal values. This behavior is similar to the current REOPT(AUTO) behavior and host variable usage.

The LITERAL_REPL column is added to the DSN_STATEMENT_CACHE_TABLE to identify those cached statements which have their literals replaced with ampersands (&). The ampersands are also visible in the actual SQL statement text, which is externalized to the STMT_TEXT column. Column LITERAL_REPL is defined as CHAR(1). Table 13-3 shows its possible values.

Table 13-3 Column LITERAL_REPL values

Replacement character	Meaning
“R”	Literals are replaced with an ampersand (&)
“D”	Literals are replaced with an ampersand (&) and can have an identical value for STMT_TEXT to another cached statement or statements, but literal reusability criteria is different
“ “	(Default) The statement was not prepared with CONCENTRATE STATEMENTS WITH LITERALS and no literal constants are replaced

Example A-11 on page 623 shows the changes to the Dynamic Statement Cache Statement Statistics trace record, IFCID 316. The QW0316LR field is included to indicate that the statement used literal replacement.

Example A-1 on page 616 shows the changes to the Database Services Statistics trace record, IFCID 002. In addition to using the existing fields QXSTFND (short prepare or cache match found) and QXSTNFND (cache full prepare or cache match not found) for capturing caching statistics, the new fields are also captured to track the increased benefit of using CONCENTRATE STATEMENTS WITH LITERALS.

Finally, other trace records such as IFCID 063, IFCID 317, and IFCID 350 which externalize the SQL statement text, show the statement text with the statement literals replaced with ampersands (&).

13.6 INSERT performance improvement

Heavy insert applications can hit a series of issues that hinder performance, scalability, and availability. DB2 must remove these issues to take advantage of faster host and storage servers.

Figure 13-8 summarizes the main enhancements on insert processing with DB2 9 and DB2 10.

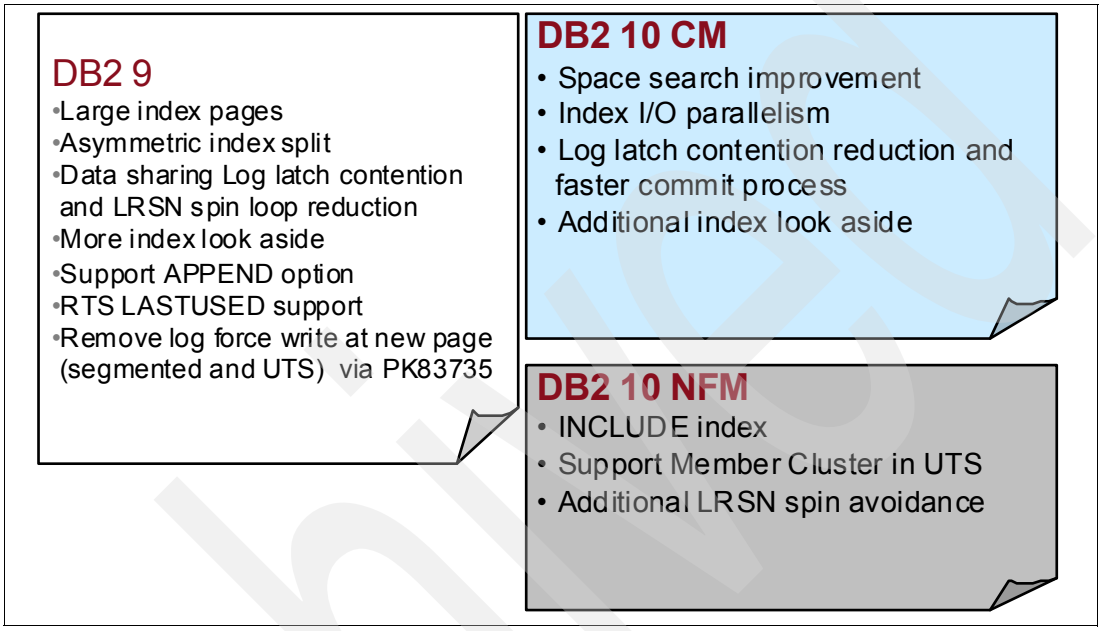


Figure 13-8 Summary of main insert performance improvements

DB2 9 introduced the following improvements for heavy INSERT processing applications:

- Large index page support

  Index pages larger than 4 KB (enable index compression and) reduce the index splits proportionally to the increase in size of the page.

- Asymmetric index split

  Based on the insert pattern, DB2 splits the index page by choosing from several algorithms. If an ever-increasing sequential insert pattern is detected for an index, DB2 splits index pages asymmetrically using approximately a 90/10 split.If an ever-decreasing sequential insert pattern is detected in an index, DB2 splits index pages asymmetrically using approximately 10/90 split. If a random insert pattern is detected in an index, DB2 splits index pages with a 50/50 ratio.

- Data sharing log latch contention and LRSN spin loop reduction

  Allows for duplicate LRSN values for consecutive log records for different pages on a given member.

- More index look aside

  DB2 keeps track of the index value ranges and checks whether the required entry is in the leaf page accessed by the previous call. It also checks against the lowest and highest key of the leaf page. If the entry is found, DB2 can avoid the getpage and traversal of the index tree.

- APPEND=YES

  Requests data rows to be placed into the table by disregarding the clustering during SQL INSERT and online LOAD operations. Rows are appended at the end of the table or partition.

- RTS LASTUSED column

  RTS records the day the index was last used to process an SQL statement. Not used indexes can be identified and DROPped.

- Remove log force write at new page through PK83735

  Forced log writes are no longer done when inserting into a newly formatted or allocated page for a GBP dependent segmented or universal table space.

For details, see *DB2 9 for z/OS Performance Topics*, SG24-7473.

DB2 10 further improves the performance for heavy INSERT applications, through:

- I/O parallelism for index updates
- Sequential inserts into the middle of a clustering index

13.6.1 I/O parallelism for index updates

When DB2 9 inserts a row into a table, it must perform a corresponding insert into all the indexes that are defined on that table. All of these inserts into the indexes are done sequentially. Each insert into an index must be completed before the insert into the next index can start. If there are many indexes defined on the table and if the index pages that are needed for the insert are not always in the buffer pool, the inserting transactions can suffer from high response times due to index I/O wait times.

DB2 10 provides the ability to insert into multiple indexes that are defined on the same table in parallel. Index insert I/O parallelism manages concurrent I/O requests on different indexes into the buffer pool in parallel, with the idea being to overlap the synchronous I/O wait time for different indexes on the same table. This processing can significantly improve the performance of I/O bound insert workloads. It can also reduce the elapsed times of LOAD RESUME YES SHRLEVEL CHANGE utility executions, because the utility functions similar to a MASS INSERT when inserting to indexes.

DB2 performs the insert on the first index. A conditional getpage returns the page if it is already in the buffer pool, but it never initiates an I/O. Thus, if leaf pages are buffer hits, DB2 does not schedule the prefetch engine. When DB2 gets a buffer miss, it schedules a prefetch engine and continues with the next index. Then, later it must do another getpage after the prefetch completes.

Because DB2 cannot avoid waiting for I/O when reading the clustering index to find the candidate data page, I/O parallelism cannot be performed against the clustering index. In general, there is also no benefit to not waiting for the I/O to complete against the last index, because DB2 will probably have to wait for the I/O to complete for the first index revisited anyway, (An I/O is orders of magnitude slower than CPU processing time.) So, asynchronous I/O is not scheduled for the last index and the clustering index.

In general, this enhancement benefits tables with three or more indexes defined. The exceptions are tables defined as MEMBER CLUSTER, tables created with APPEND YES option, or tables created with the ORGANIZE BY HASH clause. In these cases, indexes are not really used to position the rows in the table. So tables with two or more indexes benefit from this enhancement, rather than three or more.

All I/Os for non-leaf pages are still done synchronously. Only the leaf page I/Os can be scheduled asynchronously.

I/O parallelism for index updates also does not apply for any indexes that are defined on the DB2 catalog and directory objects, either DB2 created or user created. There is a small CPU overhead in scheduling the prefetch. This enhancement benefits tables with three indexes or more, especially in case of poor disk performance. Tables with many large indexes that are also not already in the buffer pool will see the greater performance improvement.

I/O parallelism for index update is active in CM and a rebind or bind is not required. However, it is only available for classic partitioned table spaces and universal table spaces (partition-by-growth and range-partitioned). Segmented table spaces are not supported.

Index I/O parallelism likely reduces insert elapsed time and class 2 CPU time. Elapsed time savings are greatest when I/O response times are high. Due to the extra overhead of a sequential prefetch, DBM1 service request block (SRB) time increases, and the total CPU time increases. However, in DB2 10, because the prefetch SRB time is zIIP eligible, the total cost of the CPU time can be reduced.

I/O parallelism for index updates can also be disabled by setting the new online changeable DSNZPARM parameter INDEX_IO_PARALLELISM to NO. The default is YES. You want to disable this function if the system has insufficient zIIP capacity to redirect the prefetch engine.

The new IFCID 357 and IFCID 358 are available in DB2 10 to trace the start and end of index I/O parallel insert processing. You can use these IFCIDs to monitor for each table insert operation the degree of I/O parallelism, which is the number of synchronous I/O waits DB2 has avoided during the insertions into the indexes for a given table row insert.

Example A-8 on page 622 shows the contents of IFCID 357, and Example A-9 on page 622 shows the contents of IFCID 358.

13.6.2 Sequential inserts into the middle of a clustering index

When inserting rows into a table, a set of space management algorithms are in place to find the candidate page where the row is to be inserted according to the clustering index. DB2 10 enhances the way the first candidate page is selected.

To select the initial candidate page, DB2 selects the data page where the row that contains the next highest key to the row being inserted resides. If there is available space, DB2 inserts the row into that page. However, if there is not enough space, DB2 searches for another candidate page and eventually finds space to insert the row.

Now, consider the case where a second row is inserted that has a key higher than the row just inserted but lower than the next highest existing row. DB2 selects the same initial candidate page again, only to find it is still full. So, DB2 must repeat the process to find another candidate page.

DB2 10 changes this behavior. Rather than choosing the page pointed to by the next highest key as the initial candidate page, DB2 chooses the first candidate page based on the next *lower* key. In our example that uses sequential inserts, DB2 chooses the page where the previous row was inserted as the first candidate page to check. Because a row was just inserted, the chances are reasonable that the page still contains enough space for this new row. So, DB2 does not have to search for another candidate page.

This behavior helps sequential inserts into the middle of the table based on the clustering index. On the second and subsequent sequential insert, DB2 does not have to repeatedly find

the first candidate page as full, which translates directly into CPU and getpage savings because fewer candidate pages need to be searched for sequential insert workloads. This performance improvement is available in CM with no rebind or bind required.

13.7 Referential integrity checking improvement

When inserting into a dependent table, DB2 must access the parent key for referential constraint checking. DB2 10 changes help to reduce the CPU overhead of referential integrity checking by minimizing index probes for parent keys:

- DB2 10 allows sequential detection to trigger dynamic prefetch for parent key referential integrity checking.
- DB2 10 also enables index look-aside for parent key referential integrity checking. Index look-aside is when DB2 caches key range values. DB2 keeps track of the index value ranges and checks whether the required entry is in the leaf page accessed by the previous call. If the entry is found, DB2 can avoid the getpage and traversal of the index tree. If the entry is not found, DB2 checks the parent non-leaf page's lowest and highest key. If the entry is found in the parent non-leaf range, DB2 must perform a getpage but can avoid a full traversal of the index tree.
- DB2 can also avoid index lookup for referential integrity checking, if the non-unique key to be checked has been checked before.
  - INSERT KEY A, INSERT KEY A, ... INSERT KEY A, COMMIT;

    For the 1st INSERT KEY A, DB2 checks the parent table index for RI. no RI checking takes place for all subsequent inserts.
  - INSERT KEY A, COMMIT; INSERT KEY A, COMMIT;

    Only the 1st INSERT checks the parent index, all subsequent INSERT will not check.

  So for INSERT within or without the same commit scope, if the key is already in the child table, DB2 does not check the parent key value again. If key A is already in the child table (already committed), when you insert another key A (assuming non-unique index on child), DB2 detects that key A is already there, so there is no need to check the parent again because key A already matches the RI rule otherwise it cannot be in the child table.

  However there must be an index on the child table, with the relationship primary key column(s) defined as leading columns in the index. Otherwise, you would just benefit of index look-aside on parent table, but not due to key already exist.

Referential integrity checking can take advantage of index enhancements and hash access can be used for parent key checking, however referential integrity checking is not externalized in the Explain tables.

13.8 Buffer pool enhancements

The buffer pool enhancements in DB2 10, which are all available in CM, allow you to increase transaction throughput and take advantage of larger buffer pools by reducing latch class 14 and 24 contention, reducing buffer pool CPU overhead, and avoiding transaction I/O delays by preloading objects into the buffer pool.

We describe these enhancements in the following sections:

- Buffer storage allocation
- In-memory table spaces and indexes

- Reduce latch contention

13.8.1 Buffer storage allocation

In previous versions of DB2, storage is allocated for the entire size of the buffer pool (VPSIZE) when the buffer pool is first allocated, (the first logical open of a page set), even if no data was accessed in any table space or index using that buffer pool. Now, buffer pool storage is allocated on-demand as data is brought in. If a query touches only a few data pages, only a small amount of buffer pool storage is allocated.

Here, *logical open* is when the page set is either physically opened (the first SELECT) or pseudo opened (the first UPDATE after being physically opened for read). There is no buffer pool allocation as it is already allocated at read time.

In addition, for a query that performs index-only access, the buffer pool for the table space does not need to have any buffer pool storage allocated. DB2 10 no longer performs logical open of the table space page set for index-only access.

For buffer pools defined with PGFIX=YES, DB2 requests buffer pools to be allocated using 1 MB page frames if they are available, rather than 4 KB pages frames. 1 MB page frames are available in z10 and later. You define the number of 1 MB page frames that are available to z/OS in the LFAREA parameter of SYS1.PARMLIB(IEASYSxx). Manipulating storage in 1 MB chunks rather than 4 KB chunks can significantly reduce CPU overhead in memory management, by increasing the hit ratios of the hardware translation lookaside buffer.

Note that although DB2 can request 1 MB page frames from the operating system, DB2 itself still manages the buffer pools as 4 KB, 8 KB, 16 KB, and 32 KB pages. Nothing changes inside DB2.

DB2 requests 1 MB page frames only for PGFIX=YES buffer pools because individual 4 KB buffer pool pages are already page fixed in memory for read/write operations in 1 MB chunks. If there are no more 1 MB page frames available, then DB2 requests 4 KB page frames. See 2.2.3, “Increase of 64-bit memory efficiency” on page 5 for details.

Buffer pools do not automatically shrink when page sets are logically closed. When allocated, the buffer pools remain until either all the page sets are physically closed or DB2 is stopped. Workload managed buffer pool support introduced in DB2 9 can dynamically reduce the size of buffer pools when there is less demand. See *DB2 9 for z/OS Performance Topics*, SG24-7473 for details about WLM buffer pool support.

There can be a temptation either to make buffer pools bigger than they normally would be or to define more buffer pools with PAGEFIX=YES, thinking that the extra buffer pool space might not be used because it is allocated only when it is needed. We advise that you resist this temptation. You still need enough real storage to back the buffer pools to keep real paging at an acceptable level and your amount of real storage has not changed.

For BP0, BP8K0, BP16K0, and BP32K, the minimum size set by DB2 is 8 MB.

13.8.2 In-memory table spaces and indexes

In the past, the cost of physically opening a page set was worn by the first SQL to access that data set. This cost adversely impacted application performance, typically after DB2 restart. In addition, some tables might be critical for application performance, so they need to be always resident in the buffer pools.

DB2 10 provides a buffer pool attribute that you can use to specify that all objects using that buffer pool are in-memory objects. The data for in-memory objects is preloaded into the buffer pool at the time the object is physically opened, unless the object has been opened for utility access. The pages remain resident as long as the object remains open.

When the page set is first accessed (the first getpage request initiated by SQL) an asynchronous task is scheduled under the DBM1 address space to prefetch the entire page set into the buffer pool. The CPU for loading the page set into the buffer pool is therefore charged to DB2. If this first getpage happens to be for a page that is being asynchronously read at the time, then it waits. Otherwise, the requested page is read synchronously.

If a page set is opened as a part of DB2 restart processing, the entire index or table space is not prefetched into the buffer pool.

Page sets can still be physically opened before the first SQL access by using the -ACCESS DATABASE command introduced in DB2 9. Now, you can also preload all of the data into the buffer pool or pools before the first SQL access.

To realize the benefit of in-memory page sets, you still need to make sure that the buffer pools are large enough to fit all the pages of all the open page sets. Otherwise, I/O delays can occur as the buffer pool fills up and DB2 must steal buffers, (on a FIFO in this case). This behavior is the same as with previous versions of DB2.

In-memory page sets help DB2 to reduce overall least recently used (LRU) chain maintenance and latch class 14 contention, because there is much less buffer pool activity and buffer pools with in-memory page sets are managed with FIFO. They also avoid unnecessary prefetch and latch class 24 contention, because the data is already in the buffer pools and because prefetch is disabled for in-memory page sets.

Important: It is more important in DB2 10 to make sure that you have large enough buffer pools to store all the in-memory page sets. If the buffer pools are too small to store all of the data, then performance can be impacted, because DB2 might be using a nonoptimal access plan that did not allow for the extra I/O. In addition, if DB2 needs to perform I/O to bring a page into the buffer pool for processing, this I/O is synchronous because prefetch is disabled.

Archived

A new option is available for the PGSTEAL parameter of the -ALTER BUFFERPOOL command. PGSTEAL(NONE) indicates that no page stealing can occur. All the data that is brought into the buffer pool remains resident. Figure 13-9 shows the new syntax.

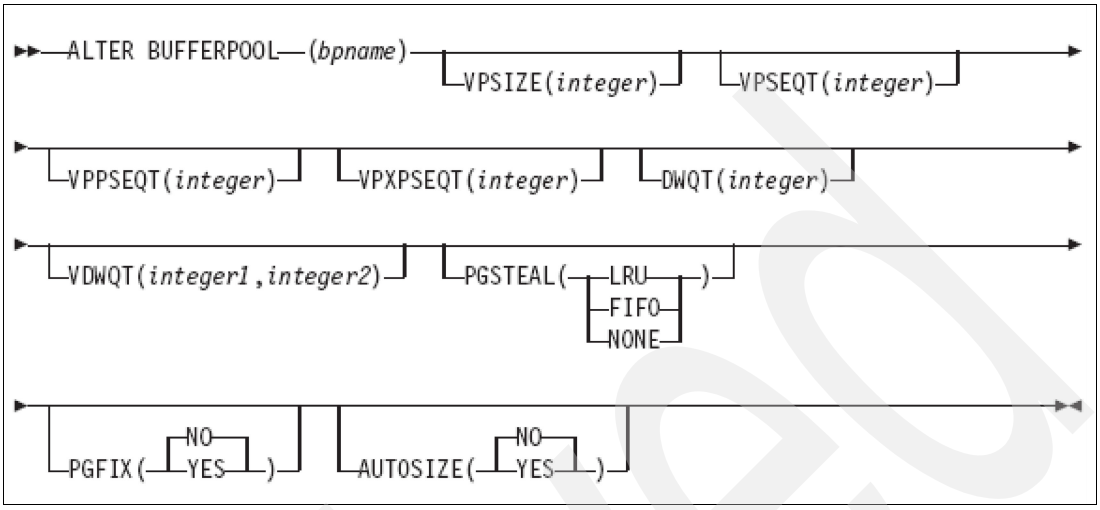


Figure 13-9 ALTER BUFFERPOOL command

Altering the PGSTEAL value takes effect immediately. For PGSTEAL LRU or FIFO, new pages added to the LRU chain take the new behavior immediately, but the ALTER does not affect the pages already on the chain. Altering the buffer pool to PGSTEAL(NONE) also has an immediate effect. The ALTER schedules prefetches for all of the page sets in the buffer pool.

You can define in-memory table spaces and indexes in DB2 10 CM. On fallback to DB2 9, PGSTEAL=NONE reverts to its previous value, which is LRU if the parameter was never changed. On remigration, PGSTEAL returns to NONE if it was set prior to fallback.

The following buffer manager display messages are modified to accommodate PGSTEAL=NONE:

```
DSNB402I
```

The BUFFERS ACTIVE count is removed, because it reflects the number of buffers that have ever been accessed, which is essentially the same behavior as BUFFERS ALLOCATED:

```
DSNB406I
DSNB519I
```

IFCID 201 records the buffer pool attributes changed by the -ALTER BUFFERPOOL command. A new value of *N* indicates that PGSTEAL(NONE) is defined for the QW0201OK and QW0201NK trace fields, which records the old and new values of PGSTEAL respectively. Similarly, IFCID 202, which records the current attributes of a buffer pool, also uses N to indicate PGSTEAL(NONE).

13.8.3 Reduce latch contention

DB2 10 introduces improvements to buffer pool management to reduce latch contention on latch class 14 (buffer pool manager exclusive latch) and latch class 24 (buffer pool manager page latch), particularly for large buffer pools, which is achieved through faster suspend and resume processing.

Latch class 14 is reduced during commit on update transactions. During update commit, DB2 must take an exclusive latch. DB2 10 takes the latch at the partition level rather than table space level. Latch class 24 is reduced as DB2 10 reduced the serialization in concurrent read threads such as CPU parallelism. DB2 must serialize when reading the same page at the same time from multiple threads, typically using CPU parallelism.

Class 14 hash latch contention reduction takes place also by having 10 times more latches for a given buffer pool size.

In-memory table spaces also significantly reduce buffer pool latch contention.

13.9 Work file enhancements

DB2 10 in NFM supports partition-by-growth table spaces in the work file database.

Declared global temporary tables (DGTTs) compete for space with other activities in the work file database, but they cannot span work file table spaces. Partition-by-growth work file table spaces help these applications to reduce SQLCODE -904 (unavailable resource) for lack of space in the work file database by allowing you to control the maximum size of work file table spaces using DSSIZE and MAXPARTITIONS.

For example, you can limit a partitioned by growth table space's use at 3 GB with the following setting:

```
MAXPARTITIONS 3 DSSIZE 1G
```

With DB2-managed segmented table spaces, this function was not possible. You can limit the growth at only 2 GB or less (using PRIQTY nK SECQTY 0).

If the DSNZPARM WFDBSEP is NO (the default), DB2 tries to use only work file partitioned-by-growth table spaces for DGTT; however, if there is no other table space available, DB2 also uses it for work files (for example, sorts and CGTTs).

With WFDBSEP YES, DB2 uses only work file partitioned by growth table spaces for DGTT, and if there is no other table space available, a workfile application receives a resource unavailable message.

For DGTTs using large DSSIZE, you can have larger work file table space (that is, greater than 64 GB).

You can have a mixture of table space types (some segmented and some universal table spaces partition-by-growth) in the work file database. A partitioned by growth table space in work file database must be produced using the CREATE TABLESPACE statement when in DB2 10 NFM. It cannot be altered to from other table spaces.

The records of work files that are created for joins and large sorts can span multiple pages to accommodate larger record lengths and larger sort key lengths for sort records. The maximum length of a work file record is 65529 bytes. This enhancement reduces the instances of applications receiving SQLCODE -670 when the row length of a large sort record or as a result of a join exceeds approximately the maximum page size for a work file table space.

Support for spanned work file records is available only in NFM; however, a rebind or bind is not required.

The maximum limit of sort key length is also increased from 16000 to 32707 bytes, which reduces instances of applications receiving SQLCODE -136.

In DB2 9, the use of in-memory work files is restricted to small work files that do not require any predicate evaluation. DB2 10 extends the use of in-memory work files by allowing simple predicate evaluation for work files. This enhancement helps to reduce the CPU time for workloads that execute queries that require the use of small work files. The in-memory work file enhancement is available in CM, and a rebind or bind is not required.

Finally, the maximum size of all 4 KB WORKFILE table spaces can now be up to 8,388,608,000 MB and the maximum size of all 32 KB WORKFILE table spaces can now be up to 67,108,864,000 MB. However, when migrating to DB2 10, the limits both remain 32,768,000 MB, the same as DB2 9, because WORKFILE table spaces cannot be created as partitioned-by-growth in DB2 10 CM. See Chapter 12, “Installation and migration” on page 471 for details about changes to the installation process.

13.10 Support for z/OS enqueue management

DB2 V8 exploits Workload Manager for z/OS (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

DB2 10 uses IBM Workload Manager for z/OS enqueue management to more effectively manage lock holders and waiters. DB2 also notifies WLM about threads that are being delayed while holding some key resources such as enqueues and critical latches.

13.11 LOB enhancements

In this section, we discuss the following enhancements, which are mostly related to LOBs:

- LOAD and UNLOAD with spanned records
- File reference variable enhancement for 0 length LOBs
- Streaming LOBs and XML between DDF and DBM1
- Inline LOBs

APAR PM24721 provides BIND performance improvement on LOB table spaces.

13.11.1 LOAD and UNLOAD with spanned records

Prior to DB2 9, DB2 sometimes could not LOAD or UNLOAD large LOB or XML columns with other non-LOB or XML columns into the same data set because the I/O record size was limited to 32 KB with VB type data sets.

In DB2 9, DB2 allows loading or unloading of LOB or XML data from or into separate data sets using file reference variables. However, the UNLOAD utility’s support of file reference variables is restricted to partitioned sets and UNIX file systems. File references variables cannot be used to unload all of the LOB or XML columns for an individual table space or partition to a single sequential file, and cannot unload LOB or XML data to tape (because all tape data sets are sequential). Writing all of the LOB or XML data to a partitioned data set or

UNIX file system is slow. Furthermore, the only way to unload LOB or XML data that is larger than 32 KB is by using file reference variables.

DB2 10 introduces support for spanned records (RECFM = VS or VBS) in LOAD or UNLOAD to allow LOB columns and XML columns of any size to be loaded or unloaded from or to the same data set with other non-LOB columns. Spanned records overcome the limitations of File Reference Variables, because all of the LOB or XML data of a given table space or partition can be written to a single sequential file, which can reside on tape or disk and can span multiple volumes.

You ask the LOAD or UNLOAD utilities to use spanned records by specifying the new SPANNED keyword, as shown in Example 13-2.

Example 13-2 Unloading in spanned format

```
UNLOAD TABLESPACE TESTDB1.CLOBBASE SPANNED YES
      FROM TABLE TB1
        (ID
        ,C1 INTEGER
        ,C2 INTEGER
        ,C3 CHAR(100)
        ,C4 CHAR(100)
        ,C5 INTEGER
        ,C6 CHAR(100)
        ,C7 CHAR(100)
        ,C8 CHAR(100)
        ,CLOB1 CLOB
        ,CLOB2 CLOB
        ,CLOB3 CLOB)
```

DB2 ignores the RECFM when SPANNED YES is specified. All LOB and XML columns must be ordered at the end of the record as specified in a field specification list. A field specification list is required, and length and POSITION must not be specified on the LOB or XML field specifications. TRUNCATE has no meaning. If the data is not ordered as DB2 UNLOAD expects, spanned records are not used and DSNU1258I will be issued. If DELIMITED is specified, the data will not be unloaded using spanned records. If SPANNED YES is specified NOPAD is the default. You can only use spanned records in NFM.

When unloading LOB or XML data to a spanned record data set, all non-LOB and non-XML data (including file reference variables) are written in the record first and then LOBs and XML documents are written, spanning to subsequent records if required. This is also the order and format the LOAD utility needs to read when using a spanned record processing of LOB or XML data. The SYSPUNCH generated by UNLOAD lists the LOB or XML data in a field specification list in the corresponding order.

The LOAD utility uses spanned record functionality if FORMAT SPANNED YES is specified, SYSREC has spanned record format, and a field specification list is provided with all LOB and XML fields at the end of the record. A field specification list is required, and position must be omitted or must use an asterisk (*)” for the LOB and XML columns. If the fields are not ordered as DB2 LOAD expects, then message DSNU1258I is issued and the utility terminates with RC8.

Performance of reading or writing from or to a single sequential file is much faster than reading or writing from or to separate files or partition data set members, because the utilities do not have as much open and close work to do.

In addition, LOAD REPLACE into a LOB table space in DB2 10 uses *format writes* in the same way that LOAD has always done for non-LOB table space. This method is faster than preformatting the table space as used by CREATE TABLESPACE or INDEX.

The following new messages are also introduced:

```
DSNU1256I csect-name - VIRTUAL STORAGE REQUIRED FOR SPANNED RECORDS EXCEEDS
DEFINED LIMIT.

DSNU1257I csect-name - REFERENCE TO LOB OR XML DATA IN A WHEN CLAUSE IS NOT
ALLOWED IF THE LOAD STATEMENT SPECIFIES FORMAT SPANNED YES.

DSNU1258I csect-name - ORDER OF FIELDS IS NOT VALID FOR KEYWORD SPANNED.

DSNU1259I csect-name - THE INPUT DATA SET DOES NOT HAVE THE ATTRIBUTE SPANNED
```

13.11.2 File reference variable enhancement for 0 length LOBs

Prior to DB2 10, unloading empty LOBs through file reference variables results in SYSREC recording a file name of an empty file. UNLOAD creates a file for every empty LOB. LOAD issues an error on a blank or zero length file reference variables. This behavior can cause performance issues, because DB2 must open and close the data set for an empty LOB. DB2 Version 8 also does not determine emptiness before inserting the base row. Thus, DB2 inserts an empty LOB (occupying a page) in the LOB table space and does not mark the column indicator as empty.

In some cases, you might be able to mark LOB columns as NULL instead of empty, but in some cases the LOB column is not nullable. Also, for some applications, NULL and empty (or zero length) have different meanings.

DB2 10 changes the way UNLOAD and LOAD handle file reference variables for empty LOBs. An empty LOB is a LOB with a length of zero. It has no data bytes and is not NULL.

When unloading an empty LOB to file reference variables, UNLOAD writes a zero length VARCHAR or blank CHAR in SYSREC and does not create a data set for the LOB. If the field is nullable, the null byte is set to '00'x, indicating not NULL.

When loading a LOB file reference variable field, LOAD treats a zero length VARCHAR, blank VARCHAR, or blank CHAR as an empty LOB. This behavior assumes the LOB is not nullable or is not NULL as indicated by the NULL byte. A NULL file reference variable results in a NULL LOB. Because LOAD can determine the length of the LOB before inserting the base row, the zero length flag in the column indicator for the LOB is set on, and an empty LOB is not inserted into AUX table space. That is, the column indicator is used to determine that the LOB is empty.

A file reference variable with a data set name that references an empty data set still represents an empty LOB. In this case, LOAD cannot determine the length of the LOB before inserting the base row, and an empty LOB is inserted into AUX table space, the same behavior as before.

This function was retrofitted by APAR PM12286 (PTF UK59680) to DB2 9, where it is even more important. In DB2 10, the preferred method of unloading LOBs is spanned records.

13.11.3 Streaming LOBs and XML between DDF and DBM1

DB2 9 introduced functionality that allowed clients to not know the total length of a LOB value before sending it to a remote DB2 for z/OS server. However, DB2 for z/OS still had to materialize the entire LOB in memory to get the length of the entire LOB before inserting it into the database. This behavior is called *LOB streaming*, and it eliminates the need for the client application to read the entire LOB to get its total length prior to sending the LOB to the server.

DB2 10 for z/OS offers additional performance improvements by extending the support for LOB and XML streaming, avoiding LOB and XML materialization in more situations. Figure 13-10 shows the differences between DB2 9 and DB2 10. The DDF server no longer needs to wait for the entire LOB or XML to be received before it can pass the LOB or XML to the data manager.

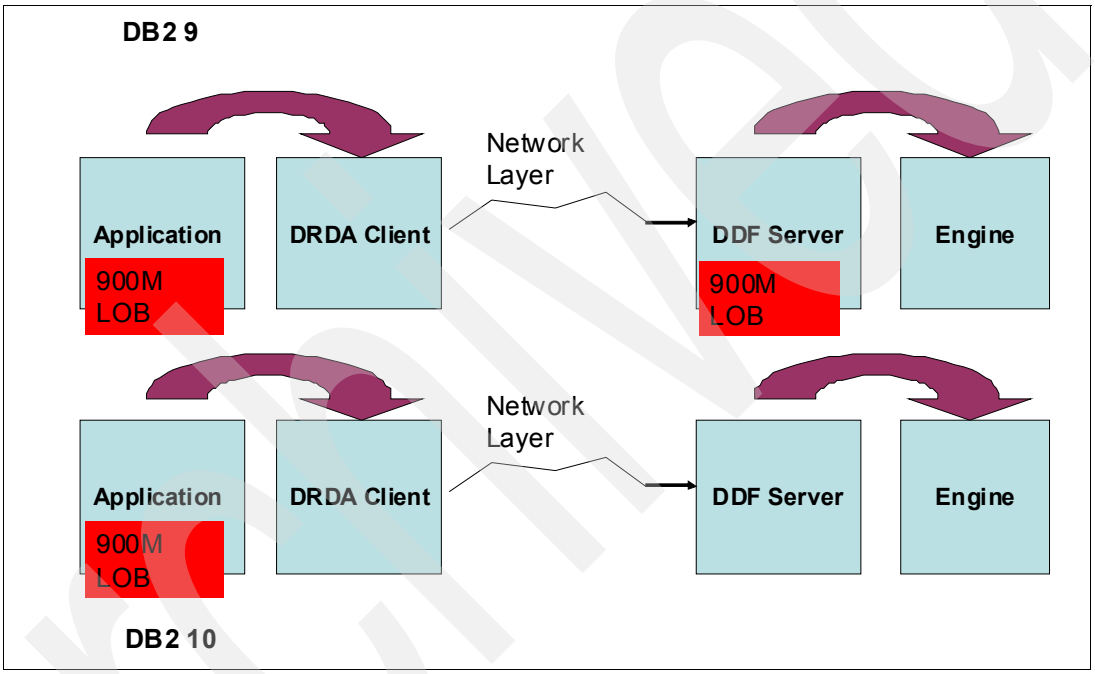


Figure 13-10 Streaming LOBs and XML

DB2 materializes up to 2 MB for LOBs and 32 KB for XML before passing the data to the database manager. The storage allocated for this LOB or XML value is reused on subsequent chunks until the entire LOB or XML is processed.

The LOAD utility also uses LOB streaming for LOBs and XML but only when file reference variables are used.

Whether materialization is reduced, and by how much, depends on the following conditions:

- JDBC 4.0 and later or ODBC/CLI V8 driver FP4 and later. If using JDBC 3.5, the application has to specify the length of the LOB or XML to be -1.
- There can be at most one LOB per row for INSERT, UPDATE, or LOAD with file reference variable.
- There can be at most one XML per row for INSERT or UPDATE with DRDA streaming.
- There can be one or more number of LOB and XML values per row with INSERT, UPDATE, or LOAD XML with file reference variables and LOB INSERT or UPDATE or using crossloader that require CCSID conversion.

- For UPDATE, an additional restriction applies in that the UPDATE must qualify just one row where a unique index is defined on the target update column.

This enhancement is available in CM and reduces the virtual storage consumption and reduces elapsed time. You can also see a reduction in class 2 CPU time.

Streaming of LOB and XML data to the DBM1 address space is also available to local applications, for example SQL INSERT UPDATE and the LOAD utility, when file reference variables are used.

13.11.4 Inline LOBs

Prior to DB2 10, DB2 for z/OS stores each LOB column, one per page, in a separate auxiliary (LOB) table space, regardless of the size of the LOB to be stored. All accesses to each LOB, (SELECT, INSERT, UPDATE, and DELETE) must access the auxiliary table space using the auxiliary index.

A requirement with LOB table spaces is that two LOB values for the same LOB column cannot share a LOB page. Thus, unlike a row in the base table, each LOB value uses a minimum of one page. For example, if some LOBs exceed 4 KB but a 4 KB page size is used to economize on disk space, then it might take two I/Os instead of one to read such a LOB, because the first page tells DB2 where the second page is.

DB2 10 supports *inline LOBs*. Depending on its size, a LOB can now reside completely in the base table space along with other non-LOB columns. Any processing of this inline LOB now does not have to access the auxiliary table space. A LOB can also reside partially in the base table space along with other non-LOB columns and partially in the LOB table space. That is, a LOB is split between base table space and LOB table space. In this case any processing of the LOB must access both the base table space and the auxiliary table space.

Inline LOBs offer the following benefits:

- Small LOBs that reside completely in the base table space can now achieve similar performance to similarly sized varchar columns.
- Inline LOBs avoid all getpages and I/Os that are associated with an auxiliary index and LOB table space.
- Inline LOBs can save disk space even if compression cannot be used on the LOB table space.
- The inline piece of the LOB can be indexed using index on expression.
- Inline LOBs access small LOB columns with dynamic prefetch.
- The inline portion of the LOB can be compressed.
- A default value other than empty string or NULL is supported.

The DB2 10 LOAD and UNLOAD utilities can load and unload the complete LOB along with other non-LOB columns.

Inline LOB support comes with DB2 10 in NFM.

Inline LOBs are only supported with either partition-by-growth or partition-by-range universal table spaces. Reordered row format is also required. Inline LOBs take advantage of the reordered row format and handle the LOB better for overall streaming and application performance. Additionally, the DEFINE NO option allows the row to be used and the data set for the LOB not to be defined. DB2 does not define the LOB data set until a LOB is saved that is too large to be completely inlined.

DB2 introduces a new parameter, INLINE LENGTH, for the column definition clause of ALTER TABLE, CREATE TABLE, ALTER TYPE, and CREATE TYPE statements. For BLOB and CLOB columns, the integer specifies the maximum number of bytes that are stored in the base table space for the column. The integer must be between 0 and 32680 (inclusive) for a BLOB or CLOB column. For a DBCLOB column, the integer specifies the maximum number of double-byte characters that are stored in the table space for the column. The integer must be between 0 and 16340 (inclusive) for a DBCLOB column.

A new version of the table is generated after the LOB's inline length is altered. Increasing the LOB length is considered an immediate change. All new rows begin using the new value, and the base table space is put in advisory REORG-pending (AREOR) status. However, if you use the ALTER TABLE statement to reduce the length of an inline column, DB2 places the base table space in REORG-pending status. The old inline quantity is used until REORG is executed. When you change the length of an inline LOB, the REORG utility always takes an inline image copy.

In addition, plans or packages and queries in the dynamic statement cache that access the altered base table are invalidated when you change the length of inline LOB columns. Any views that access these LOB columns are also regenerated.

To preserve data integrity between the base table and auxiliary table where two parts of the same LOB column can reside, the RECOVER utility now enforces recovering the base table and associated auxiliary table spaces, both LOB and XML, together to the same point in time. DB2 enforces this restriction in both CM and NFM. This is not enforced in earlier versions of DB2 although it is has always been highly recommended.

Inline LOBs allow you to index your LOB columns by indexing the inlined piece of LOB columns. DB2 10 allows you to specify LOB columns for indexes on expression; however, only the SUBSTR in built-in function is allowed. If you can search on the inlined piece of LOB columns, you no longer need to access the LOB table space to locate a specific LOB. This is achieved by exploiting index on expression.

For example, you can create a base table and an index on expression as follows:

```
CREATE TABLE mytab (clobcol CLOB(1M) INLINE LENGTH 10);
CREATE UNIQUE INDEX myindex1 on mytab (VARCHAR (SUBSTR(clobcol, 1,10))) ;
```

And you can issue the statement:

```
SELECT clobcol From mytab WHERE VARCHAR(SUBSTR(clobcol,1,10)) =  'ABCDEFGHIK'
```

If you can store the complete LOB inline, then you can potentially realize significant savings in CPU time and DASD space, by avoiding I/Os when fetching the LOB column or columns, and savings in not having to manage the auxiliary table space and index. Some preliminary performance measurements with random access to small LOBs show as much as 70% improvement on SELECTs and even higher improvement in INSERTs.

However, split LOBs incur the cost of both inline and out-of-line LOBs. Small inline LOBs use approximately 5%-10% more CPU time than VARCHAR columns. Because the base table becomes bigger, SQL that is limited to non-LOB columns can be impacted. So, there is no performance advantage except to support indexing of the LOB columns. Table scans and utilities that did not refer to the LOB columns will take longer, and image copies for the base table will become larger. (Alternatively, the image copy for the LOB table space will become smaller, or non-existent.) The buffer hit ratio for the base table can also be impacted.

DB2 10 introduces a DSNZPARM parameter, LOB_INLINE_LENGTH, to define the default maximum length to be used for the inline portion of a LOB column. This value is used when a value is not specified by the INLINE LENGTH parameter of the CREATE or ALTER TABLE

statement. The value is interpreted as number of bytes, regardless of the type of LOB column to which it is applied. The default is zero (0), meaning no inline LOBs.

Be advised that specifying a default inline LOB length can hurt the performance of some table spaces.

The following general tuning practices are practical:

- Choose the INLINE LENGTH according to the LOB distribution. A size too small takes up some more space on the base page and still requires frequent accesses to the auxiliary table space.
- Consider increasing the page size of the base table to accommodate larger rows.
- Reconsider whether to use compression for the base table. (If the inline LOBs do not compress well and if they dominate the bytes in the row, then turn off compression.)
- Retune the buffer pool configuration as needed.

Table 13-4 lists the DB2 catalog table changes to support inline LOBs.

Table 13-4 Catalog table changes for inline LOBs

Column name	Data type	Description
SYSDATATYPES.INLINE_LENGTH	INTEGER NOT NULL WITH DEFAULT	The inline length attribute of the type if it is based on a LOB source type
SYSTABLEPART.CHECKFLAG	Same as DB2 9	► D: Indicates that the inline length of the LOB column that is associated with this LOB table space was decremented when the inline length was altered. ► I: Indicates that the inline length of the LOB column that is associated with this LOB table space was incremented when the inline length was altered.
SYSCOPY.STYPE	Same as DB2 9	► I: Indicates that the inline length attribute of the LOB column was altered by REORG.
SYSCOPY.TTYPE	Same as DB2 9	When ICTYPE=A and STYPE=I, this column indicates that the inline length of a LOB column was altered: ► D: Indicates that REORG decremented the inline length of the LOB column ► I: Indicates that REORG incremented the inline length of the LOB column

13.12 Utility BSAM enhancements for extended format data sets

z/OS R9 introduced support for long-term page fixing of basic sequential access method (BSAM) buffers, and z/OS R10 introduced support for 64-bit BSAM buffers if the data set is extended format. This function frees BSAM from the CPU-time intensive work of fixing and freeing the data buffers itself.

Increasing the number of BSAM buffers uses more real storage but reduces the number of I/O operations, which can save CPU time and elapsed time. As the performance of channel and

storage hardware improves, the performance benefit of using more real storage to reduce the number of I/O operations increases. FICON Express 8, the z196 processor, and the DS8800 storage control unit are all examples of this. Thus, each new version of DB2 tries to make use of the hardware advances by increasing the I/O quantity of its sequential I/O operations.

The I/O performance is improved by reducing both the processor time and the channel start/stop time that is required to transfer data to or from virtual storage

DB2 10 utilities exploit these recent z/OS enhancements, by offering the following enhancements:

- Allocating 64-bit buffers for BSAM data sets
- Allocating more BSAM buffers for faster I/O
- Long term page fixing BSAM buffers

DB2 10 utilities, when opening BSAM data sets, increase MULTSDN from 6 to 10, and MULTACC from 3 to 5. This increase enables DB2 to reduce the number of physical I/Os by up to 40%.

MULTACC allows the system to process BSAM I/O requests more efficiently by not starting I/O until a number of buffers are presented to BSAM.

MULTSDN is used to give a hint to OPEN processing so it can calculate a better default value for NCP[2] instead of 1, 2, or 5.

These performance improvements are available in CM, and the data sets need to be defined as extended format data sets to take advantage of these enhancements.

Unlike DB2 buffers, the BSAM buffers never go dormant. So, if real storage is overcommitted and many DB2 9 utilities are running, the utilities are thrashing real storage. Therefore, long term page fixing the BSAM buffers in DB2 10 does not present a real risk to your system.

Note also that if you currently explicitly specify BUFNO on your data set allocations, your value overrides MULTSDN so nothing changes.

All utilities using BSAM benefit from this change, For example, Copy and Unload are faster in elapsed time. Copy, Unload, Load, and Recover all have less CPU time than they have without these enhancements.

The elapsed time benefits are expected to increase as the hardware speeds up sequential I/O.

13.13 Performance enhancements for local Java and ODBC applications

Before DB2 10, Java and ODBC applications running locally on z/OS did not always perform faster than the same application called remotely. This behavior is because the optimizations built over the past few DB2 versions for DDF processing with the DBM1 address space have not been available to local JDBC and ODBC applications. zIIP redirect of distributed workloads also reduced the chargeable CP consumption of distributed Java applications significantly.

[2] Number of Channel Programs, the maximum number of blocks allowed per BSAM I/O operation.

DB2 10 brings optimization functions already in place between the DDF and DBM1 address spaces and that are exploited by distributed JDBC Type 4 applications, to local JCC type 2 and ODBC z/OS driver applications:

- Limited block fetch
- LOB progressive streaming
- Implicit CLOSE

DB2 triggers block fetch for static SQL only when it can detect that no updates or deletes are in the application. For dynamic statements, because DB2 cannot detect what follows in the program, the decision to use block fetch is based on the declaration of the cursor.

To use either limited block fetch (or continuous block fetch), DB2 must determine that the cursor is not used for updating or deleting. The easiest way to indicate that the cursor does not modify data is to add the FOR FETCH ONLY or FOR READ ONLY clause to the query in the DECLARE CURSOR statement. Remember also that CURRENTDATA(NO) is required to interpret ambiguous cursors as read only.

Additional information: Limited block fetch and continuous block fetch are described in more detail in *DB2 10 for z/OS Managing Performance,* SC19-2978.

With limited block fetch, DB2 attempts to fit as many rows as possible in a query block. DB2 then transmits the block of rows to your application. Data can also be pre-fetched when the cursor is opened without needing to wait for an explicit fetch request from the requester. Processing is synchronous. The java/ODBC driver sends a request to DB2, which causes DB2 to send a response back to the driver. DB2 must then wait for another request to tell it what should be done next.

Block fetch is controlled by the OPTIMIZE FOR n ROWS clause. When you specify OPTIMIZE FOR n ROWS, DB2 prefetches and returns only as many rows as fit into the query block.

You can use the FETCH FIRST n ROWS ONLY clause of a SELECT statement to limit the number of rows that are returned. However, the FETCH FIRST n ROWS ONLY clause does not affect blocking. This clause improves performance of applications when you need no more than n rows from a potentially large result table.

If you specify FETCH FIRST n ROWS ONLY and do not specify OPTIMIZE FOR n ROWS, the access path for the statement uses the value that is specified for FETCH FIRST n ROWS ONLY for optimization. However, DB2 does not consider blocking. When you specify both the FETCH FIRST n ROWS ONLY and the OPTIMIZE FOR m ROWS clauses in a statement, DB2 uses the value that you specify for OPTIMIZE FOR m ROWS for blocking, even if that value is larger than the value that you specify for the FETCH FIRST n ROWS ONLY clause.

Limited block fetch requires less fetches, which in turn results in a significant increase in throughput and a significant decrease in CPU time. These savings are now realized by local JCC T2 and local ODBC applications.

LOB and XML progressive streaming is described in more detail in 13.11.3, “Streaming LOBs and XML between DDF and DBM1” on page 568.

Your application needs client side support to take advantage of LOB or XML progressive streaming. JDBC 4.0 and later is needed. If using JDBC 3.5, the application has to specify the length of the LOB or XML to be -1. CLI users must utilize the StreamPutData API (Version 8 driver FP 4 and later).

Fast implicit close means that DB2 automatically closes the cursor after it prefetches the *n*th row if you specify FETCH FIRST n ROWS ONLY or when there are no more rows to return. Fast implicit close can improve performance because it saves an additional interaction between DB2 and your address space.

DB2 uses fast implicit close when the following conditions are true:

- The query uses limited block fetch
- The query retrieves no LOBs
- The cursor is not a scrollable cursor
- Either of the following conditions is true
  - The cursor is declared WITH HOLD, and the package or plan that contains the cursor is bound with the KEEPDYNAMIC(YES) option.
  - The cursor is not defined WITH HOLD.

You can expect to see significant performance improvement for applications with queries that return more than 1 row and with queries that return LOBs.

13.14 Logging enhancements

As machines get faster and faster and as DB2 can do more and more work, latch class 19 contention can become a concern. This condition is especially true for systems with high logging rates, such as applications that have heavy INSERT workloads. DB2 10 includes the following enhancements, all active in CM, that reduce logging delays:

- Long term page fix log buffers
- LOG I/O enhancements
- Log latch contention reduction

13.14.1 Long term page fix log buffers

DB2 10 page fixes the log buffers permanently in memory. On DB2 start, all the buffers as specified by OUTBUFF are page fixed. The OUTPUT BUFFER field of installation panel DSNTIPL allows you to specify the size of the output buffer that is used for writing active log data sets. The maximum size of this buffer is 400,000 KB, the default is 4,000 KB (4 MB), and the minimum is 400 KB.

Generally, the default value is sufficient for good write performance. Increasing OUTBUFF beyond the DB2 10 default might improve log read performance. For example, ROLLBACK and RECOVER with the new BACKOUT option can benefit by finding more data in the buffers. COPY with the CONSISTENT option might benefit too.

Whether a large OUTBUFF is desirable depends on the tradeoff between log read performance (especially in case of a long running transaction failing to COMMIT) versus real storage consumption. Review your OUTBUFF parameter to ensure that it is set to a realistic trade-off value.[3] If OUTBUFF is too large (because of low LOG activity), then you can monopolize real frames that might be put to better use elsewhere in your system.

[3] The QJSTWTB block in the QJST section of IFCID 001 can indicate if the log buffer is too small. This counter represents the number of times that a log record write request waits because there is not available log buffers.

13.14.2 LOG I/O enhancements

Today's enterprise-class controllers are considerably more reliable than controllers built in the 1980s. Battery backed non-volatile memory has provided the improved reliability making sure that data is not lost in the event of a system wide power outage. DB2 10 improves log I/O performance by taking advantage of this improved reliability.

With *DB2 9*, if a COMMIT needs to rewrite page 10, along with pages 11, 12, and 13, to the DB2 log, DB2 first serially writes page 10 to log 1, then serially writes page 10 to log 2, and then writes pages 11 through 13 to log 1 and log 2 in parallel. In effect, DB2 does four I/Os and waits for the duration of three I/Os.
The first time a log control interval is written to disk, the write I/Os to the log data sets are performed in parallel. However, if the same 4 KB log control interval is again written to disk, the write I/Os to the log data sets must be done serially to prevent any possibility of losing log data in case of I/O errors on both copies simultaneously.

Current DASD technology writes I/Os to a new area of DASD cache each time rather than disk. There is no possibility of a log page being corrupted when it is being re-written. So, *DB2 10* simply writes all four pages to log 1 and log 2 in parallel. Hence, DB2 10 writes these pages in two I/Os, and waits for the duration of only one I/O (that is, whichever of the two I/Os takes the longer.)

DB2 10 takes advantage of the non-volatile cache architecture of the I/O subsystem. DB2 rewrites the page asynchronously to both active log data sets. In the this example, DB2 chains the write for page 10 with the write requests for pages 11, 12, and 13. Thus, DB2 10 reduces the number of log I/Os and improves the I/O overlap.

13.14.3 Log latch contention reduction

Since DB2 Version 1, DB2 has used a single latch for the entire DB2 subsystem to serialize updates to the log buffers when a log record needs to be created. Basically, the latch is obtained, an RBA range is allocated, the log record is moved into the log buffer, and then the latch is released.

DB2 10 makes several changes to the way this latch process works, which increases logging throughput significantly and reduces latch class 19 contention. The changes improve the latch management and reduce the time that the latch is held.

13.15 Hash access

DB2 10 introduces *hash access* data organization for fast efficient access to individual rows using fully qualified unique keys rather than using an index.

Consider a typical access path for randomly accessing an individual row through an index. If the index has five levels, DB2 needs to perform a getpage each level accessed as DB2 traverses the index tree performing an index probe using the key. So, DB2 might need as many as five getpage requests to traverse the index and then another getpage request to access the row pointed to by the Index. Even if the index was built to provide index only access, five getpage requests might still be needed. Assuming the top three levels of the index are already in the buffer pool, then DB2 might need as many as three physical I/Os to access the right row.

The hash access provides a more efficient access to the data. The hashing routine points directly to the page the data is on so the access only takes 1 I/O. In fact, hashing methods

have been used in IBM and in DB2 (access to DB2 directory) for many years to provide fast access. DB2 10 introduces a hashing algorithm for access to user data.

As the name suggests, hash access employs the use of a DB2 predetermined hashing technique to transform the key into a physical location of the row. So, a query with an equal predicate can almost always access the row with a single getpage request and probably a single I/O. Hash access provides a reduction in CPU and elapsed time because the CPU used to compute the row location is small compared with the CPU expended on index tree traversal and all the getpages requests, together with the reduction in physical I/O.

The table space for a hash table consists of the fixed hash space and the overflow space, as shown in Figure 13-11.

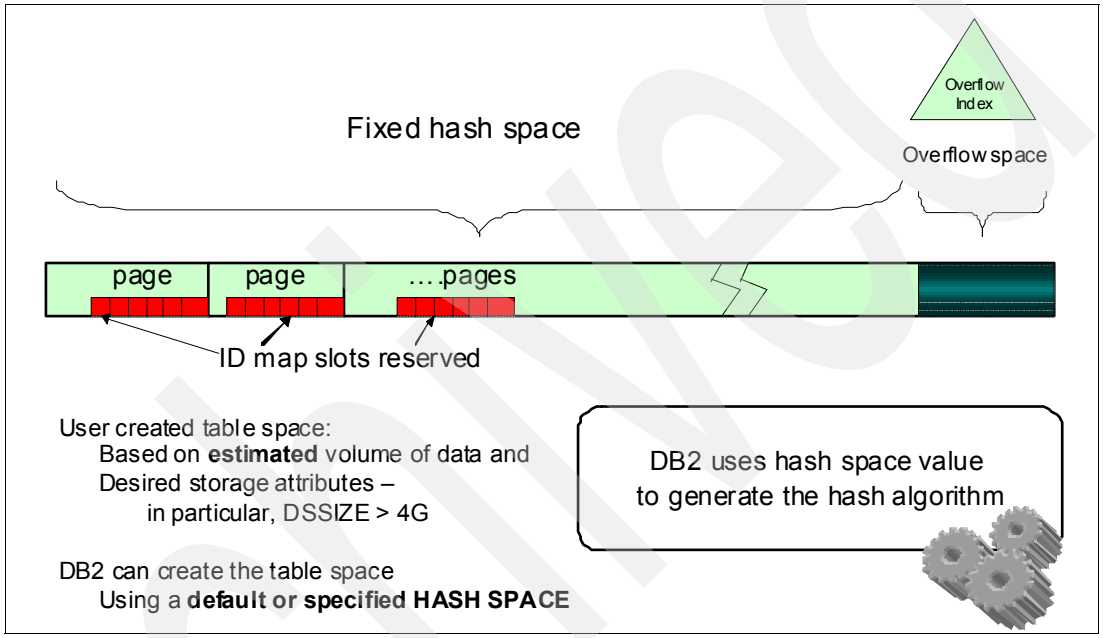


Figure 13-11 Hash space structure

On each data page, a number of ID map slots are reserved. The hashing algorithm uses the key of the row to work out which page and ID map entry on the page to use. In the case of an insert, the data is inserted on the page, and the displacement of that location is written to the ID map. It is possible that the hashing algorithm can point more than one key to the same page and ID map entry. When this situation occurs, a pointer record is set up to point to where the row was placed. The two rows are conceptually chained off the same RID entry. If there is no space on the data page, then the row is put into the overflow space, and an index entry is created to point to this row.

The fixed hash space must remain fixed because the hashing algorithm uses this space to ensure that the calculated target page for the row stays within the fixed area.

Although the two are similar, a hash table is different from an Information Management System (IMS) HDAM database. IMS data is held in segments. Each segment can equate to a row in a DB2 table. Each segment is linked to other segments using pointers. The IMS segments represent a hierarchical structure.

An example of hierarchical structure is a bank that has many customers. Each customer can have a number of accounts, and each account can have a number of transactions that are related to each account. Thus, an IMS database can have a Customer segment (the root segment) with an Account segment under the Customer segment for each account that the

customer holds. Each Account segment can have a number of Transaction segments or can have no Transaction segments, if there are no transactions.

In an IMS HDAM database, either DFSHDC40 (the default randomizes) or a home grown one is used to calculate which root anchor point (RAP) in which control interval the key of the root segment is to be placed. If there is already a root segment with another key pointed to by this RAP, then the new root segment is inserted in the next available space in the control interval, and a 4-byte pointer points from the first root to the second and so on. If the control interval is full, then IMS places the new root segment in the overflow area, which is pointed to from the root addressable area.

A hash table can exist only in a universal table space, either a partition-by-growth or partition-by-range table space, as shown in Figure 13-12.

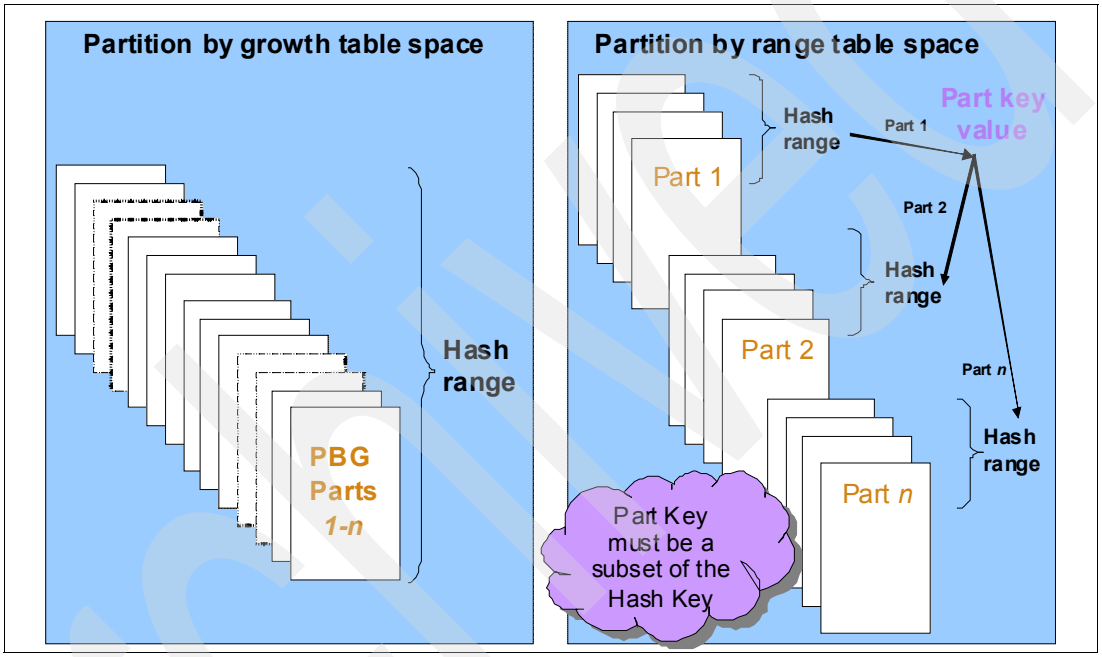


Figure 13-12 Hash access and partitioning

For a range-partitioned table space, the partitioning key columns and hashed columns must be the same; however, you can specify extra columns to hash. DB2 determines how many pages are within each partition and allocates rows to partitions based on the partitioning key. Row location within the partition is determined by the hash algorithm.

For a partition-by-growth table space, DB2 determines how many partitions and pages are needed. Rows are inserted or loaded in the partition and page according to hash algorithm. Rows are, therefore, scattered throughout the entire table space.

Only reordered row format is supported. Basic row format is not supported nor is MEMBER CLUSTER or the table APPEND option.

A hash table is expected to consume between 1.2 and 2 times the storage to obtain almost single page access. Single page access is dependent on no rows being relocated to the overflow space due to lack of space in the fixed hash access space. Thus, there is a trade-off between extra DASD usage and performance.

Although the unique key is used as the hash key, this key might not be the table's traditional primary key. A hash table can have other indexes defined.

In addition, hash key column values cannot be updated. The row must be deleted and reinserted with the new key value.

Hash tables are not a good choice for all applications and should be viewed as yet another design tool in the DBA toolbox. Use hash tables selectively to solve specific design problems. In fact, a poor choice for hash access can severely impact performance through “death by random I/O” because there is no concept of clustering.

Hash access is a good option for tables with a unique key and a known approximate number of rows. Although the number of rows can be volatile through heavy inserts, updates, or deletes, the space that the table space occupies must remain the same. Queries must be single row access that use equals predicates on the unique key, such as OLTP applications. Table spaces that have access requirements that are dependent on clustering are poor choices.

For example, using the banking example that we described earlier, a transaction to select the customers account details using the account_id (the unique key) is a good candidate for hash access. Sometimes, when you choose a hash organization, some queries perform better and some perform worse. In this case, you might want to consider the overall performance to decide whether a hash organization is best for you.

A transaction to list all the transactions for that account between a given period (either by a between clause or < and > clauses) cannot use hash access. An application that accesses a table only using a unique key is the perfect candidate for hash access; however, these situations are rare.

Consider the following starting approach to find candidate tables:

1. Find all the tables with unique indexes.
2. Disqualify the tables with small indexes. Target indexes with at least three levels. Large tables with indexes with many levels have the greater potential for CPU saving.
3. Check the statements against the tables to see if they are using the unique index.

   Statements with fully qualified equal predicates that use the unique index keys are good. For example, if you have three column composite indexes but uniqueness is determined by the first two columns, two qualified predicates are still acceptable. For joins, as long as the unique index is used to qualify a row from the inner or outer table, it is still good.

   Not fully qualified or non-equal (range) predicates are not suitable. If the statements use the non-unique index with high cluster ratio, this is also a poor choice.

Important: To avoid disappointment and “death by random I/O,” we suggest that you take time to thoroughly research table accesses before choosing candidates for hash access. Hash access works best for truly random access where there is *no* access that is dependent on clustering at all. Be warned that you might have access patterns to data that you do not even know exist.

Determining whether any queries depend on the clustering of the cluster index is not easy. If you want to know if prefetch was ever used for the cluster index, after you have found a good candidate table, you might consider moving the cluster index to a separate buffer pool to check the use of prefetch.

Choose tables where the maximum size is somewhat predictable such that the estimated hash space is also predictable within a range of +/- 15-20%. Choose at least 20% extra space for the hash space, to allow for efficient hashing. The average row size should not be too small (less than 30 bytes) or too large (larger than 2000 bytes). Smaller rows can be used

with hash access, but they will require more space. In general, tables with only a few rows per page are prone to having more overflow rows and therefore do not perform quite as well.

When a row is updated and increases in size but cannot fit in the original page, the updated row must be stored in the overflow area. Any time that DB2 hashes to the original page, it contains the RID of the new page in the overflow.

When converting to hash access, it is possible to retain the old cluster index as a non-cluster index, which implies that the cluster ratio will be poor. If the optimizer selects that index, it can choose list prefetch to prefetch the rows. Furthermore, because the list prefetch access path destroys the ordering of the index keys, switching to list prefetch can require a sort where none was needed before. That sort might cause the SQL statement to use more CPU time.

13.15.1 The hashing definitions

You can either create tables for hash access or alter existing universal table spaces, by specifying the ORGANIZE BY HASH clause on the CREATE TABLE and ALTER TABLE statements respectively, as shown in Figure 13-13.

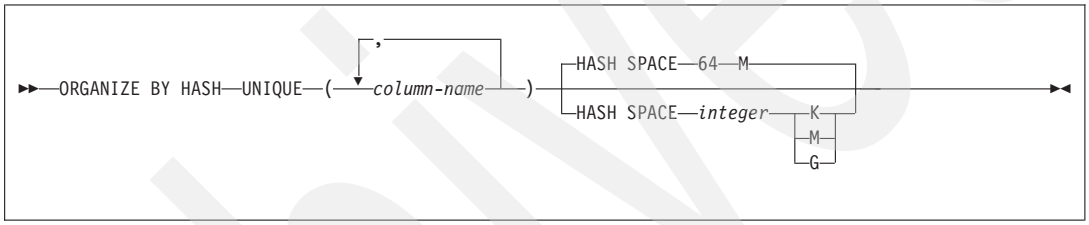


Figure 13-13 ORGANIZE BY HASH clause

We examine the hashing clauses in the next sections.

ORGANIZE BY HASH UNIQUE(column-name,...)

This clause defines a list of column names which form the hash key that is used to determine where a row will be placed. The columns must be defined as NOT NULL, the number of columns cannot exceed 64, and the sum of their lengths cannot exceed 255. In addition, the columns cannot be a LOB, DECFLOAT, or XML data type or a distinct type that is based on one of these data types. For range-partitioned table spaces, the list of column names must specify all of the partitioning columns in the same order, however more columns can be specified. Remember that for range-partitioned table spaces, the partitioning columns define the partition number where the row is to be inserted.

HASH SPACE integerK|M|G

This clause specifies the amount of fixed hash space to preallocate for the table. If the table is a range-partitioned table space, this value is the space for each partition. The default value is 64 MB for a table in a partition-by-growth table space or 64 MB for each partition of a range-partitioned table space. If you change the hash space using the ALTER TABLE statement, the hash space value is applied when the table space is reorganized using the REORG utility.

Do not over-allocate the hash space area so that there is little or no room left in the table space for the overflow area. When you run out of space in the overflow area, inserts can fail with a resource unavailable message.

You can also specify the hash space at the partition level, as shown by Figure 13-14.

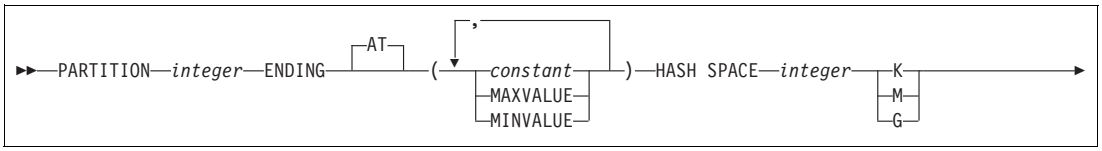


Figure 13-14 ORGANIZE BY HASH Partition clause

The create statements to create a hash table might take more time than other create table statements because DB2 must allocate the whole hash space.

DB2 also provides a DROP ORGANIZATION clause on the ALTER TABLE statement to remove hash access from a table; however, this clause will cause an outage. The entire table becomes inaccessible and is placed in REORG-pending status. You must run REORG to make the table accessible. If the table is in a range-partitioned table space, the entire table space must be reorganized at one time.

If you need to change the columns that are specified for the hash key for a table, you must first remove hash access by using ALTER DROP ORGANIZATION. Then, you can specify the new columns for the hash key with an ALTER ADD organization-clause followed by a REORG.

When you create a hash table, DB2 implicitly creates an index for rows that fall into the overflow area. The index is, therefore, a sparse index and contains only few entries, if any at all. A partitioned index is defined for a range-partitioned table space and an NPI is defined for a partition-by-growth table space. DB2 estimates the space required for these indexes.

When you run the ALTER TABLE ADD ORGANIZE BY HASH statement to convert an existing table to hash access, DB2 also implicitly creates the unique hash overflow index on the hash key and places the index in rebuild pending (RBDP). The table is also placed in advisory REORG-pending status (AREOR). You can delete rows from the table or update rows as long as the hash key columns are not updated. However, you cannot insert rows into the table. At this time, you can run REBUILD index on the overflow index which removes the rebuild pending status and allow inserts, however the underlying organization has not changed to hash access. The table remains in advisory REORG-pending state. Instead of running rebuild index, you can choose to run REORG directly after running the ALTER TABLE statement. (REORG SHARELEVEL NONE is not supported.)

Finally, you need to rebind or bind your applications to cause those suitable candidates to use hash access to the table.

13.15.2 Using hash access

Although the table is defined as a hash table, you can still create other normal DB2 indexes on it; however, you do not need to create an index to enforce uniqueness. DB2 also still considers all other access path options for the hash table.

Hash access is chosen as the access path when the SELECT statement includes values for all those columns in the key. These values are compared using an equals predicate or through an IN list. You can check whether hash access is chosen by reviewing the PLAN_TABLE for the new values in ACCESSTYPE. When ACCESSTYPE is 'H', or 'HN',[4] or 'MH', the ACCESSNAME contains the name of the hash overflow index.

Here are some sample queries that are candidates for hash access:

[4] The values 'HN' and 'MH' are used to evaluate IN predicates.

```
Simple EQUAL predicates (whth literals or host variables or place markers)
SELECT * FROM EMPTAB WHERE WHERE EMPNO = 123

IN list predicates (can have IN list)
SELECT * FROM EMPTAB WHERE EMPNO IN (123, 456)

OR predicates
SELECT * FROM EMPTAB WHERE EMPNO = 123 OR EMPNO = 456

With predicates against other columns in the table (EG: DEPTNO)
SELECT * FROM EMPTAB WHERE EMPNO = 123 AND DEPTNO = 'A001'

Multi column hash key where all columns are specified
(EG: Hash key on COMPONENT, PARTNO)
SELECT * FROM PARTTAB WHERE COMPONENT = 'TOWEL' AND PARTNO = 123
SELECT * FROM PARTTAB WHERE COMPONENT IN ('TOWEL', 'BOOK') AND PARTNO = 123
```

Hash access is not chosen if all columns in a multi column hash key are not specified. The following statements do not qualify for hash access:

```
SELECT * FROM PARTTAB WHERE COMPONENT = 'TOWEL' OR PARTNO = 123
SELECT * FROM PARTTAB WHERE COMPONENT = 'TOWEL'
SELECT * FROM PARTTAB WHERE COMPONENT = 'TOWEL' AND PARTNO > 123
SELECT * FROM PARTTAB WHERE COMPONENT NOT IN ('TOWEL', 'BOOK') AND PARTNO = 123
```

Hash access can be chosen for access to the inner table of a nested loop join. However, hash access is not used in star joins. DB2 also restricts the use of parallelism with hash access in specific cases. Although hash access itself does not use parallelism, other access paths on a table organized by hash can use parallelism, if applicable. Parallel update of indexes works as usual. If hash access is selected for a table in a parallel group, parallelism is not selected for that parallel group. Parallelism can be selected for other parallel groups in the query.

Used with the correct application, hash access offers the following performance opportunities:

- Faster access to the row using the hash key rather than using an index, with less getpages, fewer I/Os, and CPU saved in searching for the row.
- A possible saving in index maintenance if an existing unique index is not used for (skipped) sequential processing, allowing for faster insert and delete processing. However, dropping of such an index must be done carefully by monitoring for some time Real Time Statistics SYSINDEXSPACESTATS.LASTUSED.

Be aware of the following considerations:

- Slower LOAD of data into a HASH table versus non-HASH because of the non sorted distribution.
- You might see a possible INCREASE in I/O or buffer pool space in some cases. This situation can happen if the active rows were colocated or clustered without hashing, but now with hashing the rows are spread randomly across the whole hash area. Many more pages can be touched and possibly the working group might no longer fit in the buffer pool, increasing the CPU usage, I/O, and elapsed time.
- If you encounter an increase in synchronous I/Os, check your original index access. Does it have clustering in the accesses of which you were not aware? You might also consider a larger buffer pool size or not using hashed tables.
- Member cluster and clustering indexes cannot be used because the hash determines the placement of the rows. Any performance benefits from these clustering methods is lost.

- Performance degrades as space becomes over-used. The degradation should be gradual, not dramatic. So, be careful with spaces that increase in size.

Table 13-5 summarizes the new columns and column values to various DB2 catalog tables to support hash access.

Table 13-5 Catalog table changes for hash access

Column name	Data type	Description
SYSTABLESPACE. ORGANIZATIONTYPE	CHAR(1) NOT NULL WITH DEFAULT	Type of table space organization: ► blank: Not known (default) ► H: Hash organization
SYSTABLESPACE. HASHSPACE	BIGINT NOT NULL WITH DEFAULT	The amount of space, in KB, specified by the user that is to be allocated to the table space or partition For partition-by-growth table spaces the space applies to the whole table space. For range-partitioned table spaces, the space is applicable for each partition.
SYSTABLESPACE. HASHDATAPAGES	BIGINT NOT NULL WITH DEFAULT	The total number of hash data pages to be pre-allocate for hash space. For partition-by-growth, this includes all pages in the fixed part of the table space. For range-partitioned, this is the number of pages in the fixed hash space in each partition unless it is overridden by providing hash space at the part level. This is calculated by DB2 from HASH SPACE or at REORG with automatic estimation of space and it is used in the hash algorithm. The value is zero for non-hash table spaces. The value is zero for table spaces which have been altered for hash access but have not been reorganized.
SYSTABLEPART. HASHSPACE	BIGINT NOT NULL WITH DEFAULT	For partition-by-growth table spaces this is zero. For range-partitioned table spaces, this is the amount of space, in KB, specified by the user at the partition level to override the space specification at the general level. If no override is provided it will be the same as SYSTABLEPSACE. HASHSPACE.
SYSTABLEPART. HASHDATAPAGES	BIGINT NOT NULL WITH DEFAULT	For partition-by-growth table spaces, this is zero. For range-partitioned table spaces, this is the number of hash data pages corresponding to SYSTABLEPART. HASHSPACE for each part. The value is zer0 for table spaces which have been altered for hash access but have not been reorganized
SYSCOLUMNS. HASHKEY_COLSEQ	SMALLINT NOT NULL WITH DEFAULT	The column's numeric position within the table's hash key. The value is zero if the column is not part of the hash key. This column is applicable only if the table that use hash organization.
SYSTABLES. HASHKEYCOLUMNS	SMALLINT NOT NULL WITH DEFAULT	The number of columns in the hash key of the table. The value is zer0 if the row describes a view, an alias, or a created temporary table.

Column name	Data type	Description
SYSINDEXES. UNIQUERULE	SMALLINT NOT NULL WITH DEFAULT	A new value of “C” is used to enforce the uniqueness of hash key columns.
SYSINDEXES. INDEXTYPE	SMALLINT NOT NULL WITH DEFAULT	A value of “2” indicates a Type 2 index or a hash overflow index
SYSINDEXES. HASH	CHAR(1) NOT NULL WITH DEFAULT	Whether the index is the hash overflow index for a hash table.
SYSINDEXES. SPARSE	CHAR(1) NOT NULL WITH DEFAULT	Whether the index is sparse. ► N: No (default). Every data row has an index entry. ► Y: Yes. This index might not have an entry for each data row in the table.
SYSTABLESPACESTATS. REORGSCANACCESS	BIGINT	The number of times data is accessed for SELECT, FETCH, searched UPDATE, or searched DELETE since the last CREATE, LOAD REPLACE or REORG. A null value indicates that the number of times data is accessed is unknown.
SYSTABLESPACESTATS. REORGHASHACCESS	BIGINT	The number of times data is accessed using hash access for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints since the last CREATE, LOAD REPLACE or REORG. A null value indicates that the number of times data is accessed is unknown.
SYSTABLESPACESTATS. HASHLASTUSED	DATE NOT NULL WITH DEFAULT 1/1/0001	The date when hash access was last used for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints.
SYSINDEXSPACESTATS. REORGCLUSTERSENS	BIGINT	The number of times the index was used for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints. For hash overflow indexes, this is the number of times DB2 has used the hash overflow index. A null value indicates that the number of times the index has been used is unknown.
SYSCOPY. STYPE	same as before	A new value of “H” indicates the hash organization attributes of the table were altered.
SYSCOPY. TTYPE	same as before	When ICTYPE=W or X and STYPE=H, this column indicates the prior value of HASHDATAPAGES.

DB2 follows the normal locking scheme based on the applications or SQL statement’s isolation level and the LOCKSIZE definition of the table space. However, a new lock type of CHAIN is introduced to serialize hash collision chain updates when the page latch is not sufficient as in the case when the hash collision chain continues to the hash overflow area.

Hash overflow indexes are a bit different than normal indexes in that they do not have an entry for every row in the table. So, RUNSTATS handles these indexes like XML node indexes and avoids updating column statistics (SYSCOLUMNS). These indexes are sparse, and any statistics collected might not be usable.

The biggest percentage DB2 CPU reduction is for SELECT SQL call, followed by OPEN/FETCH/CLOSE and UPDATE, and then other SQL calls. However, hash tables are not for general use for the following reasons:

- Might require more space or larger buffer pools in some cases, especially if there is a small active working set.
- Are not for sequential insert (no MEMBER CLUSTER support).
- Performance slowly degrades as space becomes over-used. Monitor monitor utilization of space and the number of overflow entries regularly to determine when to REORG.
- Take longer to be created because the entire hash table has to be preformatted.
- Utility performance of LOAD is slower, because the rows are loaded according to the hash key rather than LOAD using the input in clustering order.

  Consider the following method as a performance alternative to loading data into a hash table:

  a. CREATE the table an non-hash.
  b. LOAD the data.
  c. ALTER the table to hash access.
  d. REORG the entire table space.
- Update of a hash key is not allowed because it requires the row to move physically from one location to another. This move causes DB2 semantic issues, because you can see the same row twice or miss it completely. (The same issue existed with updating partitioning keys.) DB2 returns -151 SQLCODE and 42808 SQLSTATE. The row must be deleted and reinserted with the new key value.
- Declared and global temporary tables cannot be defined as hash tables.
- REORG SHRLEVEL NONE is not supported on hash tables.

13.15.3 Monitoring the performance of hash access tables

If you decide to implement hash access for some tables, then you are most likely interested in the performance of those tables. The performance of hash tables is sensitive to the size of the hash space and to the number of rows that flow into overflow. If the fixed hash space is too small, then performance might suffer. One I/O is required to access the page pointed to by the hash routine. If there is no room on the page, then one or more I/Os are required to access the overflow Index and another I/O is required to access the data in overflow. So, ongoing monitoring of space usage is important.

During insert, update, and delete operations, DB2 real time statistics (RTS) maintains a number of statistics in the SYSTABLESPACESTATS and SYSINDEXSPACESTATS catalog tables. These statistics are relevant to monitoring the performance of hash access tables. These values are also used by the DB2 access path selection process to determine if using hash access is suitable.

- SYSIBM.SYSTABLESPACESTATS.TOTALROWS contains the actual number of rows in the table.
- SYSIBM.SYSTABLESPACESTATS.DATASIZE contains the total number of bytes used by the rows.
- SYSIBM.SYSINDEXSPACESTATS.TOTALENTRIES contains the total number of rows with keys.

TOTALROWS and DATASIZE apply for the whole table space, So, HASH SPACE from the DDL when the hash table was created should be close to DATASIZE. Ideally, TOTALENTRIES

should be less than 10% of TOTALROWS. If TOTALENTRIES is high, then you need to either ALTER the HASH SPACE and REORG the table space or let DB2 automatically recalculate the hash space upon next REORG.

The REORG utility is enhanced to support hash access tables. The AUTOESTSPACE(YES/NO) parameter directs the REORG utility to calculate the size of the hash space either by using the RTS values (the default) or by using the user-specified HASH SPACE values stored in SYSTABLESPACE and SYSTABLEPART. Note that automatic space calculation does not change the catalog values.

DB2 calculates the size of the hash space (when you specify AUTOESTSPACE YES) by estimating that about 5%-10% of the rows need to go into overflow. You can influence this by specifying a FREESPACE percentage for the table space. However, FREEPAGE is ignored.

Note: Monitor overflow entries closely.

Although the REORG utility probably cannot remove all overflow entries, run it regularly to clean up the majority of overflow entries and reset hash chains, reducing the need to look into overflow. However, if your hash area is sized appropriately, then you might be able to run REORG less often.

DB2 also maintains the following RTS columns that you need to monitor:

- SYSIBM.STSTABLESPACESTATS.REORGHASHACCESS records the number of times data is accessed using hash access for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints since the last CREATE, LOAD REPLACE, or REORG.
- SYSIBM.SYSINDEXSPACESTATS.REORGINDEXACCESS records the number of times DB2 has used the hash overflow index for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints since the last CREATE, LOAD REPLACE, or REORG.

13.15.4 New SQLCODEs to support hash access

In addition to changes to many existing SQLCODEs, DB2 10 introduces the following SQLCODEs to support hash access:

```
-20300 THE LIST OF COLUMNS SPECIFIED FOR THE <clause> CLAUSE IS NOT ALLOWED IN
COMBINATION WITH THE LIST OF COLUMNS FOR THE PARTITIONING KEY FOR THE TABLE

-20487 HASH ORGANIZATION CLAUSE IS NOT VALID FOR <table-name>

-20488 SPECIFIED HASH SPACE IS TOO LARGE FOR THE IMPLICITLY CREATED TABLE
SPACE. REASON <reason-code>. (PARTITION <partition>)
```

DB2 10 introduces the following utility messages:

```
DSNU851I csect-name RECORD rid CANNOT BE LOCATED USING HASH ACCESS ON TABLE
table-name , REASON CODE: reason

DSNU852I csect-name RECORD rid FOR PARTITION part-no CANNOT BE LOCATED USING HASH ACCESS
ON TABLE table-name , REASON CODE: reason

DSNU853I csect-name THE LOCATED RECORD rid HAS A DIFFERENT RID OR KEY THAN THE
UNLOADED ROW, REASON CODE: reason

DSNU1174I csect-name PARTITION LEVEL REORG CANNOT BE PERFORMED AFTER
```

```
ALTER ADD ORGANIZATION OR ALTER DROP ORGANIZATION

DSNU1181I csect-name THE HASH SPACE HAS BEEN PREALLOCATED AT x FOR
TABLE table-name

DSNU1182I csect-name - RTS VALUES ARE NOT AVAILABLE FOR ESTIMATING HASH
SPACE VALUE FOR table-name

DSNU1183I csect-name PARTITION LEVEL REORG CANNOT BE PERFORMED AFTER
ALTER ADD ORGANIZATION OR ALTER DROP ORGANIZATION

DSNU2700I csect-name UNLOAD PHASE STATISTICS: NUMBER OF RECORDS=xxxx
(TOTAL) , NUMBER OF RECORDS (OVERFLOW)=yyyy
```

DB2 10 includes the following reason codes:

```
00C9062E
Check Data utility identified a rid which represents a row that cannot be located in the
fixed hash space using hash key column values.

00C9062F
Check Data utility identified a rid which represents a row that cannot be located in the
hash overflow space using hash key column values to probe the hash overflow index

00C90630
Check Data utility identified a rid which represents a row that locates a different rid
using hash key column values.
```

Finally, IFCID 013 and IFCID 014 record the start and end of hash scan. See Example A-3 on page 617.

13.16 Additional non-key columns in a unique index

Consider the situation where you have an index that is used to enforce uniqueness across three columns. However, you need five columns in the index to achieve index-only access on columns that are not part of the unique constraint during queries. To achieve this situation, you need to create another index and endure the cost of more CPU time in INSERT/DELETE processing and increased storage and management costs.

DB2 10 allows you to define additional non-key columns in a unique index. You do not need the extra five column index and the processing cost of maintaining that index. DB2 in NFM includes a new INCLUDE clause on the CREATE INDEX and ALTER INDEX statements.

In DB2 9, you need to:

```
CREATE UNIQUE INDEX ix1 ON t1 (c1,c2,c3)
CREATE UNIQUE INDEX ix2 ON t1 (c1,c2,c3,c4,c5)
```

Archived

In DB2 10, use either:

```
CREATE UNIQUE INDEX ix1 ON t1 (c1,c2,c3) INCLUDE (c4,c5)
```

or

```
ALTER INDEX ix1 ADD INCLUDE (c4);
ALTER INDEX ix1 ADD INCLUDE (c5);
```

and

```
DROP INDEX ix2
```

The INCLUDE clause is only valid for UNIQUE indexes. The extra columns specified in the INCLUDE clause do not participate in the uniqueness constraint.

Indexes that participate in referential integrity either by enforcing uniqueness of primary keys or referential integrity checking, can still use indexes with extra INCLUDE columns. However, the extra columns are not used in referential integrity checking.

In addition, INCLUDE columns are not allowed in the following situations:

- Non-unique indexes
- Indexes on expression
- Partitioning index where the limit key values are specified on the CREATE INDEX statement (that is, index partitioning)
- Auxiliary indexes
- XML indexes
- Hash overflow indexes

Indexes with INCLUDEd columns cannot have additional unique columns ALTER ADDed to the index. The index needs to be dropped and recreated or created.

If you add a column to both the table and the index (using the INCLUDE clause) in the same unit of work, then the index is placed in advisory REORG-pending (AREO*) state. Otherwise, the index is placed in rebuild pending (RBDP) state or page set rebuild pending (PSRBD) state. Packages are not invalidated; however, a rebind or bind is strongly recommended.

Table 13-6 lists the DB2 catalog changes for the INCLUDEd columns.

Table 13-6 Catalog table changes for INCLUDE

Column name	Data type	Description
SYSKEYS. ORDERING	same as before	Order of the column in the key. A blank indicates this is an index based on expressions or the column is specified for the index using the INCLUDE clause
SYSINDEXES. UNIQUE_COUNT	SMALLINT NOT NULL WITH DEFAULT	The number of columns or key targets that make up the unique constraint of an index, when other non-constraint enforcing columns or key-targets exist. Otherwise, the value is zero.

Preliminary measurements with two indexes defined versus one index with INCLUDE columns show 30% CPU reduction in INSERT with same query performance using the indexes. There is a trade-off in performance of INSERT or DELETE versus SELECT. Depending on how much bigger an INCLUDE index is, there can be a significant retrieval performance regression especially in an index-only access for having fewer number of indexes to benefit INSERT/DELETE performance.

Archived

Open pageset lock and unlock of table space in an index-only access is eliminated, which has a bigger impact with exploitation of INDEX INCLUDE to promote more index-only access.

A query can be issued against the catalog for identifying possible candidate columns to be evaluated for inclusion. See Example 13-3 and note the comments in the disclaimer.

Example 13-3 Query to identify indexes for possible INCLUDE

```
//CONSOLQZ JOB 'USER=$$USER','<USERNAME:JOBNAME>',CLASS=A,
//         MSGCLASS=A,MSGLEVEL=(1,1),USER=ADMF001,REGION=0M,
//         PASSWORD=CODECODE
/*ROUTE PRINT STLVM.MS
//*********************************************************************
//*
//*  DISCLAIMER:
//*    - THIS QUERY IDENTIFIES INDEXES THAT HAVE THE SAME LEADING
//*      COLUMNS IN THE SAME ORDER AS A SUBSET INDEX THAT IS UNIQUE.
//*      WHILE IT DOES MATCH INDEXES WITH COLUMNS IN THE SAME
//*      SEQUENCE, IT:
//*      - DOES NOT CONSIDER THE ORDERING ATTRIBUTE OF EACH
//*        COLUMN (ASC, DESC OR RANDOM)
//*      - DOES NOT CONSIDER IF A SUBSET/SUPERSET INDEX IS
//*        UNIQUE WHERE NOT NULL
//*      - DOES NOT CONSIDER WHETHER THE INDEXES ARE PARTITIONED OR
//*        NOT
//*      - DOES NOT CONSIDER THE IMPACT DUE TO REMOVING THE EXISTING
//*        INDEXES AND USING THE NEW ONES
//*    - THEREFORE, USE THIS AS A GUIDE FOR INDEXES
//*      THAT SHOULD BE CONSIDERED, BUT NOT GUARANTEED, TO BE
//*      CONSOLIDATED INTO A UNIQUE INDEX WITH INCLUDE COLUMNS.
//*
//*********************************************************************
//STEP1    EXEC TSOBATCH,DB2LEV=DB2A
//SYSTSIN  DD *
DSN SYSTEM(VA1A)
RUN PROGRAM(DSNTEP3)
//SYSIN    DD *
SELECT CASE WHEN SK.COLSEQ = 1 THEN
             CAST(A.DBID AS CHAR(5))
            ELSE
             ' '
       END AS DBID,
       CASE WHEN SK.COLSEQ = 1 THEN
             SUBSTR(STRIP(A.CREATOR, TRAILING) CONCAT
                    '.' CONCAT STRIP(A.NAME,TRAILING),1,27)
            ELSE ' '
       END AS SIMPLEST_UNIQUE_IX,
       CASE WHEN SK.COLSEQ = 1 THEN
             CAST(A.OBID AS CHAR(5))
            ELSE
             ' '
       END AS OBID,
       CASE WHEN SK.COLSEQ = 1 THEN
             SUBSTR(STRIP(B.CREATOR, TRAILING) CONCAT
                    '.' CONCAT STRIP(B.NAME,TRAILING),1,27)
            ELSE
```

```
                ' '
        END AS IX_WITH_CAND_INCLUDE_COLS,
        CASE WHEN SK.COLSEQ = 1 THEN
               CAST(B.OBID AS CHAR(5))
             ELSE
               ' '
        END AS OBID,
        CASE WHEN SK.COLSEQ <= A.COLCOUNT
                  AND (SK.COLSEQ <= A.UNIQUE_COUNT OR
                       A.UNIQUE_COUNT = 0) THEN
          'UNIQUE'
        ELSE
          'INCLUDE'
        END AS COLTYPE,
        SUBSTR(SK.COLNAME,1,27) AS COLNAME
FROM SYSIBM.SYSINDEXES A,
     SYSIBM.SYSINDEXES B,
     SYSIBM.SYSKEYS    SK
WHERE A.UNIQUERULE IN ('U', 'N', 'P', 'R')
  AND B.UNIQUERULE IN ('U', 'N', 'P', 'R', 'D')
  AND (A.UNIQUERULE <> 'N' OR
      (A.UNIQUERULE = 'N' AND B.UNIQUERULE IN ('D', 'N')))
  AND NOT (A.CREATOR = B.CREATOR
       AND A.NAME = B.NAME)
  AND A.TBCREATOR = B.TBCREATOR
  AND A.TBNAME = B.TBNAME
  AND SK.IXNAME = B.NAME
  AND SK.IXCREATOR = B.CREATOR
  AND A.COLCOUNT <= B.COLCOUNT
  AND A.COLCOUNT =
                    (SELECT COUNT(*)
                     FROM SYSIBM.SYSKEYS X,
                          SYSIBM.SYSKEYS Y
                     WHERE X.IXCREATOR = A.CREATOR
                      AND  X.IXNAME = A.NAME
                      AND  Y.IXCREATOR = B.CREATOR
                      AND  Y.IXNAME = B.NAME
                      AND  X.COLNAME = Y.COLNAME
                      AND  X.COLSEQ = Y.COLSEQ)
ORDER BY A.CREATOR, A.NAME, B.CREATOR, B.NAME, SK.COLSEQ
WITH UR;
/*
//
```

Example 13-4 lists the sample output.

Example 13-4 Possible INCLUDE candidates

```
+-------------------------------------------------------------------------------------------------------+
  | DBID  |  SIMPLEST_UNIQUE_IX   | OBID  | IX_WITH_CAND_INCLUDE_COLS  | OBID  | COLTYPE |     COLNAME   |
  +-------------------------------------------------------------------------------------------------+
1_| 274   | ADMF001.IX1           | 4     | ADMF001.IX2                | 6     | UNIQUE  | C1            |
2_|       |                       |       |                            |       | UNIQUE  | C2            |
3_|       |                       |       |                            |       | UNIQUE  | C3            |
4_|       |                       |       |                            |       | INCLUDE | C4            |
5_|       |                       |       |                            |       | INCLUDE | C5            |
```

Archived

+----------+

13.17 DB2 support for solid state drive

A *solid state drive* (SSD) is a storage device that stores data on solid-state flash memory rather than traditional hard disks. A SSD contains electronics that enable them to emulate hard disk drive (HDD). These drives have no moving parts and use high-speed memory, so they are fast and energy-efficient. They have been enhanced to be usable by enterprise-class disk arrays. These drives are treated like any HDD in the array.

SSD are available and can be used for allocating table space data sets or index data sets like HDD. The use of SSD for enterprise storage is transparent to both DB2 and z/OS; however, measurements have shown that SSD can improve DB2 queries two to eight times as the result of a faster I/O rate over HDD. There are other DB2 functions that are also affected by the drive type.

The need to REORG to improve query performance is reduced when the table space is on SSD.

Refer to *Ready to Access DB2 for z/OS Data on Solid-State Drives*, REDP-4537.

DB2 10 tracks the device type on which the page sets reside. The column DRIVETYPE is added to the DB2 catalog tables SYSIBM.SYSTABLESPACESTATS and SYSIBM.SYSINDEXSPACESTATS.

Table 13-7 describes the new column. The DRIVETYPE value is inserted during CREATE. If the drive type changes after page sets are created, the value is updated synchronously by utilities. The value is also updated asynchronously by the RTS service task when the page sets are physically opened or go through extent processing or EOV processing. Finally, the value can also be updated manually by SQL.

Table 13-7 Column DRIVETYPE

Column name	Data type	Description
DRIVETYPE	CHAR(3) NOT NULL WITH DEFAULT 'HDD'	Drive type that the table space or index space data set is defined on. ► HDD: Hard Disk Drive ► SSD: Solid State Drive

For multi-volume or striped data sets the drive type is set to HDD if any volume is on HDD. Whether the data sets are DB2 managed or user managed, we recommend that all volumes that can contain secondary extents, have the same drive type as the drive type of the primary extent volume and that all pieces of a multi-piece data set be defined on volumes with the same drive type.

Column DRIVETYPE provides a catalog column that can be referenced to determine the drive type on which a data set is defined. The column is used by the stored procedure, DSNACCOX, when deciding if a REORG is warranted.

The stored procedure DSNACCOX is enhanced to consider the drive type that the data set is on when making REORG recommendations. When the data set is defined on an SSD, and queries have been run that are sensitive to clustering (REORGCLUSTERSENS not equal to 0), the threshold value is changed to two times the current unclustered insert percentage

(RRTUnclustInsPctreorg). In a future release of DB2, the optimizer, which is a cost based optimizer, can also consider the drive type in access path selection.

13.18 Extended support for the SQL procedural language

The SQL Procedural Language (SQL PL) is the language used to develop SQL procedures for the DB2 family. DB2 9 introduces native SQL procedures, by executing the SQL procedures natively (without the need of a C compiled module) inside the DBM1address space, rather than the WLM stored procedure address space.

DB2 10 extends SQL PL by allowing you to develop and debug SQL scalar functions written in SQL PL. Prior to DB2 10 NFM, an SQL scalar function is restricted to a single statement which returns the results of an expression defined within the function. This type of function is not associated with a package as the expression in inlined into the invoking SQL. DB2 10 introduces non-inline scalar functions,[5] which can also be used to contain SQL PL logic. DB2 10 also introduces support for simple SQL PL table functions.[6]

NFM also brings enhancements to native SQL procedures, introduced in version 9. Together with these functional enhancements, a number of performance optimizations are also introduced which bring significant CPU reduction to a number of commonly used areas. The SQL PL assignment statement, SET, is extended to allow you to set multiple values with a single SET statement. This is similar to the existing support for SET:host-variable statement.

Figure 13-15 shows the syntax of the SET statement. Reference to any SQL variables or parameters in the multiple assignment statement always use the values that were present before any assignment is made. Similarly, all expressions are evaluated before any assignment is made.

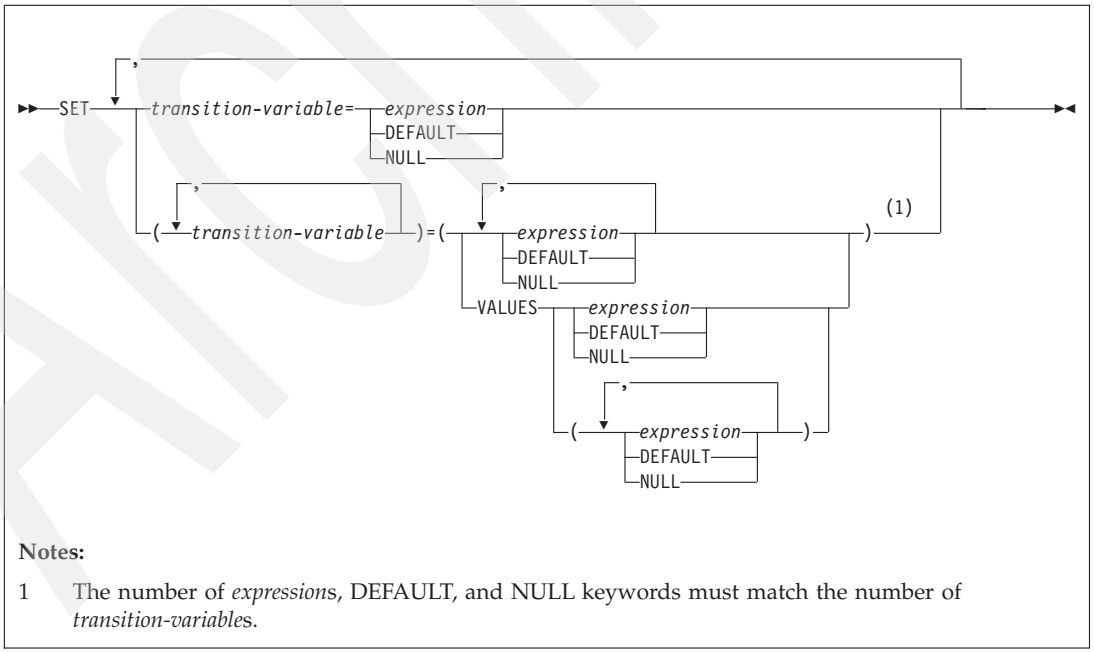


Figure 13-15 SQL PL multiple assignment statement

[5] See 6.1, “Enhanced support for SQL scalar functions” on page 126.
[6] See 6.2, “Support for SQL table functions” on page 144.

Multiple assignments together with other optimizations combine to reduce CPU consumption in commonly used statements in SQL PL procedures. Other SQL PL optimizations include; path length reduction when evaluating IF statements and, most significantly, CPU reduction with SET statement with function.

Preliminary performance results for an OLTP workload using SQL PL show a 20% reduction in CPU and a 5% improvement in response time, running in DB2 10 in CM. The SQL procedures need to be regenerated first.

Prerequisite: You need to be in CM and recreate or regenerate SQL PL procedures to take full advantage of all of these performance enhancements.

13.19 Preemptable backout

Prior to DB2 10, abort and backout processing was scheduled under a non-preemptable SRB. On a single CPU system, this processing can give the impression DB2 is hung looping. Now, DB2 creates an enclave. This enclave is used to classify the work and to enable preemptable SRB backout processing for must complete processing. Must complete processing includes DB2 abort, rollback, or backout processing and commit phase 2 processing.

Preemptable enclaves allow the z/OS dispatcher and workload manager to interrupt these tasks for more important tasks, whereas a non-preemptable unit of work can also be interrupted but must receive control after the interrupt is processed.

13.20 Eliminate mass delete locks for universal table spaces

The *mass delete* lock is used to serialize a mass delete operation with an insert at the table level. Mass delete locks are used for classic partitioned table spaces in V8, DB2 9, and DB2 10. The mass delete process gets an exclusive mass delete lock. Then, insert (if a user wanted to allocate a deallocated segment of pages) asks for a shared mass delete lock to make sure that the mass delete that released the segment is committed, eliminating any possibility that the mass delete can roll back and reallocate the segment. The mass delete lock is also used to serialize uncommitted readers on indexes with a mass delete.

DB2 10 avoids the need for a mass delete lock to be held for universal table spaces. Serialization is performed through a mass delete timestamp recorded in the header page.

In data sharing, for other members of a data sharing system to see an update to the mass delete time stamp, the data set must now be closed and re-opened. In DB2 10, a mass delete causes a UTS to be drained, which causes the table space to be closed on the other members.

However, a mass delete on a classic segmented table space still must use a mass delete lock and an X-lock on the table, instead of a drain. Thus, the data set is not closed and, therefore, other members might not see the updated mass delete time stamp. This situation presents no problem.

13.21 Parallelism enhancements

DB2 10 includes the following changes to enhance query parallelism performance:

- Remove some restrictions
- Improve the effectiveness of parallelism
- Straw model for workload distribution
- Sort merge join improvements

You can take advantage of these parallelism enhancements in CM; however, a rebind or bind is required.

13.21.1 Remove some restrictions

Two significant restrictions are removed. DB2 now allows parallelism for multi-row fetch queries when the cursor is read only. Parallelism is also allowed if a parallel group contains a work file created from view or table expression materialization.

Support for parallelism in multi-row fetch

In previous releases, parallelism is disabled when multi-row fetch is used for simple queries such as SELECT * FROM TABLE T1. You were, therefore, forced to choose between parallelism and multi-row fetch to improve the performance of such queries. You could not have both.

DB2 10 removes this restriction for multi-row fetch read-only queries only. However, parallelism is still not allowed for multi-row fetch ambiguous queries.[7]

Support for parallelism parallel group contains a work file

In previous versions, when an inner work file is used and when the number of partitioning keys and sort keys are different or they have different data types or lengths, then either parallelism is disabled or the inner work file is not partitioned. When the inner work file is not partitioned, then each parallel child task must scan through the whole work file which consumes extra CPU.

In DB2 10, if the join is not a full outer join, DB2 does not attempt to partition the inner work file. DB2 partitions only the outer table and builds a sparse index on the inner work file. Each parallel child task then uses the sparse index to position on the starting point to commence its processing.

Probing the sparse index for positioning is almost like having the work file partitioned. In this way, DB2 can exploit parallelism more often. However, this enhancement applies only to CP parallelism.

DB2 generates temporary work files when a view or table expression is materialized, however parallelism was disabled for these work files. DB2 10 now supports parallelism if a parallel group contains a work file created from view or table expression materialization. However, only CP parallelism is supported.

DB2 also generates temporary work files for sort merge join (SMJ), which also benefit from parallelism improvements in work files. See 13.21.4, “Sort merge join improvements” on page 597 for information about these improvements.

[7] A query or better a cursor is considered ambiguous if DB2 cannot tell whether it is used for update or read-only purposes.

13.21.2 Improve the effectiveness of parallelism

The DB2 optimizer currently chooses the best cost based sequential access plan, then attempts to parallelize it. So, sometimes the best sequential access plan is not the most efficient access plan for parallelism.

To evaluate parallelism, DB2 chooses key ranges decided at bind time by optimizer based on statistics (low2key, high2key, and column cardinality) and the assumption of uniform data distribution within low2key and high2key. This makes DB2 dependant on the availability and accuracy of the statistics. Also the assumption of uniform data distribution does not always stand. Figure 13-16 shows an example.

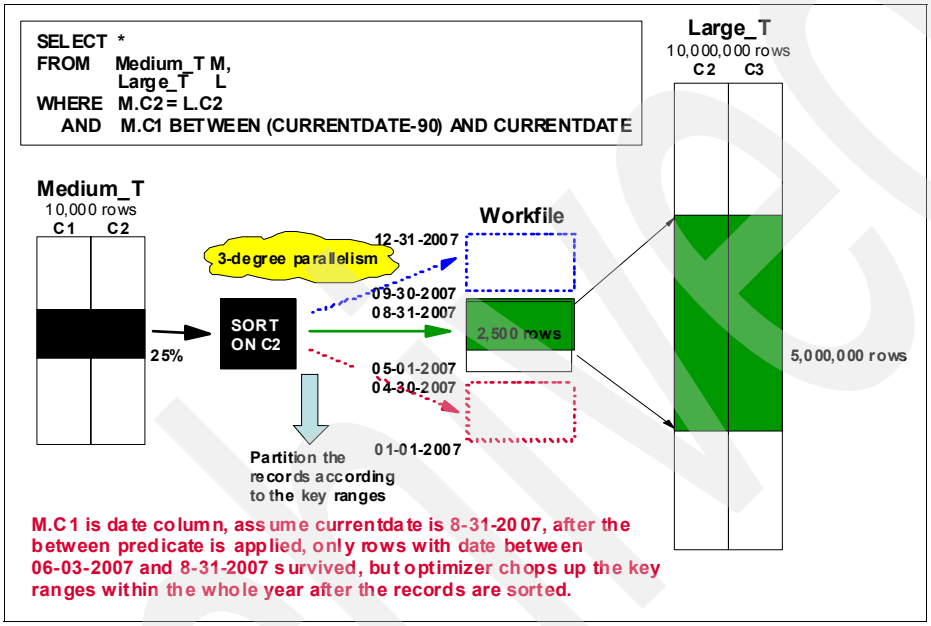


Figure 13-16 Key range partitioning

DB2 normally partitions on the leading table (Medium_T in this example) using page range or key range based on catalog statistics. Here, although at bind time DB2 chose a parallelism degree of three, the majority of the qualifying rows in the leading table are in only one parallelism group.

In previous releases, DB2 might not have had effective parallelism because of the following situations:

- Parallelism might be disabled because the leading table was a work file.
- The leading table only has one row.
- Parallelism might not be chosen for an I/O bound query if the leading table is not partitioned.
- The degree of parallelism might be limited by the column cardinality. For example, if the matching column in the leading table has a cardinality of two, then the degree of parallelism will be limited to two.

For page range partitioning, the degree of parallelism is limited to the number of pages in the leading table.

To overcome this issue, DB2 10 introduces the concept of *dynamic record range based partitioning*. DB2 materializes the intermediate result in a sequence of join process, and the results are divided into ranges with an equal number of records.

Figure 13-17 shows the same example but using dynamic record based partitioning.

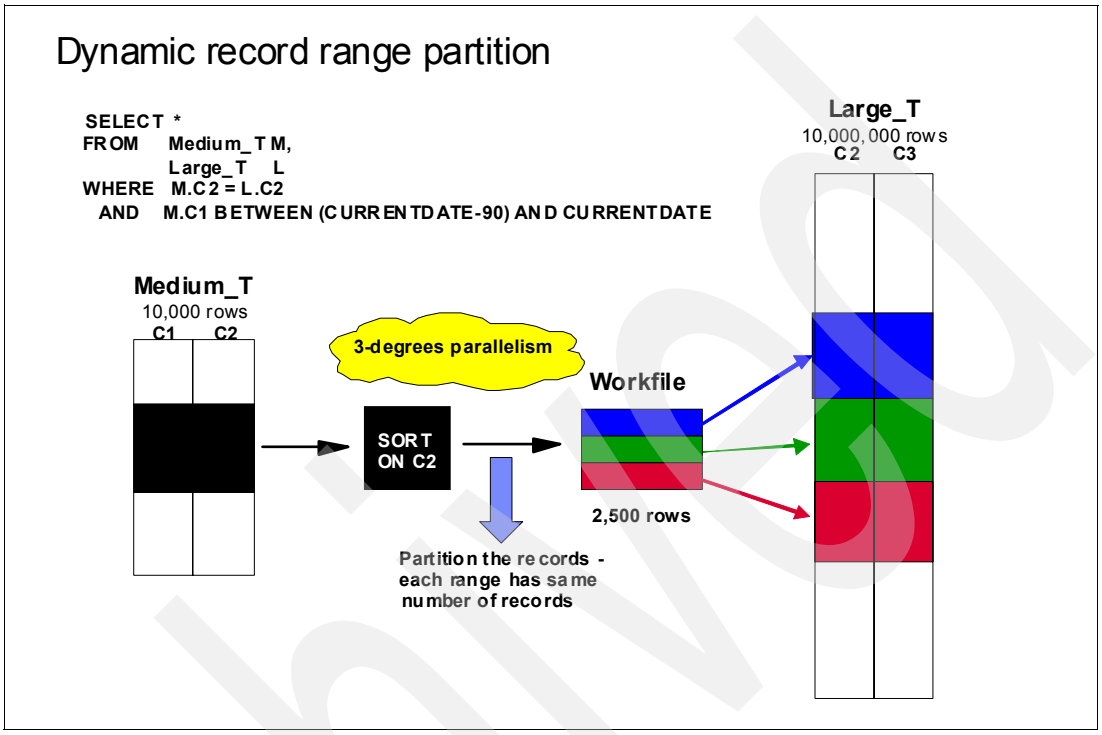


Figure 13-17 Dynamic record based partitioning

Partitioning now takes place on the work file after it is sorted. Each parallel task now has the same number of rows to process.

This division does not have to be on the key boundary unless it is required for group by or distinct function. Record range partitioning is dynamic because partitioning is no longer based on the key ranges decided at bind time. Instead, it is based on the number of composite side records and the number of workload elements. All the problems associated with key partitioning, such as the limited number of distinct values, lack of statistics, data skew or data correlation, are bypassed and the composite side records are distributed evenly.

13.21.3 Straw model for workload distribution

Prior to DB2 10, DB2 divides the number of keys or pages to be processed by the number representing the parallel degree. One task is allocated per degree of parallelism, the range is processed and the task ends. So, parallelism tasks can take different times to complete.

DB2 might, therefore, have bad workload balancing issues as depicted by Figure 13-16 on page 594, caused by lack of statistics or data skew. Even with dynamic record based partitioning on the leading table, a workload can go out-of-balance through repeated nested joins.

DB2 10 introduces a new *straw model* workload distribution method, as depicted by Figure 13-18. More key or page ranges are allocated than the number of parallel degrees. The same number of parallel processing tasks are allocated as before, (the same as the

degree of parallelism). However, after a task finishes its smaller range, it does not terminate, but it processes another range until all ranges are processed. Even if data is skewed, this process should make processing faster, because all parallel processing tasks finish at about the same time.

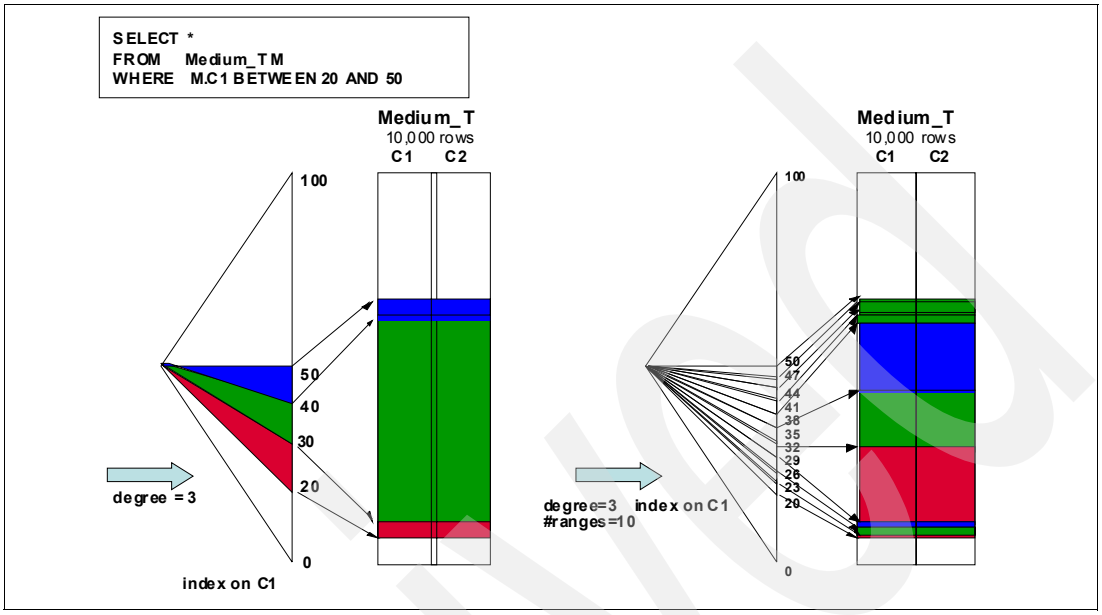


Figure 13-18 Workload balancing straw model

The degree of parallelism for both examples is 3. In the old method (on the left), the key ranges are divided into three tasks, but the middle task takes the longest time to process because it has far more rows to process. In the straw model (on the right), the key ranges are split into 10, even though the degree of parallelism is still 3. The three tasks are allocated smaller key ranges and when each one task finishes, it processes another key range. In the old method, the second task processed most of the rows, but in the straw model all three tasks share the work, making the query run quicker.

You can see whether the straw model is used by viewing the DSN_PGROUP_TABLE EXPLAIN table. Table 13-8 shows the new column.

Table 13-8 DSN_PGROUP_TABLE changes

Column name	Data type	Description
DSN_PGROUP_TABLE. STRAW_MODEL	CHAR(1) NOT NULL WITH DEFAULT '	► Y: The parallel group is processed in straw model. ► N: The parallel group is not processed in straw model (default value).

The straw model is a new concept for providing better workload distribution in SQL parallelism. An IFCID 363 is introduced to monitor the use of the straw model in parallelism. IFCID 363 is started through Performance Trace class 8. The contents of IFCID 363 are listed in Example A-11 on page 623,

13.21.4 Sort merge join improvements

Prior to DB2 10, DB2 uses the same key range to partition both the outer table and the inner table for sort merge join (SMJ). However, in some cases, the inner work file cannot be partitioned. See “Support for parallelism parallel group contains a work file” on page 593 for a more detailed discussion. DB2 10 takes advantage of building a sparse index for the inner work file by using the index to position on the starting point for each child parallel task before starting normal SMJ processing. Even if parallelism is disabled at run time, DB2 still uses the sparse index to access the work file.

Table 13-9 shows the two new columns in the PLAN_TABLE.

Table 13-9 Sort merge join PLAN_TABLE changes

Column Name	Data type	Description
PLAN_TABLE.MERGEC	CHAR(1) NOT NULL WITH DEFAULT	► Y: The composite table is consolidated before join. ► N: The composite table is not consolidated before join (default value).
PLAN_TABLE.MERGEN	CHAR(1) NOT NULL WITH DEFAULT	► Y: The new table is consolidated before join. ► N: The new table is not consolidated before join (default value).

13.22 Online performance buffers in 64-bit common

31-bit ECSA has been a precious resource in versions prior to DB2 10 and member consolidation in DB2 10 is likely to continue to grow demand for 31-bit ECSA. In DB2 10, IFC is a significant exploiter of 64-bit common.

DB2 online performance buffers moved to 64-bit common. The buffers now support 64 MB maximum sizes. Thus the -START TRACE command keyword BUFSIZE now allows a max value of 67108864. Any value larger than that is interpreted as 67108864. The minimum and default sizes is also increased to 1 MB. The values originally were 16 MB and 256 KB respectively.

Minimum requirements: Verify that you have a minimum of 128 GB of SHARED storage and 6 GB of 64-bit COMMON storage for each DB2 subsystem that you plan to start on each MVS image.

We also suggest you define an additional 64-bit common to accommodate other potential exploiters. If a monitor application is being used that issues group scope IFI requests, up to 128 MB times the number of members in the group of data can be returned to the IFI requesting member. That LPAR should be configured to accommodate this potential spike in real storage required to manage these requests.

If the shared virtual space remaining is less than 128 GB or if there is less than 6 GB of 64-bit COMMON storage available for this DB2, DB2 abends with reason code 00E8005A and issues the following message:

```
DSNY011I csect_name z/OS DOES NOT HAVE SUFFICIENT 64-BIT SHARED AND
COMMON STORAGE AVAILABLE
```

13.23 Enhanced instrumentation

DB2 instrumentation also includes the following enhancements in DB2 10:

- One minute statistics trace interval

  Statistics trace records are generated to SMF every minute, regardless of what you specify in DSNZPARM.
- IFCID 359 for index leaf page split

  A new IFCID to help you to monitor index leaf page splits.
- Separate DB2 latch and transaction lock in Accounting class 8

  A new monitor class to monitor SQL statements at the system-wide level rather that the thread-level.
- Separate DB2 latch and transaction lock in Accounting class 8

  DB2 10 externalizes both types of waits in two different fields.
- Storage statistics for DIST address space

  You can now monitor storage used by the DIST address space
- Accounting: zIIP SECP values

  zAAP on zIIP and zIIP SECP values
- Package accounting information with rollup

  Package accounting statistics are also rolled up.
- DRDA remote location statistics detail

  MVS function can be invoked o compress DB2 SMF records.
- DRDA remote location statistics detail

  More granularity in monitoring DDF locations.

13.23.1 One minute statistics trace interval

As processors are getting faster and faster, more things can happen in a shorter time. For example, on a Uni-processor z10, we have the equivalent of ~60 billion instructions occurring in a single minute. That is a lot of instructions that can happen in a minute. The current default of 5 minutes is too long to see trends occurring in the system.

To help to capture events that happen inside DB2 for performance or problem diagnosis, DB2 10 always generates an SMF type 101 trace record (statistics trace) every one minute interval, no matter what you specify in the STATIME DSNZPARM parameter. This setting applies to selective statistics records critical for performance problem diagnosis.

This trace interval should not severely impact SMF data volumes, because only 1359 records are produced per DB2 subsystem per day.

13.23.2 IFCID 359 for index leaf page split

Index leaf page splits can cause performance problems, especially in a data sharing environment. So, DB2 10 introduces a new trace record, IFCID 359, to help you to monitor leaf page splits. IFCID 359 records when index page splits occur and on what indexes. IFCID 359 is discussed in more detail in 5.8, “IFCID 359 for index split” on page 119.

Archived

13.23.3 Separate DB2 latch and transaction lock in Accounting class 8

Prior to DB2 10, plan and package accounting IFCID data does not differentiate between the time threads wait for locks and the time threads wait for latches. Accounting records produce a single field with the total for both types of wait time reported.

DB2 10 externalizes both types of waits in two different fields in all relevant IFCID records, IFCID 3, IFCID 239, and IFCID 316.

OMEGAMON PE V5.1 externalizes both counters in both the Accounting Detail report and Accounting Trace, as shown in Figure 13-19.

Report:

PACKAGE	AVERAGE TIME	AVG.EV	TIME/EVENT
LOCK/LATCH	5.954479	23.6K	N/C
IRLM LOCK+LATCH	5.000000	23.0K	
DB2 LATCH	0.954479	0.6K	
SYNCHRONOUS I/O	1:03:11.9093	557.7K	0.033630
OTHER READ I/O	58:53.212253	260.8K	N/C
OTHER WRITE I/O	0.000000	0.00	N/C
SERV.TASK SWITCH	1.148303	6.00	0.887426
ARCH.LOG(QUIESCE)	0.000000	0.00	N/C
ARCHIVE LOG READ	0.000000	0.00	N/C
DRAIN LOCK	0.000000	0.00	N/C
CLAIM RELEASE	0.000000	0.00	N/C
PAGE LATCH	0.000000	0.00	N/C
NOTIFY MESSAGES	0.000000	0.00	N/C
GLOBAL CONTENTION	0.000000	0.00	N/C
TCP/IP LOB	0.000000	0.00	N/C
TOTAL CL8 SUSPENS.	2:02:12.2243	842.1K	0.318229

Trace:

DSNTEP2	TIME	EVENTS	TIME/EVENT
LOCK/LATCH	0.000000	0	N/C
IRLM LOCK+LATCH	0.000000	0	N/C
DB2 LATCH	0.000000	0	N/C
SYNCHRONOUS I/O	0.000000	0	N/C
OTHER READ I/O	0.000000	0	N/C
OTHER WRITE I/O	0.000000	0	N/C
SERV.TASK SWITCH	0.000191	2	0.000095
ARCH.LOG(QUIESCE)	0.000000	0	N/C
ARCHIVE LOG READ	0.000000	0	N/C
DRAIN LOCK	0.000000	0	N/C
CLAIM RELEASE	0.000000	0	N/C
PAGE LATCH	0.000000	0	N/C
NOTIFY MESSAGES	0.000000	0	N/C
GLOBAL CONTENTION	0.000000	0	N/C
TCP/IP LOB	0.000000	0	N/C
TOTAL CL8 SUSPENS.	0.000191	2	0.000095

Figure 13-19 Accounting suspend times

Prior to DB2 10, field IRLM LOCK+LATCH shows the accumulated elapsed time spent by the package or DBRM waiting for lock and latch suspensions. DB2 10 breaks this counter down. Field IRLM LOCK+LATCH shows the accumulated elapsed time waiting in IRLM for locks and latches. Field DB2 LATCH now records the accumulated elapsed time waiting for latches in DB2.

13.23.4 Storage statistics for DIST address space

DB2 Version 7 introduced two IFCIDs to help manage and monitor the virtual storage usage in the DBM1 address space. IFCID 225 provides summary storage information, and IFCID 217 provides more detailed information. Both IFCIDs have become key components for performance analysis and system tuning.

DB2 10 restructures IFCID 225 to take into account the DB2 10 memory mapping and 64-bit addressing. The trace record is also now divided into data sections for a more logical grouping of data. IFCID 225 also contains information about storage in the DIST address space.

IFCID 217 is also overhauled. Duplicate data with IFCID 225 is now removed. Enable both IFCID 225 and IFCID 217 to generate a detail system storage profile. However, in most cases IFCID 225 is sufficient for general monitoring and reporting.

Example A-5 on page 618 lists the restructured IFCID 225.

OMEGAMON PE V5.1externalizes the new DIST address space storage statistics in two new trace blocks in the Statistics Short and Long reports, as shown in Figure 13-20 and Figure 13-21.

```
DIST STORAGE ABOVE 2 GB
-------------------------------------------
GETMAINED STORAGE                        (MB)
  FIXED STORAGE                          (MB)
  VARIABLE STORAGE                       (MB)
  STORAGE MANAGER CONTROL BLOCKS         (MB)
```

Figure 13-20 Statistics, DIST storage above 2 GB

```
DIST AND MVS STORAGE BELOW 2 GB
-------------------------------------------
TOTAL DIST STORAGE BELOW 2 GB            (MB)
 TOTAL GETMAINED STORAGE                 (MB)
 TOTAL VARIABLE STORAGE                  (MB)
   NUMBER OF ACTIVE CONNECTIONS
   NUMBER OF INACTIVE CONNECTIONS
 TOTAL FIXED STORAGE                     (MB)
 TOTAL GETMAINED STACK STORAGE           (MB)
   TOTAL STACK STORAGE IN USE            (MB)
STORAGE CUSHION                          (MB)

24 BIT LOW PRIVATE                       (MB)
24 BIT HIGH PRIVATE                      (MB)
24 BIT PRIVATE CURRENT HIGH ADDRESS
31 BIT EXTENDED LOW PRIVATE              (MB)
31 BIT EXTENDED HIGH PRIVATE             (MB)
31 BIT PRIVATE CURRENT HIGH ADDRESS
EXTENDED REGION SIZE (MAX)               (MB)
```

Figure 13-21 Statistics, DIST storage below 2 GB

13.23.5 Accounting: zIIP SECP values

z/OS V1.11 is enhanced with a new function that can enable System z Application Assist Processor (zAAP) eligible workloads to run on System z Integrated Information Processors (zIIPs). This function can enable you to run zIIP- and zAAP-eligible workloads on the zIIP. This capability is ideal for customers without enough zAAP or zIIP-eligible workload to justify a specialty engine today. The combined eligible workloads can make the acquisition of a zIIP cost effective. This capability is also intended to provide more value for customers having only zIIP processors by making Java and XML-based workloads eligible to run on existing zIIPs.

This capability is available with z/OS V1.11 (and z/OS V1.9 and V1.10 with PTF for APAR OA27495) and all System z9® and System z10 servers. This capability does not provide an overflow so additional zAAP eligible workload can run on the zIIP, it enables the zAAP eligible work to run on zIIP when no zAAP is defined.

PK51045 renames the redirection eligible zIIP CPU time (IIPCP CPU TIME) to SECP CPU and the consumed zIIP CPU time (IIP CPU TIME) to SE CPU TIME, to more accurately reflect what they are, Specialty Engine times which can have both zAAP and zIIP components. These fields are externalized in the OMEGAMON PE DB2 Accounting reports.

Beginning with DB2 10, the possible redirection value SECP, which used to indicate either the estimated (PROJECTCPU) or zIIP overflow to CP if zIIP is too busy, is no longer supported and is always zero. However the actual redirected CPU time continues to be available in the

SE CPU time field. The reason is that z/OS cannot provide possible redirection values for zIIP on zAAP engines to DB2, so the value cannot represent all the specialty engines as the name implies. SECP CPU is still reported for DB2 9 and earlier; however, it lists only the possible redirection for zIIP eligible processes. SE CPU TIME remains the actual CPU time consumed for both zAAP and zIIP processors.

In DB2 10, you need to review the RMF Workload Activity reports to get an indication of how much work is eligible for zIIP or zAAP processing (PROJECTCPU) or how much work overflowed to CP because zIIP or zAAP was too busy. Look at the AAPCP (for zAAP) and IIPCP (for zIIP) in the APPL% section of the RMF workload activity report Service or Reporting class section as shown in the following example:

```
---APPL %---
CP    131.62
AAPCP   0.00
IIPCP  79.00

AAP     0.00
IIP     0.00
```

13.23.6 Package accounting information with rollup

DB2 currently provides the facility to accumulate accounting trace data. A *rollup record* is written with accumulated counter data for the following type of threads:

- Query parallelism child tasks if the DSNZPARM PTASKROL parameter is set to YES
- DDF and RRSAF threads if the DSNZPARM ACCUMACC parameter is greater than or equal to 2

For query parallelism child tasks, a rollup record is written with accumulated counter data when the parent task (agent) deallocates on an originating DB2 or when an accumulating child task deallocates on an assisting DB2. The rollup data is an accumulation of all the counters for that field for each child task that completed and deallocated.

For DDF and RRSAF threads, a rollup record is written with accumulated counter data for a given user when the number of occurrences for that user reaches the DSNZPARM value for ACCUMACC. The user is the concatenation of the following values, as defined by the DSNZPARM ACCUMUID parameter:

- User user ID (QWHEUID, 16 bytes)
- User transaction name (QWHCEUTX, 32 bytes)
- User workstation name (QWHCEUWN, 18 bytes)

Accounting statistics are rolled up only at the plan level (Accounting class 1, 2, and 3) in releases prior to DB2 10.

DB2 10 enhances accounting rollup for the package level, (accounting class 7, 8 and 10 if available).

This change impacts IFCID 003, IFCID 239, IFCID 147, IFCID 148, and SMF type 100 and 101 records (subtypes in header).

13.23.7 DRDA remote location statistics detail

Prior versions of DB2 collect statistics for all locations accessed by DRDA and group them under one collection name, DRDA REMOTE LOCS, and these statistics are reported in the

Statistics Detail reports in one single block, DRDA REMOTE LOCATIONS. DB2 10 introduces a number of enhancements to IFI tracing to address these challenges.

Because many thousands of remote clients can potentially be in communication with a DB2 subsystem, DB2 10 introduces a statistics class 7 trace to externalize DRDA statistics data by location. The new statistics class 7 trace triggers a new IFCID, 365, to be activated.

When a statistics trace is started with class 7 specified in the class list, the location data is only written out at the interval specified by the STATIME DSNZPARM parameter. During normal DB2 statistics trace processing (CLASS 1, 2, 4, 5, or 6), only the statistics for the location named DRDA REMOTE LOCS are externalized to the specified statistics destination, which defaults to SMF. The information is written to the specified destination every minute.

To get the new detail location statistics, you must either specify CLASS(7) or IFCID(365) on a -START TRACE or -MODIFY TRACE command, which activates a new or modifies an existing statistics trace. DB2 then writes new IFCID 365 records to the specified destination of the statistics trace for the remote locations that are communicating with the subsystem.

The new record includes only the statistics for those locations with activity since a record was generated. The statistics for up to 95 locations are written in one record with more than one record being written until all locations are retrieved. The default DRDA location, DRDA REMOTE LOCS, which has all location statistics, is still written every time the default statistics trace classes are written, but it is not written with the other detail location statistics in the IFCID 365 trace.

When a monitor trace is started for IFCID 365, multiple IFCID 365 records are returned in response to a READS request. The issuer of the READS must provide a buffer of suitable size. The number of IFCID 365 records written to the buffer depends on the size of the buffer passed as a parameter of the READS. The buffer must be large enough to hold at least one QW0365 location detail section and the QW0365HE header section. Again, only the locations that have activity since the last READS were processed are returned.

DDF continues to capture statistics about all DRDA locations in one location named DRDA REMOTE LOCS. The statistics for this default DRDA location is the only location statistics written with the default statistics trace data, which is a change from DB2 9 where the statistics for up to 32 locations were written.

DDF now captures statistics about each location that it is in communication with as a server or a requester. The statistics for these locations is written every time the STATIME interval has elapsed. The statistics detail for up 75 locations is written in one IFCID 365 record. Multiple records are produced until the detail statistics for all locations showing activity are written.

DDF continues to capture statistics at the accounting level for every location referenced by the thread during a single accounting interval or a rollup of multiple accounting intervals. The information is written every time an accounting record is requested to be sent to disk.

IFCID 365 trace data can be accessed through the monitor interface by starting a user defined class monitor trace with IFCID 365 specified. The location statistics are returned in response to a READS request.

Note: The STATIME subsystem parameter applies only to IFCIDs 0105, 0106, 0199, and 0365.

In addition, some location statistics counters collected are private protocol only and need to be removed from the statistics that are captured. DB2 10 also restructures trace records to expand DDF counters and remove redundant private protocol counters. The layout for the

DRDA remote locations reporting has not changed in OMEGAMON PE statistics reports. Obsolete fields are shown as N/A.

13.24 Enhanced monitoring support

Although DB2 for z/OS has support for problem determination and performance monitoring at the statement level for both dynamic and static SQL, primarily through tracing, the following limitations apply:

- Identifying the specific SQL statements involved in a problem (for example deadlocks, timeouts and unavailable resource conditions) can be expensive and costly. In addition, the process to narrow the application in question can be time-consuming and can involve resources from many different teams including (DBA, application developers, system administrators, and users). You might need to examine traces after the original problem occurs because insufficient information was captured at the point where the problem first occurred. Trace and log files can be huge and can change the timing behavior of applications.
- For distributed workloads, low priority or poorly behaving client applications can monopolize DB2 resources and prevent high-priority applications from executing. There is no way to prioritize traffic before it arrives into DB2. WLM classification of workload does not help because it does not occur until after the connection has been accepted and a DBAT is associated to the connection. WLM makes sure that high priority work completes more quickly, but if there is a limited number of DBATs or connections, low priority work has equal access to the threads or connections and can impact the higher priority work. (In a data sharing environment, you can use LOCATION ALIAS NAMES to connect to a subset of members but this has limited granularity.)

For distributed workloads, the number of threads, the number of connections and idle thread timeout are controlled by system level parameters (MAXDBAT, CONDBAT, and IDTHTOIN), so all distributed clients are treated equally. This situation is a problem if the client applications do not represent equal importance to the business.

DB2 10 for z/OS enhances performance monitoring support and monitoring support for problem determination for both static and dynamic SQL. This new support uses the IFI to capture and externalize monitoring information for consumption by tooling.

To support problem determination, the statement type (dynamic or static) and statement execution identifier (STMT_ID) are externalized in several existing messages (including those related to deadlock, timeout, and lock escalation). In these messages, the statement type and statement identifier are associated with thread information (THREAD-INFO) that can be used to correlate the statement execution on the server with the client application on whose behalf the server is executing the statement.

To support performance monitoring, several existing trace records that deal with statement level information are modified to capture the statement type, statement identifier, and new statement-level performance metrics. New trace records are introduced to provide access to performance monitoring statistics in real time and to allow tooling to retrieve monitoring data without requiring disk access. In addition, profile monitoring is introduced to increase granularity and to improve threshold monitoring of system level activities. The monitor profile supports monitoring of idle thread timeout, the number of threads, and the number of connections, as well as the ability to filter by role and client product-specific identifier.

13.24.1 Unique statement identifier

DB2 10 introduces a unique statement execution identifier (STMT_ID) to facilitate monitoring and tracing at the statement level. The statement ID is defined at the DB2 for z/OS server, returned to the DRDA application requester, and captured in IFCID records for both static and dynamic SQL.

For dynamic statement, the statement identifier is available when dynamic statement cache is enabled. For static statement, the statement identifier matches the STMT_ID column value in the SYSIBM.SYSPACKSTMT table. STMT_ID column is a new column in the SYSIBM.SYSPACKSTMT table.

To use the enhanced monitoring support functions, you must rebind or bind any existing pre-DB2 10 package in DB2 10 new function mode in order for the STMT_ID column to be populated and loaded into the package.

Through DRDA, the statement identifier is returned to the application requesters in addition to a 2-byte header information which includes the SQL type information (indicating dynamic or static statement), and a 10-byte compilation time in a representation of a timestamp format with the form of *yyyymmddhhmmssnnnnnn* (the last bind time for static SQL, and the time of compilation for dynamic SQL). This information is contained in DRDA MONITORID object and is returned through the DRDA MONITORRD reply. When DRDA MONITORID is returned in the DRDA MONITORRD reply data, DB2 for z/OS server also returns DRDA SRVNAM object along in the DRDA MONITORRD reply data providing the specific server.

The following IFCIDs are changed to externalize the new statement identifier:

- IFCID 58 (End SQL) is enhanced to return statement type, statement execution identifier and statement level performance metrics. For related statements such as OPEN, FETCH, and CLOSE, only the IFCID 58 of the CLOSE request provides information which reflect the statistics collected on the OPEN and FETCHes as well.
- IFCID 63 and 350 (SQL Statement) are enhanced to return statement type, statement execution identifier, and the original source CCSID of the SQL statement. This new information is returned for the statement types that are candidates to be in the DB2 dynamic statement cache (SELECT, INSERT, UPDATE, DELETE, MERGE, etc) when the dynamic statement cache is enabled.
- IFCID 124 (SQL Statement Record) is enhanced to return the statement type and statement execution identifier.
- IFCID 66 (Close Cursor) is enhanced to trace implicit close statement. A new field is added to indicate if the close statement is an explicit close statement or an implicit close statement.
- IFCID 65 (Open Cursor) is enhanced to trace the cursor attribute on the implicit commit request. A new field is added to indicate if implicit commit cursor attribute is specified.
- IFCID 172 and IFCID 196 (Deadlock/Timeout) are enhanced to return statement type and statement execution identifier for both dynamic and static statement when a deadlock or timeout condition is detected.
- IFCID 337 (Lock Escalation) is enhanced to return statement type and statement execution identifier for both dynamic and static statement when lock escalation condition is detected.

In addition, the THREAD-INFO token is extended to include statement type information, besides role and statement ID information. The affected messages are DSNL030I, DSNV436I, DSNL027I, DSNI031I, DSNT318I, DSNT375I, DSNT376I, DSNT377I, DSNT378I, DSNT771I, and DSNT772I.

13.24.2 New monitor class 29 for statement detail level monitoring

Monitor class 9 tracing provides real-time statement level detail per thread basis, with IFCID 124. Monitor class 29 is introduced to monitor detailed trace information in real-time for all the dynamic statements in the dynamic statement cache and all static statements currently in the EDM Pool. Monitor class 29 looks at the system-wide level rather that the thread-level.

Monitor class 29 has the following IFCIDs:

- IFCID 318 is an existing IFCID that is merely a flag to instruct DB2 to collect more detailed statistics about statements in the dynamic statement cache.
- IFCID 316 is an existing IFCID that provides detailed information about the dynamic statement cache when promoted by an IFI READS request. IFCID is also written when a statement is evicted from the dynamic statement cache.
- IFCID 400 is a new IFCID that essentially mirrors the behavior of IFCID 318.
- IFCID 401 is a new IFCID that when prompted externalizes all static statements in the EDM pool together with their detailed statistics. IFCID is also written when a statement is removed from the EDM Pool. STMTID support (as well as 401 generation) requires REBIND in NFM.

  The statement identifiers and new IFCID 401 are utilized in the new *Extended Insight* feature of OMEGAMON PE V5.1 within the Optim Performance Manager infrastructure.

13.24.3 System level monitoring

DB2 9 for z/OS introduced profiles to allow you to identify a query or set of queries. It is possible to identify SQL statements by authorization ID and IP address, or to specify combinations of plan name, collection ID, and package name. These profiles are specified in the table SYSIBM.DSN_PROFILE_TABLE. Profile tables allow you to record information about how DB2 executes or monitors a group of statements. SQL statements identified by a profile are executed based on keywords and attributes in the table SYSIBM.DSN_PROFILE_ATTRIBUTES. System parameters such as NPGTHRSH, STARJOIN and SJTABLES can be specified as keywords, providing greater granularity than DSNZPARM parameters.

In DB2 9 for z/OS it is possible to identify SQL statements by authorization ID and IP address, or to specify combinations of plan name, collection ID, and package name. All statements so identified can be monitored. The following tables are required to monitor statements:

- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY
- SYSIBM.DSN_STATEMENT_RUNTIME_INFO

The following tables might also be required, depending on the information that DB2 is to record:

- SYSIBM.DSN_OBJECT_RUNTIME_INFO
- SYSIBM.PLAN_TABLE
- SYSIBM.DSN_STATEMNT_TABLE
- SYSIBM.DSN_FUNCTION_TABLE
- Other EXPLAIN tables that are used by optimization tools

Statement execution is controlled by the following keywords:

- NPAGES THRESHOLD

- STAR JOIN
- MIN STAR JOIN TABLES

Other keywords control the amount of monitoring data recorded.

The information collected in these table was used by tools such as the Optimization Service Center and Optimization Expert. See *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421.

DB2 10 enhances support for filtering and threshold monitoring for system related activities, such as the number of threads, the number of connections, and the period of time that a thread can stay idle.

Two new scope filters on role (available through trusted context support) and client product-specific identifier are added to provide more flexibility to monitor the activities across the DB2 system. Allowing filtering by role and client product-specific identifier gives a finer degree of control over the monitor profiles.

Similarly, the addition of new function keywords related to the number of threads, the number of connections and idle thread timeout values allows thresholds (limits) that were previously available only at the system level through DSNZPARM to be enforced at a more granular level.

Together, these enhancements provide a greater flexibility and control with regard to allocating resources to particular clients, applications or users according to their priorities or needs.

The enhancements in DB2 10 apply only to the system level monitoring related to threads and connections, not to the statement level monitoring and tuning.

Briefly, to use system level monitoring, you must first create the following tables, if they do not already exist, by running either installation job DSNTIJSG or DSNTIJOS:

- SYSIBM.DSN_PROFILE_TABLE

  Has one row per monitoring or execution profile. A row can apply to either statement monitoring or system level monitoring but not both. Multiple profile rows can apply to an individual execution or process, in which case the more restrictive profile is applied.
- SYSIBM.DSN_PROFILE_HISTORY

  Tracks the state of rows in the profile table or why a row was rejected.
- SYSIBM.DSN_PROFILE_ATTRIBUTES

  Relates profile table rows to keywords, which specify monitoring or execution attributes which define how to direct or define the monitoring or execution.
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY

  Tracks the state of rows in the attributes table or why a row was rejected.

To monitor system level activities, you must then:

1. Create a profile to specify what activities to monitor.
2. Define the type of monitoring. (There are three keywords and two attribute columns.)
3. Load or reload profile tables and start the profile.
4. Stop Monitoring.

To create a profile, add a row to the table SYSIBM.DSN_PROFILE_TABLE. DB2 10 provides the ROLE and PRDID columns for additional filtering. ROLE filters by the role that is assigned

to the user who is associated with the thread. PRDID filters by the client product-specific identifier that is currently associated with the thread, for example JCC03570. In addition, different profiles can apply to different members of a data sharing group.

The following filtering categories are supported for system level monitoring:

- IP Address (IPADDR)
- Product Identifier (PRDID)
- Role and Authorization Identifier (ROLE, AUTHID)
- Collection ID and Package Name (COLLID, PKGNAME)

The filtering criteria in different filtering categories cannot be specified together. For example, IP Address and Product ID cannot be specified together as a valid filtering criteria because IP Address and Product ID are in different filtering categories.

The newly introduced ROLE and Product ID filtering scope can also only apply to the system related monitoring functions or keywords regarding threads and connections.

The product-specific identifier of the remote requester (PRDID), is in the form *pppvvrrm*, an 8-byte field with alphanumeric characters, where:

ppp	Identifies the specific database product.
vv	Identifies the product version.
rr	Identifies the product release level.
m	Identifies the product modification level.

When there are more than one filtering scope in the same filtering category, the filtering scope can exist without the other. For example, for the role and authorization ID filtering category, role can be filtered by itself or authorization ID can be filtered by itself, or role and authorization ID can be filtered together. The same applies to collection ID and package name filtering category. That is for collection ID and package name filtering category, collection ID can be filtered by itself or package name can be filtered by itself, or collection ID and package name can be filtered together. These new filtering categories rules only apply to the system related monitoring functions or keywords regards to threads and connections.

Figure 13-22 shows profiles 11 and 15 as invalid because there are different filtering categories defined in the same row. There are corresponding rows in SYSIBM.DSN_PROFILE_HISTORY that show the reason for the rejection in the STATUS column.

ROLE	AUTHID	IPADDR	PRDID	COLLID	PKGNAME	PROFILEID	PROFILE_ENABLED
DB2DEV	Tom	Null	Null	Null	Null	10	Y
DB2DEV	Null	Null	SQL09073	Null	Null	11	Y
Null	Tom	Null	Null	Null	Null	12	Y
Null	Null	Null	JCC03570	Null	Null	13	Y
Null	Null	129.42.16.152	Null	Null	Null	14	Y
Null	Null	129.42.16.152	Null	COLL1	Null	15	Y

Figure 13-22 System level monitoring - Invalid profile

Multiple qualified filtering profiles from different filtering category can all apply if the filtering criteria matches the thread or connection for system level monitoring. For example in the Figure 13-22, profile 10 and 14 might both apply to Tom's thread or connection.

Within the same filtering category, there is only one qualified profile applied. When there is more than one profile qualified within the same filtering category, the most specific profile is chosen. In Role and Authorization ID filtering category, Role has higher precedence over Authorization. In Collection ID and Package name filtering category, Collection ID has higher precedence over Package name. In the previous example, if profile 10 and 12 both apply to Tom's thread or connection for the same keyword, row 10 will apply, as ROLE takes precedence.

You define the type of monitoring that you want to perform by inserting a row into the table SYSIBM.DSN_PROFILE_ATTRIBUTES with the following attributes:

- Profile ID column

  Specify the profile that defines the system activities that you want to monitor. Use a value from the PROFILEID column in SYSIBM.DSN_PROFILE_TABLE.

- KEYWORDS column

  Specify one of the following monitoring keywords:

MONITOR THREADS	The number of active threads for this profile. This value cannot exceed MAXDBAT
MONITOR CONNECTIONS	The total number of connections, active plus inactive, for this profile. This value cannot exceed CONDBAT
MONITOR IDLE THREADS	The number of seconds before idle threads timeout, for this profile. This value can exceed IDTHTOIN

- ATTRIBUTE1, ATTRIBUTE2, and ATTRIBUTE3 columns

  Specify the appropriate attribute values depending on the keyword that you specify in the KEYWORDS column.

  There are two levels of messaging mechanisms introduced (DIAGLEVEL1 and DIAGLEVEL2) as described later for system profile monitoring related to threads and connections. You can choose which messaging level you prefer by defining a value in the ATTRIBUTE1 column in the DSN_PROFILE_ATTRIBUTES table.

  You specify the threshold for the corresponding keyword in ATTRIBUTE2.

  Currently ATTRIBUTE3 does not apply to any of the system related monitoring functions so there is no need to enter any value for the ATTRIBUTE3 column. If there is value specified in ATTRIBUTE3 the row for system level monitoring keyword, a row is inserted into SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY table to indicate this row is invalidated.

  You do no need to specify values for any other columns for SYSIBM.DSN_PROFILE_ATTRIBUTES.

  MONITOR THREADS is used to monitor the total number of concurrent active threads on the DB2 subsystem. This monitoring function is subject to the filtering on IPADDR, PRDID, ROLD/AUTHID and COLLID/PKGNAME defined in SYSIBM.DSN_PROFILE_TABLE.

  MONITOR CONNECTIONS is used to monitor the total number of remote connections from the remote requesters using TCP/IP, which includes the current active connections and the inactive connections. This monitoring function is subject to the filtering on the IPADDR column only in the SYSIBM.DSN_PROFILE_TABLE for remote connections. Active connections are those currently associated with an active DBAT or have been

queued and are waiting to be serviced. Inactive connections are those currently not waiting and not associated with a DBAT.

MONITOR IDLE THREADS is used to monitor the approximate time (in seconds) that an active server thread should be allowed to remain idle.

It is important to note that these system level function keywords are to monitor system related activities such as the number of threads, the number of connections, and idle time of the threads, not to monitor any SQL statements activities.

ATTRIBUTE1 specifies either WARNING or EXCEPTION and the messaging level, as follows:

- WARNING (defaults to DIAGLEVEL1)
- WARNING DIAGLEVEL1
- WARNING DIAGLEVEL2
- EXCEPTION (defaults to DIAGLEVEL1)
- EXCEPTION DIAGLEVEL1
- EXCEPTION DIAGLEVEL2

ATTRIBUTE2 specifies the actual threshold value, for example the number of active threads or total connections, or seconds for idle threads

ATTRIBUTE3 must remain blank.

If a WARNING is triggered, then either message DSNT771I or DSNT772I is issued depending on the DIAGLEVEL specified and processing continues. However if an EXCEPTION is triggered, then processing varies depending on the exception triggered;

- MONITOR THREADS

  Threads are either queued or suspended as follows:

  - If IPADDR filter, then return RC00E30506 and queue the threads
  - If PRODID, ROLE or AUTHID filter, then return RC00E30507 and queue and suspend for threshold, then fail additional connection requests with SQLCODE -30041
  - If COLLID or PKGNAME filter, then return RC00E30508 and queue and suspend for threshold, then fail SQL statement and return SQLCODE -30041

- MONITOR CONNECTIONS (IPADDR filtering only)
  - Issue RC00E30504 and reject new connection requests
- MONITOR IDLE TREADS – thread terminated
  - Issue RC00E30502

Note that the system-wide installation parameters related to maximum connections (e.g. CONDBAT), maximum threads (for example MAXDBAT) and idle thread timeout (IDTHTOIN) still apply. In the event that the system-wide threshold is set lower than a monitor profile threshold, the system-wide threshold is enforced before the monitor profile threshold could apply.

When DIAGLEVEL1 is chosen, a DSNT771I console message is issued at most once every 5 minutes for any profile threshold that is exceeded. This level of messaging provides minimum console messages activity and limited information in the message text itself. You should refer to the statistics trace records to determine the accumulated occurrences of a specific profile threshold that is exceeded under a specific profile ID.

```
DSNT771I csect-name A MONITOR PROFILE condition-type CONDITION OCCURRED
numberof-time TIME(S).
```

When DIAGLEVEL2 is chosen, a DSNT772I console message is issued at most once every 5 minutes for each unique occurrence of the profile threshold in a specific Profile ID that is

exceeded. This level of messaging provides more information in the message text itself which includes the specific profile ID and the specific reason code. You can also refer to the statistics trace records to determine the accumulated occurrences of a specific profile threshold that is exceeded under a specific profile ID.

```
DSNT772I csect-name A MONITOR PROFILE condition-type CONDITION OCCURRED
numberof-time TIME(S) IN PROFILE ID=profile-id WITH PROFILE FILTERING
SCOPE=filtering-scope WITH REASON=reason
```

For both message levels, when a profile warning or exception condition occurs, a DB2 statistics class 4 IFCID 402 trace record is written at a statistical interval which is defined in the STATIME DSNZPARM. Each statistics trace record written can contain up to 500 unique profiles. Multiple trace records can be written if there are more than 500 unique profiles whose profile thresholds are exceeded in a given STATIME interval.

Figure 13-23 show sample rows from SYSIBM.DSN_PROFILE_ATTRIBUTES. The REMARKS column is not shown.

ProfileID	Keywords	Attribute1	Attribute2	Attribute3	Attribute Timestamp
1	MONITOR THREADS	Exception_diaglevel2	10 Queue 11-20 *Reject 21st		2009-12-19...
2	MONITOR CONNEC-TIONS	Warning	50		2009-12-19…
3	MONITOR IDLE THREADS	Exception_diaglevel1	300		2009-12-21…

Figure 13-23 System level monitoring - Attributes table

The first row indicates that DB2 monitors the number of threads that satisfy the scope that is defined by PROFILEID 1 in SYSIBM.DSN_PROFILE_TABLE. When the number of the threads in the DB2 system exceeds 10 that is defined in ATTRIBUTE2 column, a DSNT772I message is generated to the system console (if there is no other DSNT772I message being issued within last 5 minutes due to this particular profile threshold in PROFILEID 1 is exceeded) and DB2 queues or suspends the number of any new connection request up to 10, the defined exception threshold in ATTRIBUTE2, as EXCEPTION_DIAGLEVEL2 is defined in the ATTREBUTE1 column.

When the total number of threads that are queued or suspended reaches 10, DB2 begins to fail the connection request with SQLCODE -30041 (up to 10 threads for this profile, PLUS up to 10 queued connections for this profile). The 21st connection for this profile receives a -30041, unless the profile filters on IPADDR, in which case any number of connections are allowed, up to MONITOR CONNECTIONS, if such a row is provided for this profile with Exception in Attribute 1 or until CONDBAT is reached in the system.

The second row indicates that DB2 monitors the number of connections that satisfy the scope that is defined by PROFILEID 2 in SYSIBM.DSN_PROFILE_TABLE. When the number of connections in the DB2 system exceeds 50 that is defined in ATTRIBUTE2 column, a

DSNT771I message is generated to the system console (if there is no other DSNT771I message being issued within last 5 minutes) and DB2 continues to service the new connection request as WARNING is defined in the ATTRIBUTE1 column.

The third row indicates that DB2 monitors the period of time that the thread can remain idle that satisfies the scope that is defined by PROFILEID 3 in SYSIBM.DSN_PROFILE_TABLE. When the thread stays idle for more than 5 minutes (300 in second) that is defined in ATTRIBUTE2 column, a DSNT771I message is generated to the system console (if there is no other DSNT771I message being issued within last 5 minutes) and DB2 terminates the thread as EXCEPTION_DIAGLEVEL1 is defined in the ATTRIBUTE1 column.

Next, you need to load or reload the profile tables into memory by issuing the following command, (the command has no parameters):

```
-START PROFILE
```

Any rows with a “Y” in the PROFILE_ENABLED column in SYSIBM.DSN_PROFILE_TABLE are now in effect. DB2 monitors any system activities related to threads and connections that meet the specified criteria. When threshold is reached, DB2 takes certain action according to the value specified in the ATTRIBUTE1 column, EXCEPTION or WARNING. Refer to the section of the changes to SYSIBM.DSN_PROFILE_ATTRIBUTES for specific action that DB2 is taking. It is important to note that when you start the START PROFILE command, DB2 begins to monitor the next thread or the next connection activities for the monitoring functions that are in effect.

When you finish monitoring these system related threads and connections activities, stop the monitoring process by performing one of the following tasks:

- To disable the monitoring function for a specific profile Delete that row from SYSIBM.DSN_PROFILE_TABLE, or change the PROFILE_ENABLED column value to N. Then, reload the profile table by issuing the command START PROFILE.
- To disable all monitoring for both system related and statement related monitoring that are specified in the profile tables, issue the following command:

  ```
  - STOP PROFILE
  ```

DB2 provides the following commands to manage performance profiles:

- START PROFILE

  To load or reload active profiles into memory. This command can be issued from the z/OS console through a batch job or IFI. This command can also be used by tools, such as Optim Query Tuner.
- DISPLAY PROFILE

  Allows you to see if profiling is active or inactive. This command can be issued from the z/OS console, using a batch job or instrumentation facility interface (IFI). This command can also be used by tools, such as Optim Query Tuner.
- STOP PROFILE

  To stop or disable the profile monitoring function. This command can be issued from the z/OS console, using a batch job or IFI. This command can also be used by tools, such as Optim Query Tuner.

By using profiles, you can easily monitor system level activities. Profiling provides granularity of control for warning and exception handling based on filtering capability for Idle threads, number of connections and number of threads. Now thread and connection resources can be monitored and controlled by filters to correspond to business priorities.

The information collected in these table was used by tools such as the Optimization Service Center and Optimization Expert. See *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421.

Archived

Part 4

Appendixes

Archived

© Copyright IBM Corp. 2010. All rights reserved.

Archived

A

Information about IFCID changes

This appendix includes the details of new or changed IFCIDs that we mention in the chapters of this book. For more information about IFCIDs, refer to “Chapter 12. Planning for DB2 10 for z/OS” of *DB2 10 for z/OS What's New?*, GC19-2985.

DB2 for z/OS has system limits, object and SQL limits, length limits for identifiers and strings, and limits for certain data type values. Restrictions exist on the use of certain names that are used by DB2. In some cases, names are reserved and cannot be used by application programs. In other cases, certain names are not recommended for use by application programs though not prevented by the database manager.

For information about limits and name restrictions, refer to “Appendix. Additional information for DB2 SQL” of *DB2 10 for z/OS SQL Reference,* SC19-2983.

You can find up-to-date mapping in the SDSNMACS data set that is delivered with DB2.

© Copyright IBM Corp. 2010. All rights reserved.

A.1 IFCID 002: Dynamic statement cache

Example A-1 lists the changes to IFCID 002 to support literal replacement in the dynamic statement cache.

Example A-1 Changes to IFCID 002 - Dynamic statement cache

```
QXSTCWLP DS    D           # of times DB2 parsed dynamic statements
*                          because CONCENTRATE STATEMENTS WITH
*                          LITERALS behavior was in effect for
*                          the prepare of the statement for
*                          the dynamic statement cache
QXSTCWLR DS    D           # of times DB2 replaced at least one
*                          literal in a dynamic statement because
*                          CONCENTRATE STATEMENTS WITH LITERALS
*                          was in effect for the prepare of
*                          the statement for dynamic statement cache
QXSTCWLM DS    D           # of times DB2 found a matching reusable
*                          copy of a dynamic statement in statement
*                          cache during prepare of a statement that
*                          had literals replaced because of
*                          CONCENTRATE STATEMENTS WITH LITERALS
QXSTCWLD DS    D           # of times DB2 created a duplicate stmt
*                          instance in the the statement cache for
*                          a dynamic statement that had literals
*                          replaced by CONCENTRATE STATEMENTS WITH
*                          LITERALS behavior and the duplicate stmt
*                          instance was needed because a cache
*                          match failed solely due to literal
*                          reusability criteria
```

A.2 IFCID 002 - Currently committed data

Example A-2 lists the changes to the Data Manager Statistics Block (DSNDQIST) of IFCID 002 to record the use of currently committed data.

Example A-2 IFCID 002 - Currently committed data

```
QISTRCCI DS    F                /* RCCI: Read Currently       */
*                               /*       Committed Insert     */
*                               /* How many rows skipped by   */
*                               /* read transaction due to    */
*                               /* uncommitted insert when    */
*                               /* using Currently Committed  */
*                               /* semantic for fetch.        */
QISTRCCD DS    F                /* RCCD: Read Currently       */
*                               /*       Committed Delete     */
*                               /* How many rows accessed by  */
*                               /* read transaction due to    */
*                               /* uncommitted delete when    */
*                               /* using Currently Committed  */
*                               /* semantic for fetch.        */
QISTRCCU DS    F                /* RCCU: Read Currently       */
*                               /*       Committed Update     */
```

```
*                                  /* How many rows accessed by  */
*                                  /* read transaction due to    */
*                                  /* uncommitted update when    */
*                                  /* using Currently Committed  */
*                                  /* semantic for fetch.        */
```

A.3 IFCID 013 and IFCID 014

Example A-3 records the start and end of hash access scan.

Example A-3 IFCID 013 and IFCID 014

```
************************************************************************
* IFC ID 0013 FOR RMID 14 RECORDS INPUT TO  HASH SCAN                  *
************************************************************************
*
QW0013   DSECT                IFCID(QWHS0013)
QW0013S1 DS    0F             SELF DEFINING SECTION 1 - QWT02R10
QW0013DB DS    XL2            DATABASE ID (DBID)
QW0013PS DS    XL2            PAGESET OBID
QW0013OB DS    XL2            RECORD OBID
         ORG   QW0013
QW0013S2 DS    0F           SELF DEFINING SECTION 2 - QWT02R20
*                       NEXT FIELDS DESCRIBE PREDICATE FOR SCAN
*                            REPEATING GROUP - MAXIMUM 10
QW0013C1 DS    XL2            FIRST COLUMN NUMBER
QW0013OP DS    CL2            OPERATOR - NE,G,GE,LE ETC
QW0013CO DS    CL1            CONNECTOR A=AND, O=OR, BLANK
QW0013TF DS    CL1            T=TRUE / F=FALSE / OR BINARY ZERO
QW0013TP DS    CL1            C=COLUMN OR V=VALUE FOLLOW
         DS    CL1            RESERVED
QW0013VA DS    CL8            FIRST EIGHT BYTES OF VALUE
         ORG   QW0013VA
QW0013C2 DS    XL2            SECOND COLUMN NUMBER
         DS    XL6            RESERVED
         SPACE 2
*
*
************************************************************************
* IFC ID 0014 FOR RMID 14 RECORDS END OF HASH SCAN                     *
************************************************************************
*
QW0014   DSECT                IFCID(QWHS0014)
QW0014RT DS    F              RETURN CODE - 0 SUCCESSFUL
QW0014RE DS    F              (S)
*    (S) = FOR SERVICEABILTY
         SPACE 2
```

Archived

A.4 IFCID 106

Example A-4 shows a new reason code in IFCID 106 to record that DB2 attempted to delete all the CF structures on restart, as directed by the new DEL_CFSTRUCTS_ON_RESTART DSNZPARM parameter.

Example A-4 Changes to IFCID 106

```
*
QWP1FLG2 DS X
QWP1CSMF EQU X'80' Compress trace records destined for SMF
QWP1DCFS EQU X'40' During restart, attempt delete of CF
* structures, including the SCA, IRLM lock
* structures and allocated group buffer
pools
```

A.5 IFCID 225

Example A-5 list the restructured IFCID 225.

Example A-5 Changes to IFCID 225

```
* ! IFCID225 summarizes system storage usage
* ! The record is divided into data sections described as follows:
* !
* ! Data Section 1: Address Space Summary (QWO225)
* !                 This data section can be a repeating group.
* !                 It will report data for DBM1 and DIST
* !                 address spaces. Use QWO225AN to identify
* !                 the address space being described.
* ! Data Section 2: Thread information (QWO2252)
* ! Data Section 3: Shared and Common Storage Summary (QWO2253)
* ! Data Section 4: Statement Cache and xPROC Detail (QWO2254)
* ! Data Section 5: Pool Details (QWO2255)
*
* Data Section 1: Address Space Summary
QWO225 DSECT
QWO225AN DS  CL4        ! Address space name (DBM1 or DIST)
QWO225RG DS  F          ! MVS extended region size
QWO225LO DS  F          ! MVS 24-bit low private
QWO225HI DS  F          ! MVS 24-bit high private
QWO225EL DS  F          ! MVS 31-bit extended low private
QWO225EH DS  F          ! MVS 31-bit extended high private
QWO225TP DS  A          ! Current high address of 24-bit private region
QWO225EP DS  A          ! Current high address of 31-bit private region
QWO225CR DS  F          ! 31-bit storage reserved for must complete
QWO225MV DS  F          ! 31-bit storage reserved for MVS
QWO225SO DS  F          ! Storage cushion warning to contract
QWO225GS DS  F          ! Total 31-bit getmained stack
QWO225SU DS  F          ! Total 31-bit stack in use
QWO225VR DS  F          ! Total 31-bit variable pool storage
QWO225FX DS  F          ! Total 31-bit fixed pool storage
QWO225GM DS  F          ! Total 31-bit getmained storage
QWO225AV DS  F          ! Amount of available 31-bit storage
```

```
         DS  F          ! Reserved
QWO225VA DS  D          ! Total 64-bit private variable pool storage
QWO225FA DS  D          ! Total 64-bit private fixed pool storage
QWO225GA DS  D          ! Total 64-bit private getmained storage
QWO225SM DS  D          ! Total 64-bit private storage for storage
*                       !   manager control structures
QWO225RL DS  D          ! Number of real 4K frames in use for
*                       !   31 and 64-bit private
QWO225AX DS  D          ! Number of 4K auxiliary slots in use
*                       !   for 31 and 64-bit private
QWO225HVPagesInReal  DS  D ! Number of real 4K frames is use for
*                          !   64-bit private (available in
*                          !   >= z/OS 1.11)
QWO225HVAuxSlots     DS  D ! Number of 4K auxiliary slots in use
*                          !   for 64-bit private (available in
*                          !   >= z/OS 1.11)
QWO225HVGPagesInReal DS  D ! High water mark for number of real 4K
*                          !   frames is use for 64-bit private
*                          !   (available in >= z/OS 1.11)
QWO225HVGAuxSlots    DS  D ! High water mark for 4K auxiliary slots
*                          !   in use for 64-bit private
*                          !   (available in >= z/OS 1.11)
*
* Data Section 2: Thread information
QWO2252 DSECT
QWO225AT DS  F          ! # of active threads
QWO225DB DS  F          ! # of active and disconnected DBATs
QWO225CE DS  F          ! # of castout engines
QWO225DW DS  F          ! # of deferred write engines
QWO225GW DS  F          ! # of GBP write engines
QWO225PF DS  F          ! # of prefetch engines
QWO225PL DS  F          ! # of P-lock/notify exit engines
*
* Data Section 3: Shared and Common Storage Summary
QWO2253 DSECT
QWO225EC  DS  F         ! MVS extended CSA size
QWO225FC  DS  F         ! Total 31-bit common fixed pool storage
QWO225VC  DS  F         ! Total 31-bit common variable pool storage
QWO225GC  DS  F         ! Total 31-bit common getmained storage
QWO225FCG DS  D         ! Total 64-bit common fixed pool storage
QWO225VCG DS  D         ! Total 64-bit common variable pool storage
QWO225GCG DS  D         ! Total 64-bit common getmained storage
QWO225SMC DS  D         ! Total 64-bit common storage for
*                       !   storage manager control structures
QWO225SV  DS  D         ! Total 64-bit shared variable pool storage
QWO225SF  DS  D         ! Total 64-bit shared fixed pool storage
QWO225SG  DS  D         ! Total 64-bit shared getmained storage
QWO225SMS DS  D         ! Total 64-bit shared storage for
*                       !   storage manager control structures
QWO225GSG_SYS DS  D     ! Total 64-bit shared system agent stack
QWO225SUG_SYS DS  D     ! Total 64-bit shared system agent stack
*                       !   in use
QWO225GSG DS  D         ! Total 64-bit shared non-system agent
*                       !   stack
QWO225SUG DS  D         ! Total 64-bit shared non-system agent
```

Archived

```
*                       !   stack in use
          DS  F         ! Reserved
QW0225SHRNMOMB    DS  F ! Number of shared memory objects
*                       !   allocated for this MVS LPAR
QW0225SHRPAGES    DS  D ! Number of 64-bit shared memory pages
*                       !   allocated for this MVS LPAR (this
*                       !   count includes hidden pages)
QW0225SHRGBYTES   DS  D ! High water mark for number of 64-bit
*                       !   shared bytes for this MVS LPAR
QW0225SHRINREAL   DS  D ! Number of 64-bit shared pages backed
*                       !   in real storage (4K pages) for this
*                       !   MVS LPAR
QW0225SHRAUXSLOTS DS  D ! Number of auxiliary slots used for
*                       !   64-bit shared storage for this
*                       !   MVS LPAR
QW0225SHRPAGEINS  DS  D ! Number of 64-bit shared pages
*                       !   paged in from auxiliary storage for
*                       !   this MVS LPAR
QW0225SHRPAGEOUTS DS  D ! Number of 64-bit shared pages
*                       !   paged out to auxiliary storage
*                       !   for this MVS LPAR
*
* Data Section 4: Statement Cache and xPROC Detail
QW02254 DSECT
QW0225SC DS  F          ! Total 31-bit xPROC storage for dynamic
*                       !   SQL used by active threads
*                       !   (31-bit DBM1 private variable pool)
QW0225LS DS  F          ! Allocated 31-bit xPROC storage for dynamic
*                       !   SQL used by active threads
QW0225SX DS  F          ! Total 31-bit xPROC storage for static
*                       !   SQL statements
*                       !   (31-bit DBM1 private variable pool)
QW0225HS DS  F          ! High water mark allocated for 31-bit xPROC
*                       !   storage for dynamic SQL used by active
*                       !   threads
QW0225LC DS  F          ! # of statements in 64-bit agent local pools
*                       !   (64-bit shared agent local variable pools)
QW0225HC DS  F          ! High water mark # of statements in 64-bit
*                       !   agent local pools at high storage time
*                       !   (64-bit shared agent local variable pools)
QW0225L2 DS  D          ! Allocated statement cache storage in 64-bit
*                       !   agent local pools
*                       !   (64-bit shared agent local variable pools)
QW0225H2 DS  D          ! High water mark for allocated statement
*                       !   cache storage in 64-bit agent local pools
*                       !   (64-bit shared agent local variable pools)
QW0225HT DS  CL8        ! Timestamp of high water mark for storage
*                       !   allocated in 64-bit agent local pools
*                       !   since the last IFCID225 was written
*                       !   (64-bit shared agent local variable pools)
QW0225S2 DS  D          ! Total 64-bit STMT CACHE BLOCKS 2G
*                       !   storage
*                       !   (64-bit shared variable pool)
QW0225F1 DS  D          ! (S)
QW0225F2 DS  D          ! (S)
```

Archived

```
*
* Data Section 5: Pool Details
QWO2255 DSECT
QWO225AL DS  F          ! Total agent local storage
*                       !   (31-bit DBM1 private variable pools)
QWO225AS DS  F          ! Total system agent storage
*                       !   (31-bit DBM1 private variable pools)
QWO225ALG DS  D         ! Total agent local storage
*                       !   (64-bit shared variable pools)
QWO225ASG DS  D         ! Total system agent storage
*                       !   (64-bit shared variable pools)
QWO225BB DS  D          ! Total buffer manager storage blocks
*                       !   (31-bit DBM1 private variable pool)
QWO225RP DS  D          ! Total RID pool storage
*                       !   (64-bit shared fixed pool)
QWO225CD DS  D          ! Total compression dictionary storage
*                       !   (64-bit DBM1 private getmained)
*
```

A.6 IFCID 267

Example A-6 shows a new reason code in IFCID 267 to record a detected restart delay and lock structure rebuild.

Example A-6 Changes to IFCID 267

```
DSNDQWO4
QWO267   DSECT              IFCID(QWHSO267)
...
QWO267RR EQU   C'R'         Rebuild started due to RESTART delay
```

A.7 IFCID 316

Example A-7 lists the changes to IFCID 316 to support literal replacement in the dynamic statement cache.

Example A-7 Changes to IFCID 316

```
QWO316LR  DS   CL1       Cache literal replacement indicator:
QWO316LRB EQU  C' '      Blank = no literal replacement was done
QWO316LRR EQU  C'R'      'R' = literals were replaced in statement
QWO316LRD EQU  C'D'      'D' = Same as 'R' but cached statement is
*                             a duplicate cache entry instance
*                             because cache match failed solely due
*                             to literal reusability criteria
```

Archived

A.8 IFCID 357

Example A-8 lists the new IFCID 357, which records the start of index I/O parallel insert processing.

Example A-8 New IFCID 357

```
*
QW0357   DSECT              IFCID(QWHS0357)
QW0357DB DS    CL2          DATA BASE ID
QW0357TB DS    CL2          TABLE OBID
QW0357PS DS    CL2          INDEX SPACE PAGE SET ID
*
```

A.9 IFCID 358

Example A-9 lists the new IFCID 358 which records the end of index I/O parallel insert processing.

Example A-9 New IFCID 358

```
*
QW0358   DSECT              IFCID(QWHS0358)
QW0358DB DS    CL2          DATA BASE ID
QW0358TB DS    CL2          TABLE OBID
QW0358PS DS    CL2          INDEX SPACE PAGE SET ID
QW0358DE DS    XL2          DEGREE OF PARALLELISM
*
```

A.10 IFCID 359

Example A-10 lists the new IFCID 359, which records index page split events.

Example A-10 New IFCID 359

```
********************************************************************
*   IFCID 0359 to record index page split
********************************************************************
*
QW0359   DSECT              IFCID(QWHS0359)
QW0359DB DS    CL2          DATA BASE ID
QW0359OB DS    CL2          INDEX PAGE SET ID
QW0359PT DS    CL2          PARTITION NUMBER
QW0359FL DS    CL1          FLAGS
         DS    CL1          RESERVED
QW0359PG DS    CL4          SPLITTING PAGE NUMBER
QW0359TS DS    CL8          TIMESTAMP AT BEGINNING OF SPLIT
QW0359TE DS    CL8          TIMESTAMP AT ENDING OF SPLIT
QW0359DP EQU   X'80'        INDEX IS GBP DEPENDENT DURING SPLIT
*
```

Archived

A.11 IFCID 360

IFCID 0360 records information about queries that are incrementally rebound because parallelism was chosen in packages that were created before DB2 10. This record is written when performance trace class 3 or 10 is on.

A.12 IFCID 363

Example A-11 shows the new IFCID 363, which monitors the use of the straw model in query parallelism.

Example A-11 New IFCID 363

```
************************************************************************
* IFCID 0363 FOR RMID 22 - PARALLEL GROUP STRAW MODEL TRACE            *
************************************************************************
QW0363    DSECT             IFCID(QWHS0363)
QW0363TS  DS   CL8          Time-stamp (consistency token)
QW0363SN  DS   F            Statement number... Same
*                            as QUERYNO in PLAN_TABLE
*                            if PLAN_TABLE exists
QW0363QN  DS   H            Query block number... Same
*                            as QBLOCKNO in PLAN_TABLE
*                            if PLAN_TABLE exists
QW0363GN  DS   H            Parallel group number
QW0363BD  DS   H            Planned (bind time) degree
QW0363RK  DS   CL1          Partition kind of the parallel
QW0363C1  EQU  C'1'         Key range
*                            Constant for QW0363RK
QW0363C2  EQU  C'2'         Page range
*                            Constant for QW0363RK
QW0363C3  EQU  C'3'         Rid
*                            Constant for QW0363RK
QW0363C4  EQU  C'4'         Record on key boundary
*                            Constant for QW0363RK
QW0363C5  EQU  C'5'         Record not on key boundary
*                            Constant for QW0363RK
QW0363OD  DS   CL1          Record order. Use only
*                            if QW0363RK = 4 or 5
QW0363CA  EQU  C'A'         Ascending order
*                            Constant for QW0363OD
QW0363CD  EQU  C'D'         Descending order
*                            Constant for QW0363OD
QW0363IW  DS   CL1          In memory workfile
QW0363IY  EQU  C'Y'         Is in-memory workfile
*                            Constant for QW0363IW
QW0363IN  EQU  C'N'         Not in-memory workfile
*                            Constant for QW0363IW
QW0363RI  DS   CL1          (S)
QW0363RY  EQU  C'Y'         (S)
QW0363RN  EQU  C'N'         (S)
QW0363RO  DS   CL1          (S)
QW0363OY  EQU  C'Y'         (S)
```

Archived

```
QWO363ON  EQU  C'N'          (S)
QWO363RV  DS   CL1           (S)
QWO363VY  EQU  C'Y'          (S)
QWO363VN  EQU  C'N'          (S)
QWO363NE  DS   XL4           Number of total input workload elements
QWO363AE  DS   XL4           Number of actual workload elements
QWO363NR  DS   XL8           Total number of input records
QWO363PD  DS   XL4           Pipe degree
QWO363PS  DS   CL10          Pipe created time
QWO363PT  DS   CL10          Pipe terminated time
QWO363EN  DS   H             Number of QWO363WE entries in QWO363E
QWO363NQ  DS   H             Number of QWO363 records that
*                             together complete this
*                             series of QWO363 records
QWO363TR  DS   H             The sequence number of the
*                             QWO363 records in a series of
*                             QWO363 records
QWO363LN_OFF DS   H          Offset from QWO363 to location
*                             name
QWO363PC_OFF DS   H          Offset from QWO363 to
*                             package collection id
QWO363PN_OFF DS   H          Offset from QWO363 to program
*                             name
QWO363LN_D   DSECT           Use if QWO363LN_OFF ,= 0
QWO363LN_LEN DS   H          Length of the following field
QWO363LN_VAR DS   OCL128     %U LOCATION NAME (RDB NAME)
*
QWO363PC_D   DSECT           Use if QWO363PC_Off,=0
QWO363PC_LEN DS   H          Length of the following field
QWO363PC_VAR DS   OCL128     %U Package collection id
*
QWO363PN_D   DSECT           Use if QWO363PN_Off,=0
QWO363PN_LEN DS   H          Length of the following field
QWO363PN_VAR DS   OCL128     %U Program name
*
QWO363E   DSECT
QWO363NM  DS   H             Total length of all QWO363WE
*                             entries + 8
QWO363RE  DS   CL6           Unused
QWO363WE  DS   OCL128        Individual workload element entry
*                             There are multiple QWO363WE
*                             entries. See field QWO363EN for
*                             the number of times that QWO363WE
*                             structure repeats
QWO363IX DS   XL4            The sequence number of workload element
QWO363PI DS   XL4            Subpipe index - task number
QWO363LP DS   XL4            Page number of low bound of
*                             logical partition. Use only
*                             if QWO363RK = 2
QWO363HP DS   XL4            Page number of high bound of
*                             logical partition. Use only
*                             if QWO363RK = 2
QWO363PB DS   CL10           Subpipe start time
QWO363PE DS   CL10           Subpipe end time
QWO363FF DS   CL4            Unused
QWO363CN DS   XL8            Counter for input #rid, #record. Use only
*                             if QWO363RK = 3, 4 or 5
```

```
QW0363NI DS   XL4            Number of rows consumed
QW0363NO DS   XL4            Number of rows produced
QW0363BI DS   XL4            (S)
QW0363EI DS   XL4            (S)
QW0363LB DS   XL32           Low key buffer data.
*                             Use only if QW0363RK = 1
QW0363HB DS   XL32           High key buffer data
*                             Use only if QW0363RK = 1
*   (S) = FOR SERVICEABILITY
```

The IFCID 363 record is made up of a set of the following mapped sections:

- QWT02PSO is mapped by DSNDQWHS.
- QWT02R1O is mapped by QW0363.
- QWT02R2O is mapped by QW0363E. In one QW0363E, QW0363WE might repeat up to 100 times, which is indicated by the field QW0363EN.

Straw model trace for one parallel group is written in a series of IFCID 363 records. Each record contains the information about 100 workload elements except the last one. The repetitions of QW0363WE in the last record is the remainder of number of actual workload elements /100.

Archived

Archived

B

Summary of relevant maintenance

With a new version of DB2 reaching general availability, the maintenance stream becomes extremely important. Feedback from early users and development of additional functions cause a flux of APARs which enrich and improve the product code.

In this appendix, we look at recent maintenance for DB2 10 for z/OS that generally relates to new functionality and critical corrections:

- DB2 APARs
- z/OS APARs
- OMEGAMON PE APARs

These lists of APARs represents a snapshot of the current maintenance at the moment of writing. As such, the list becomes incomplete or even incorrect at the time of reading. It is here to help identify areas of functional improvements. Make sure that you contact your IBM Service Representative for the most current maintenance at the time of your installation. Also check on RETAIN® for the applicability of these APARs to your environment, as well as to verify pre- and post-requisites.

We recommend that you use the Consolidated Service Test (CST) as the base for service.

© Copyright IBM Corp. 2010. All rights reserved.

B.1 DB2 APARs

In Table B-1 we present a list of APARs providing functional enhancements to DB2 10 for z/OS.

This list is not and cannot be exhaustive, check RETAIN and the DB2 Web site.

Table B-1 DB2 10 current function-related APARsPM43293

APAR #	Area	Text	PTF and notes
II10817	Virtual storage	Info APAR for storage usage fixlist	
II14219	zIIP	zIIP exploitation support use information	
II14334	LOBs	Info APAR to link together all the LOB support delivery APARs	
II14401	Migration	Info APAR to link together all the migration APARs	
II14426	XML	Info APAR to link together all the XML support delivery APARs	
II14441	Incorrout PTFs	Recommended DB2 9 SQL INCORROUT PTFs	
II14464	Migration	DB2 V8 migration/fallback info APAR to/from DB2 9 (continued from II14401)	
II4474	Migration	Prerequisites for migration to DB2 10 from DB2 V8	
II4477	Migration	Prerequisites for migration to DB2 10 from DB2 99	
PK28627	DCLGEN	Additional DCLGEN option DCLBIT to generate declared SQL variables for columns defined as FOR BIT DATA (COBOL, PL/I, C, and C++).	UK37397
PK76100	Star join (EN_PJSJ)	Enable dynamic index ANDing for star join (pair wise)	UK44120 V9
PM00068	DSMAX	Support up to 100000 open data sets in the DB2 DBM1 address space	UK58204/5 V8 and V9
PM01821	Migration	Conversion of DBRMs to packages	UK53480/1 V8 and V9
PM04968	Premigration	This APAR adds the V10 DSNTIJPM job to DB2 V8 and V9 under the name DSNTIJPA	V8 (UK56305) and V9 (UK56306)
PM13466	Pricing	Use the IFAUSAGE FBFE keyword when SMF 89 detailed data collection is enabled	UK62326/7 V8 and V9
PM13467	Pricing	Use the IFAUSAGE FBFE keyword when SMF 89 detailed data collection is enabled	UK62328/9 V8 and V9
PM13525	SQL procedures	Implicit auto regeneration for native SQL procedures	UK67267 also V9
PM13631	IX on expression	Implicit auto regeneration for index on expression	UK68476
PM17542	Open/close	Enable new z/OS 1.12 allocation interface	UK60887/8 V8 and V9
PM18196	DFSORT	Optimization of DFSORT algorithms for selecting the sort technique that makes the best use of available storage	UK62201

APAR #	Area	Text	PTF and notes
PM18557	Restart	DB2 to support z/OS R12 changes to improve high allocation requests processing	UK59887/8 V8 and V9
PM19034	DSNZPARM	CHECK_FASTREPLICATION parameter to control FASTREPLICATION keyword on DSSCOPY command of CHECK utilities.	UK63215 (also V8, V9)
PM19584	LOAD	LOAD utility performance improvement option for data that is presorted in clustering key order. See also PM35284.	UK68097 also V9
PM21277	DB2 utilities	REORG, CHECK INDEX, and REBUILD INDEX users who use DB2 Sort for z/OS	UK61213 V9 also V8
PM21747	DB2 Sort	Performance enhancements	UK60466
PM24721	BIND	BIND performance improvement when using LOBs in DB2 catalog. Also RTS fix.	UK63457
PM24723	IFCID 225	Provide real storage value with z/OS support	UK68652
PM24808	DB2 installation	Several installation changes	UK63971 also V9
PM24937	Optimizer	Various fixes for optimization hints	UK66087
PM25282	IRLM	IRLM Large Notify support. IRLM uses ECSA storage to handle response data to a NOTIFY request. Prior to DB2 10, the amount of data per member was 4 MB. In DB2 10, this limit has been increased to 64 MB. IRLM NOTIFY RESPONSE exceeding 4 MB is processed using IRLM private storage.	UK64370
PM25357	Optimizer	Getpage increase using subquery	UK63087
PM25525	REORG	PARALLEL keyword on REORG with LISTDEF PARTLEVEL (apply with PM28654/UK64589)	UK64588 also V9
PM25635	ADMIN_INFO_SQL	Corrected DDL and STATs issues (also for DSNADMSB)	UK62150
PM25648	REORG	Improving REORG concurrency on defer ALTER materialization	UK70310
PM25652	DSNACCOX	Enhancement to recommend REORG on Hash Access objects based on overflow index ratio vs. total rows	UK66610
PM25679	Optimizer	Enhancement for APREUSE/APCOMPARE	UK70233
PM26480	DDF	Availability (MODIFY DDF ALIAS...) new functions	UK63820
PM26762	DSNZPARM	FLASHCOPY_PPRC (V10 only) and REC_FASTREPLICATION for use by utilities	UK63366 (also V8, V9)
PM26781	DDF	Availability (MODIFY DDF ALIAS...) preconditiong	UK63818
PM26977	Security	Separate system privileges from system DBADM authority	UK65205
PM27073	SPT01	Inline LOB preconditioning	UK65379
PM27097	WLM	Support to start and maintain a minimum number of WLM stored procedure address spaces	UK65858 also V9

APAR #	Area	Text	PTF and notes
PM27099	LISTDEF	Empty lists change from RC8 to RC4. Important since the new LISTDEF DEFINED keyword defaults to YES	UK64424
PM27811	SPT01	Inline LOB	UK66379
PM27828	UPDATE	UTS update with small record goes to overflow unnecessarily	UK64389
PM27835	Security	DB2 now supports a TCB level ACEE for the authid used in an IMS transaction that calls DB2.	UK70647 also V9
PM27872	SMF	Decompression routine DSNTSMFD and sample JCL DSNTEJDS to call DSNTSMFD	UK64597
PM27973	Segmented TS	Better use free space for SEGMENTED pagesets including UTS PBG and PBR	UK65632
PM28100	Stored procedures	Support JCC JDBC 4.0 driver for Java stored procedures	UK65385
PM28296	Security	Support for secure audit policy trace start	UK65951
PM28458	Casting	Timestamp with timezone, add restrictions for extended implicit cast for set operators	UK63890
PM28500	System profile	Filters on client information fields	UK68364
PM28543	Security	Implicit system privileges have to be separated from system DBADM authority	UK65253
PM28796	SYSROUTINEAUTH	Inefficient access to DB2 catalog during GRANT stored procedures	UK65637
PM28925	DSNZPARM	Force deletion of CF structures at group restart	UK66376
PM29037	LOBs	Altered LOB inline length materialization via REORG SHRLEVEL CHANGE	UK70302
PM29124	CHAR incompatibility	Help with handling the release incompatible change for the CHAR(decimal) built-in function on DB2 10	UK67578
PM29900	Built-in functions	Additions	UK66476
PM29901	Built-in functions	More additions	UK66046
PM30394	Security	DBADM authorities enhancements	UK67132
PM30425	Optimizer	Optimization hints enhancements	UK67637 also V9
PM30468	zIIP	Prefetch and deferred write CPU, when running on a zIIP processor, is to be reported by WLM under the DBM1 address space, not under the MSTR	UK64423
PM30991	RECOVER	Prohibit RECOVER BACKOUT YES after mass delete on segmented or UTS	UK66327
PM31003 PM31004 PM31006 PM31009	Data sharing	DELETE data sharing member	UK65750 UK67512 UK67958 UK69286

APAR #	Area	Text	PTF and notes
PM31243	REORG	REORG FORCE to internally behave like CANCEL THREAD	OPEN
PM31313	Temporal	ALTER ADD COLUMN to propagate to History Tables	UK70215
PM31314	Temporal	TIMESTAMP WITH TIMEZONE	UK71412
PM31614	Packages	Improvement in package allocation	UK66374
PM31641	LOGAPSTG	Default changed for Fast Log Apply from 100 to 500 MB	UK66964
PM31807	IRLM	IRLM support for DB2 DSNZPARM DEL_CFSTRUCTS_ON_RESTART (PM28925) to delete IRLM lock structure on group restarts.	UK65920
PM31813	DSNZPARM	New DSNZPARM DISABLE_EDMRTS to specify whether to disable collection of real time statistics (RTS) by the DB2 Environmental Descriptor Manager (EDM). This system configuration parameter requires PM37672 to be fully enabled.	UK69055
PM33501	DSNZPARM	Add a DSNZPARM to disable implicit DBRM to package conversion during BIND PLAN with MEMBER option and automatic REBIND processing.	UK68743
PM33767	Optimizer	Various enhancements and fixes (OPTHINT, APRETAINDUP, EXPLAIN PACKAGE).	UK69377
PM33991	Migration	Several installation jobs fixes.	UK69735 also V8 and V9
PM35190	Catalog	Enable SELECT from SYSLGRNX and SYSUTILX (see also PM42331).	UK73478
PM35284	LOAD	Companion APAR to PM19584 LOAD/UNLOAD FORMAT INTERNAL and LOAD PRESORTED.	UK68098 also V9
PM37057	SSL	Additional enhancements to DB2 10 for z/OS digital certificate authentication support.	UK73180
PM36177	DSNZPARM	Pre-conditioning APAR that includes changes to support the enhancement for IRLM timeout value for DDL statements enabled by APAR PM37660.	UK69029
PM37112	REORG	Enhancement for log apply phase of REORG SHRLEVEL CHANGE when run at partition level (see also PM45810).	UK71128 Also V9
PM37300	DDF	Authorization changes when there is no private protocol (see also PM17665 and PM38417). DSN6FAC PRIVATE_PROTOCOL reinstated in V10 with new option AUTH.	UK67639 also V8 and V9
PM37622	REORG	Enablement of zIIP processor exploitation of the UNLOAD phase of the REORG TABLESPACE utility. See also PM47091 for V9.	UK71458 also V9
PM37625	DSNZPxxx module	Preconditioning support in DB2 subsystem parameter macro DSNDSPRM for APARs PM24723, PM36177, PM31813, and PM33501.	UK67634

APAR #	Area	Text	PTF and notes
PM37630	REORG	For REORG TABLESPACE SORTDATA against a non-partitioned table space containing a single table, can avoid sorting on the table obid in the clustering data sort and the clustering index keys when parallel index build is used.	UK71437 also V9
PM37647	Real storage monitoring	External enablement for APAR PM24723 (IFCID 225 REAL STORAGE STATISTICS ENHANCEMENTS	UK68659
PM37660	DSNZPARM	DDLTOX in DSN6SPRM is introduced a separate time out factor for DDL and DCL (GRANT, REVOKE, and LOCK) statements. The time out value is the product of DDLTOX and the IRLM time out value specified by DSN6SPRM.IRLMRWT.	UK69030
PM37816	DSNZPARM	Follow on to APAR PM33501, it adds DSNZPARM DISALLOW_DEFAULT_COLLID in DSN6SPRM which can prevent using DSN_DEFAULT_COLLID_plan-name on implicitly generated packages during the DB2 automatic DBRM to package conversion process.	UK69199
PM38164	Access path	During access path selection, index probing can have repeated access to SYSINDEXSPACESTATS to retrieve NLEAF. This occurs if NLEAF is NULL. Make index probing more fault tolerant when NLEAF is NULL.	UK71333
PM38417	DDF	Complete DSNZPARM related changes for PM37300 (security with private protocol removal)	UK74175 also V8 and V9
PM39342	Online compression	Build Dictionary routine for compression during INSERT was modified not to be redriven by subsequent INSERTs if the dictionary could not be built and MSGDSNU235I is issued. See also PM45651.	UK68801
PM41447	DDF security	Resolve issues with CICS or IMS related new user processing at a DB2 10 for z/OS TCP/IP requester system.	UK70483
PM42331	Catalog	Foundation for SELECT from SYSLGRNX and SYSUTILX.	UK71875
PM42528	Data sharing	Delete data sharing member. See also PM51945 and PM54873.	UK74381
PM42924	RUNSTATS	Optimize sequential prefetch requests during RUNSTATS TABLESAMPLE	UK70844
PM43292	DDF	Allow RACF protected userIDs to be PassTicket authenticated.	UK72212 also V9
PM43293	DDF	MAXCONQN and MAXCONQW new ZPARMs	UK90325
PM43597	REORG	ALTER MAXROWS to set AREO* rather than AREOR	UK71467
PM43817	Statistics	DB2 code has been modified to improve execution statistics reported by both IFCIDs 316 and 401 when many concurrent threads execute same SQL statement	UK73630
PM45650	LOB	LOB pageset support for RECOVER BACKOUT YES	UK77584

APAR #	Area	Text	PTF and notes
PM45651	Online compression	The Build Dictionary routine for compression during INSERT now periodically redrive attempts to build a compression dictionary to maximize the benefits of compression while minimizing the cost of unsuccessful attempts to build a dictionary.	UK72447
PM45810	REORG	Enhancement for log apply phase of REORG SHRLEVEL CHANGE when run at partition level. (See also PM37112)	UK71128
PM47091	REORG	Completion for PM37622	UK71459
PM47618	XML	Addition of XQuery support	UK73139
PM51467	Data sharing	Reduce high Coupling Facility F utilization, from 30% to up to 90%, after migrating to V10 because of much more Delete-Name processing (during pseudo close). See also OA38419 and PM51655.	UK75324
PM51655	Data sharing	The castout logic has been modified to not use a new restart token algorithm.	UK73864
PM51945	Data sharing	Delete member completion. See PM42528.	UK74381
PM52012	REORG	New zParm REORG_LIST_PROCESSING to control the default behavior of REORG TABLESPACE LIST partition processing	UK76650
PM52724	Mass delete	Lock escalation on SYSCOPY. Follow on to PM30991	UK80113
PM52327	DDF	Reduce CPU for excessive block scanning for each DDF call to a remote location.	UK74981
PM53237	RESTORE	RESTORE to work with Space Efficient FlashCopy	UK77490
PM53243	Stored procedures	Monitoring improvements to easily identify problematic stored procedures or a statement within that stored procedure	UK78514
PM53254	REORG	REORG to ignore free space for PBG	UK78208
PM54873	Data sharing	Delete member, more code	UK74381
PM55051	NPSI	SORTNPSI parm in REORG and ZPARM (also PM60449)	UK78229 also V9
PM56631	SPSS support	Pack/Unpack buit-in functions (also PM55928)	UK79243
PM56845	Access plan	DSNZPARM for OPTIMIZE FOR 1 ROW to allow sort access plans	UK77500
PM57206	Accounting	IBM specialty engine eligible time that runs on a general purpose CP will be reported in a serviceability field QWACZIIP_ELIGIBLE in DB2 accounting records.	UK79406
PM57632	Load	LOAD SHRLEVEL CHANGE partition parallelism with a single input dataset	UK78632
PM57878	XML	Performance improvement for XQuery constructors	UK77739
PM58177	REORG	REORG to accept a mapping table defined in a PBG table space	UK78241 also V9

APAR #	Area	Text	PTF and notes
PM62797	Accounting	New accounting aggregation (every 1 min with statistics) functionality into an IFC statistics record by connection type	UK81047
PM66287	Catalog	SYSPACKAGE LASTUSED interference with RTS or AUTOBIND	UK82732
PM67544	Data sharing	CF DELETE_NAME, performance	UK82633
PM69522	DB2 Sort	DB2 Sort 1.3 performance with DB2 utilities	UK81520 also V9
PM70046	Access plan	Re-establishes OPTIOWGT	UK83168
PM70270	Buffer pool	Lower buffer pool hit ratio or more synchronous I/O	UK82555
PM70575	Data sharing	Coupling Facility 18 cache 'write-around' to reduce impact of updates to GBP (also XES OA37550)	OPEN
PM71114	Stored procedures	New SYSPROC.ADMIN_UPDATE_SYSPARM to change the value of DSNZPARMs	UK83171 also V9
PM72526	XML	Asynchronous deletion of unneeded XML document versions is now candidate for execution on zIIP	UK91203
PM80779	Performance	ACCESS DB performance improvement (also PM91930) with parallel tasks	UK95407
PM82301	LOB	Users get down level page for lob objects defined with GBPCACHE SYSTEM. It caused storage overlay (also PM84750).	UK82633
PM86952	Storage	Contraction for multi-block memory segments now implemented for 64 bit pools	OPEN
PM94885	DDF	Follow on to PM43293	OPEN

B.2 z/OS APARs

In Table B-2 we present a list of APARs providing additional enhancements for z/OS.

This list is not and cannot be exhaustive, check RETAIN and the z/OS Web site.

Table B-2 z/OS DB2-related APARs

APAR #	Area	Text	PTF and notes
OA03148	RRS exit	RRS support.	UA07148
OA31116	1 MB page	Large frames support.	UA57254
OA33106	ECSA memory for SRB	Reduce SRB storage usage.	UA56174
OA33529	1 MB page	Large frames support.	UA57243
OA33702	1 MB page	Large frames support.	UA57704
OA34865	SMF	Remove cause for accumulation of storage use in subpool 245 (SQA/ESQA)	UA55970

APAR #	Area	Text	PTF and notes
OA35057	Media manager	Important media manager fixes.	UA58937
OA35885	RSM	RSM IARV64 macro provides an interface to obtain the real frame and auxiliary storage in use to support an input high virtual storage range.	UA60823
OA37550	Coupling facility	Full function small programming enhancement (SPE) for z/OS V1R12 (HBB7770) and V1R13 (HBB7780) to provide Coupling facility cache structure performance enhancements and RAS improvements.	UA66416
OA38419	Coupling facility	Add two new operands to control the execution flow when the DELETE_NAME command is deleting both directory entries and the associated data elements. One option controls generation or suppression of XI signals as entries are deleted. The other option controls whether or not the command should delete changed directory entries and data. See PM51467.	UA66420

B.3 OMEGAMON PE APARs

In Table B-3 we present a list of APARs providing additional enhancements for IBM Tivoli OMEGAMON XE for DB2 PE on z/OS V5.1.0, PID 5655-W37.

This list is not and cannot be exhaustive; check RETAIN and the DB2 tools website.

Table B-3 OMEGAMON PE GA and DB2 10 related APARs

APAR #	Area	Text	PTF and notes
II14438		Info APAR for known issues causing high CPU utilization.	
PM22628		Various fixes and improvements to be considered part of the GA installation package.	UK61094
PM23887		Various fixes and improvements to be considered part of the GA installation package.	UK61093
PM23888		Various fixes and improvements to be considered part of the GA installation package.	UK61142
PM23889		DB2 10 support for PLAN_TABLE for component EXPLAIN.	UK61139
PM24082		Various fixes and improvements to be considered part of the GA installation package.	UK61317
PM24083		Updates including columns to ACCOUNTING FILE DDF, GENERAL and PROGRAM.	UK65325
PM32638		Collection of new functionality and development fixes.	UK65399
PM32647		Collection of new functionality and development fixes.	UK65412
PM35049		Collection of new functionality and development fixes.	UK65924

Archived

APAR #	Area	Text	PTF and notes
PM47871		Batch Record Trace can now "FILE" IFCID 316 and 401 trace data (dynamic and static SQL statements evicted from caches, IFI READA data) into a DB2 LOAD format.	UK72590
PM67565		Support stored procedure monitoring as delivered by DB2 APAR PM53243/UK78514	UK8112
PM70645		Follow on for PM53243	UK81124

Archived

Abbreviations and acronyms

AC	Autonomic computing
ACS	Automatic class selection
AIX	Advanced Interactive eXecutive from IBM
APAR	Authorized program analysis report
API	Application programming interface
AR	Application requester
ARM	Automatic restart manager
AS	Application server
ASCII	American National Standard Code for Information Interchange
B2B	Business-to-business
BCDS	DFSMShsm backup control data set
BCRS	Business continuity recovery services
BI	Business Intelligence
BLOB	Binary large objects
BPA	Buffer pool analysis
BSDS	Boot strap data set
CBU	Capacity BackUp
CCA	Channel connection address
CCA	Client configuration assistant
CCP	Collect CPU parallel
CCSID	Coded character set identifier
CD	Compact disk
CDW	Central data warehouse
CF	Coupling facility
CFCC	Coupling facility control code
CFRM	Coupling facility resource management
CICS	Customer Information Control System
CLI	Call level interface
CLOB	Character large object
CLP	Command line processor
CM	conversion mode
CMOS	Complementary metal oxide semiconductor
CP	Central processor
CPU	Central processing unit
CRCR	Conditional restart control record
CRD	Collect report data
CRUD	Create, retrieve, update or delete
CSA	Common storage area
CSF	Integrated Cryptographic Service Facility
CTE	Common table expression
CTT	Created temporary table
CUoD	Capacity Upgrade on Demand
DAC	Discretionary access control
DASD	Direct access storage device
DB	Database
DB2	Database 2
DB2 PE	DB2 Performance Expert
DBA	Database administrator
DBAT	Database access thread
DBCLOB	Double-byte character large object
DBCS	Double-byte character set
DBD	Database descriptor
DBID	Database identifier
DBM1	Database master address space
DBRM	Database request module
DCL	Data control language
DDCS	Distributed database connection services
DDF	Distributed data facility
DDL	Data definition language
DES	Data Encryption Standard
DLL	Dynamic load library manipulation language
DML	Data manipulation language
DNS	Domain name server
DPSI	Data partitioning secondary index
DRDA	Distributed Relational Data Architecture
DSC	Dynamic statement cache, local or global
DSNZPARMs	DB2's system configuration parameters
DSS	Decision support systems
DTT	Declared temporary tables
DWDM	Dense wavelength division multiplexer

© Copyright IBM Corp. 2010. All rights reserved.

DWT	Deferred write threshold
EA	Extended addressability
EAI	Enterprise application integration
EAS	Enterprise Application Solution
EBCDIC	Extended binary coded decimal interchange code
ECS	Enhanced catalog sharing
ECSA	Extended common storage area
EDM	Environmental descriptor manager
EJB	Enterprise JavaBean
ELB	Extended long busy
ENFM	enable-new-function mode
ERP	Enterprise resource planning
ERP	Error recovery procedure
ESA	Enterprise Systems Architecture
ESP	Enterprise Solution Package
ESS	Enterprise Storage Server®
ETR	External throughput rate, an elapsed time measure, focuses on system capacity
EWLC	Entry Workload License Charges
EWLM	Enterprise Workload Manager™
FIFO	First in first out
FLA	Fast log apply
FTD	Functional track directory
FTP	File Transfer Program
GB	Gigabyte (1,073,741,824 bytes)
GBP	Group buffer pool
GDPS	Geographically Dispersed Parallel Sysplex
GLBA	Gramm-Leach-Bliley Act of 1999
GRS	Global resource serialization
GUI	Graphical user interface
HALDB	High Availability Large Databases
HPJ	High performance Java
HTTP	Hypertext Transfer Protocol
HW	Hardware
I/O	Input/output
IBM	International Business Machines Corporation
ICF	Internal coupling facility
ICF	Integrated catalog facility
ICMF	Integrated coupling migration facility
ICSF	Integrated Cryptographic Service Facility
IDE	Integrated development environments
IFCID	Instrumentation facility component identifier
IFI	Instrumentation Facility Interface
IFL	Integrated Facility for Linux
IMS	Information Management System
IORP	I/O Request Priority
IPL	initial program load
IPLA	IBM Program Licence Agreement
IRD	Intelligent Resource Director
IRLM	Internal resource lock manager
IRWW	IBM Relational Warehouse Workload
ISPF	Interactive system productivity facility
ISV	Independent software vendor
IT	Information technology
ITR	Internal throughput rate, a processor time measure, focuses on processor capacity
ITSO	International Technical Support Organization
IU	information unit
IVP	Installation verification process
J2EE	Java 2 Enterprise Edition
JDBC	Java Database Connectivity
JFS	Journaled file systems
JNDI	Java Naming and Directory Interface
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
KB	Kilobyte (1,024 bytes)
LCU	Logical Control Unit
LDAP	Lightweight Directory Access Protocol
LOB	Large object
LPAR	Logical partition
LPL	Logical page list
LRECL	Logical record length
LRSN	Log record sequence number
LRU	Least recently used
LSS	Logical subsystem
LUW	Logical unit of work
LVM	Logical volume manager

Archive

MAC	Mandatory access control
MB	Megabyte (1,048,576 bytes)
MBps	Megabytes per second
MLS	Multi-level security
MQT	Materialized query table
MTBF	Mean time between failures
MVS	Multiple Virtual Storage
NALC	New Application License Charge
NFM	new-function mode
NFS	Network File System
NPI	Non-partitioning index
NPSI	Nonpartitioned secondary index
NVS	Non volatile storage
ODB	Object descriptor in DBD
ODBC	Open Database Connectivity
ODS	Operational Data Store
OLE	Object Link Embedded
OLTP	Online transaction processing
OP	Online performance
OS/390	Operating S/390®
OSC	Optimizer service center
PAV	Parallel access volume
PCICA	Peripheral Component Interface Cryptographic Accelerator
PCICC	PCI Cryptographic Coprocessor
PDS	Partitioned data set
PIB	Parallel index build
PLSD	page level sequential detection
PPRC	Peer-to-Peer Remote Copy
PR/SM	Processor Resource/System Manager
PSID	Pageset identifier
PSP	Preventive service planning
PTF	Program temporary fix
PUNC	Possibly uncommitted
PWH	Performance Warehouse
QA	Quality Assurance
QMF	Query Management Facility
QoS	Quality of Service
QPP	Quality Partnership Program
RACF	Resource Access Control Facility
RAS	Reliability, availability and serviceability
RBA	Relative byte address
RBLP	Recovery base log point
RDBMS	Relational database management system
RDS	Relational data system
RECFM	Record format
RI	Referential Integrity
RID	Record identifier
RLSD	row level sequential detection
ROI	Return on investment
RPO	Recovery point objective
RR	Repeatable read
RRS	Resource recovery services
RRSAF	Resource recovery services attach facility
RS	Read stability
RTO	Recovery time objective
RTS	Real-time statistics
SAN	Storage area networks
SBCS	Store single byte character set
SCT	Skeleton cursor table
SCUBA	Self contained underwater breathing apparatus
SDM	System Data Mover
SDP	Software Development Platform
SLA	Service-level agreement
SMIT	System Management Interface Tool
SOA	Service-oriented architecture
SPL	Selective partition locking
SPT	Skeleton plan table
SQL	Structured Query Language
SQLJ	Structured Query Language for Java
SRM	Service Request Manager®
SSL	Secure Sockets Layer
SU	Service Unit
TCO	Total cost of ownership
TPF	Transaction Processing Facility
UA	Unit Addresses
UCB	Unit Control Block
UDB	Universal Database
UDF	User-defined functions
UDT	User-defined data types
UOW	Unit of work
UR	Uncommitted read
UR	Unit of recovery
vCF	Virtual coupling facility

VIPA	Virtual IP Addressing
VLDB	Very large database
VM	Virtual machine
VSE	Virtual Storage Extended
VSIP	Visual Studio Integrator Program

Archived

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks publication

For information about ordering these publications, see “How to get Redbooks publications” on page 643. Note that some of the documents referenced here might be available in softcopy only.

- *DB2 9 for z/OS and Storage Management,* SG24-7823
- *DB2 9 for z/OS Performance Topics*, SG24-7473
- *DB2 9 for z/OS Technical Overview*, SG24-7330
- *DB2 9 for z/OS: Distributed Functions*, SG24-6952
- *Extremely pureXML in DB2 for z/OS*, SG24-7915
- *Data Studio and DB2 for z/OS Stored Procedures*, REDP-4717
- *z/OS Version 1 Release 12 Implementation*, SG24-7853
- *DB2 9 for z/OS: Configuring SSL for Secure Client-Server Communications*, REDP-4630
- *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079
- *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7604
- *DB2 9 for z/OS: Packages Revisited*, SG24-7688
- *Securing DB2 and Implementing MLS on z/OS*, SG24-6480
- *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421
- *z/OS Version 1 Release 10 Implementation*, SG24-7605
- *z/OS Version 1 Release 11 Implementation*, SG24-7729
- *IBM z/OS V1R11 Communications Server TCP/IP Implementation Volume 2: Standard Applications*, SG24-7799

Other publications

These publications are also relevant as further information sources:

- *DB2 Version 8 Installation Guide*, GC18-7418-11
- *DB2 10 for z/OS Administration Guide,* SC19-2968
- *DB2 10 for z/OS Application Programming and SQL Guide,* SC19-2969
- *DB2 10 for z/OS Application Programming Guide and Reference for Java,* SC19-2970
- *DB2 10 for z/OS Codes,* GC19-2971
- *DB2 10 for z/OS Command Reference,* SC19-2972
- *DB2 10 for z/OS Data Sharing: Planning and Administration,* SC19-2973

© Copyright IBM Corp. 2010. All rights reserved.

- *DB2 10 for z/OS Installation and Migration Guide,* GC19-2974
- *DB2 10 for z/OS Internationalization Guide,* SC19-2975
- *DB2 10 for z/OS Introduction to DB2 for z/OS,* SC19-2976
- *IRLM Messages and Codes for IMS and DB2 for z/OS,* GC19-2666
- *DB2 10 for z/OS Managing Performance,* SC19-2978
- *DB2 10 for z/OS Messages,* GC19-2979
- *DB2 10 for z/OS ODBC Guide and Reference,* SC19-2980
- *DB2 10 for z/OS pureXML Guide,* SC19-2981
- *DB2 10 for z/OS RACF Access Control Module Guide,* SC19-2982
- *DB2 10 for z/OS SQL Reference,* SC19-2983
- *DB2 10 for z/OS Utility Guide and Reference,* SC19-2984
- *DB2 10 for z/OS What's New?,* GC19-2985
- *DB2 10 for z/OS Diagnosis Guide and Reference,* LY37-3220
- *DB2 Version 9.5 for Linux, UNIX, and Windows Call Level Interface Guide and Reference, Volume 2*, SC23-5845
- *Program Directory for DB2 10 for z/OS*, GI10-8829
- *Program Directory for z/OS Application Connectivity to DB2 10 for z/OS*, GI10-8830
- *Program Directory for DB2 Utilities Suite for z/OS V10*, GI10-8840
- *z/OS V1R10.0 Security Server RACF Security Administrator's Guide,* SA22-7683
- *IBM z/OS V1R11 Communications Server TCP/IP Implementation Volume 2: Standard Applications*, SG24-7799
- *z/OS V1R11.0 Communications Server New Function Summary z/OS V1R10.0-V1R11.0,* GC31-8771
- *Effective zSeries Performance Monitoring Using Resource Measurement Facility*, SG24-6645
- *DFSMSdss Storage Administration Reference,* SC35-0423
- *IBM DB2 Sort for z/OS Version 1 Release 1 User's Guide*, SC19-2961

Online resources

The following websites are also relevant as further information sources:

- DB2 10 for z/OS

  `http://www.ibm.com/software/data/db2/zos/`
- DB2 Tools for z/OS

  `http://www.ibm.com/software/data/db2imstools/products/db2-zos-tools.html`
- Data Studio

  `http://www.ibm.com/software/data/optim/data-studio/`
- z/OS V1R10 IPv6 certification refer to *Special Interoperability Test Certification of the IBM z/OS Version 1.10 Operating System for IBM Mainframe Computer Systems for Internet Protocol Version 6 Capability*, US government, Defense Information Systems Agency, Joint Interoperability Test Command

http://jitc.fhu.disa.mil/adv_ip/register/certs/ibmzosv110_dec08.pdf

- DB2 Sort for z/OS

  http://www.ibm.com/software/data/db2imstools/db2tools/db2-sort/

- DB2 Information Center

  http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db29.doc/db2prodhome.htm

- *IBM zEnterprise 196 I/O Performance Version 1,* technical paper

  ftp://public.dhe.ibm.com/common/ssi/ecm/en/zsw03169usen/ZSW03169USEN.PDF

How to get Redbooks publications

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Archived

Index

Numerics

45829
Head 4
CURRENT TIMESTAMP WITH TIME ZONE special register 174

A

ABIND 478
Acceptable value 58
access 61, 82, 116, 132, 223, 287, 310, 337, 431, 535, 617
access control 133
catalog changes 390
Access Control Authorization Exit 367
access data 367
-ACCESS DATABASE 121
access path 134, 224, 401–402, 487–488, 535, 537
erratic changes 224
wild swings 224
ACCESSCTRL 339
ACCESSCTRL authority 356, 365, 369
grant privilege 371
active DB2
group member 474
subsystem 110
Active log 98–99, 477–478, 518
active version 130, 138–139
ADD VERSIONING USE HISTORY TABLE clause 208
administrative authority 339
ADVISORY REORG 75, 89
advisory REORG-pending (AREOR) 71
aggregate functions 184
aggregation group 184, 187, 189
aggregation group boundaries 194
ALTER
restrictions 77
ALTER BUFFERPOOL 486, 563
ALTER Function 128, 366
main portion 133
remaining portion 134
ALTER statement 71–72, 116, 130, 206, 259, 410
ALTER COLUMN clause 208
COLUMN clause 208
ALTER TABLE 81, 163, 259, 391, 459, 570
DB2R5.CUST OMER 395
DROP Organization 85
statement 85, 91, 100, 103, 209, 263, 396, 570, 579
syntax 84–85
TB1 90
TRYTEMPORAL_B 216
TRYTEMPORAL_E 207
ALTER Table 70, 180, 207, 216, 570, 580
alter table
DDL statement 387
SQL statement 391
ALTER TABLESPACE 55, 73, 75, 115
CHANGES keyword 91
clause 80
command 78, 91, 103
DSSIZE 78–79, 87
FREEPAGE 74, 79, 91
ITSODB.TS1 BUFFERPOOL BP16K0 78
ITSODB.TS1 BUFFERPOOL BP8K0 75–76
ITSODB.TS1 BUFFERPOOL BP8K9 77
ITSODB.TS1 Drop 76
ITSODB.TS1 DSSIZE 79
ITSODB.TS1 DSSIZE 8 G 86
ITSODB.TS1 SEGSIZE 8 86
ITSODB.TS2 MEMBER Cluster 92
MAXPARTITIONS 73–74, 87
option 78, 92
SQL code 77
statement 74, 76, 78
ALTER TABLESPACE statement 74, 84, 116
APAR 55, 120–121, 224–225, 244, 600
APAR II4474 473
APAR II4477 473
APIs 413
APPEND=YES 558
application 52, 55, 67, 125, 262, 310, 317, 337, 535
application developer 166, 204, 371, 603
application period 205
AREO* 71
AREOR 71–72, 75, 116, 487, 570
AREOR state 74
AREOR status 73, 78
argument 130, 245, 400
AS ROW BEGIN attribute 206
AS TRANSACTION START ID attribute 206
attribute 62, 73, 127, 231, 245, 339, 516, 554
ATTRIBUTE2 column 610–611
Audit category 338
CONTEXT 339
DBADMIN 339
EXECUTE 339
group auditable events 354
OBJMAINT 339
SECMAINT 339
SYSADMIN 340
VALIDATE 340
Audit category CHECKING 339
audit policy 338, 488
auditing capability 337
AUDITPOLICYNAME value 343
AUDTPLCY keyword 342
AUDTPLCY parameter 344
display trace command 351
auth ID
DB0BAC 370–371

© Copyright IBM Corp. 2010. All rights reserved.

DB0BDA 363, 368
DB2R51 375–376
DB2R52 373
DB2R53 419
auth Id
trusted context thread reuse 424
AUTHORIZATION Id 60, 130, 338, 514, 605
authorization Id
SECADM authority 362
automatic class selection (ACS) 516
automatic GRECP recovery 120
AUX keyword 452
auxiliary index 271, 569
auxiliary table 92, 277, 454, 569
space 454, 569–570
auxiliary table space 569

B

base table 88, 269, 452, 541
base table space
asynchronous multi-page read requests 545
online reorganization 107
pending definition changes 92
base tables 452
BASE TABLESPACE 454
Basic row format (BRF) 577
BIGINT 149, 249, 515, 582
BIND DEPLOY 140
bind option 128
bind package options 374
bind parameter 537
BIT 134, 287
BLOB 280, 408, 489, 570
BLWLINTHD 35
BLWLTRPCT 35
BSDS 63, 100, 365
BSDS expanded format 477
buffer pool 56, 73, 75, 109–110, 132, 486, 536, 545, 558
activity 119, 562
attribute 77, 562–563
BP0 91
change 521
complete scans 114
CPU overhead 559–560
data entries 120
directory 119–120, 559
entire size 561
manager 563
manager page 563
name 77, 119
page sets 563
relevant pages 119
scan 114
setting 520–521
size 55, 75, 77, 114, 505, 561, 581
space 561
storage 561
support 561
buffer pool management 563
buffer pool scan avoidance 114
buffer pools 114, 120, 618
BUSINESS_TIME period 205, 216
temporal table 216
BUSINESS_TIME semantics 218

C

catalog table 55, 116, 127, 163, 206, 223–224, 307, 332, 340, 496, 571
BRANCH column 397
pertinent records 225
CATMAINT 379, 517
CATMAINT Update 517
CCSID 91, 127, 492
CF 118, 506, 618
Changing the DSSIZE of a table space 80
Changing the page size of a table space 75
Changing the page size of an index 78
Changing the segment size 78
character string 150, 250, 408, 415
explicit cast 156
CHECK DATA 108, 269, 464
CHECK INDEX 269, 465, 546
CHECK LOB 108, 464
CHECK utility 108, 464–465
main purpose 465
Checkpoint Type (CHKTYPE) 97
CICS 110, 235, 309
class 54, 57, 184, 288, 377, 490
CLI 308
CLOB 73, 176, 280, 408, 489, 570
CLUSTER 62–63, 109, 114, 492, 535
CM 107, 224, 361, 535
CM* 476
COLGROUP 87
COLLID 53, 127, 241, 339
colon 155, 317
Column
XML 272
column DRIVETYPE 590
Column mask 360
column access control 398
column mask 209, 367, 376, 384, 395, 497
Column name 72, 88, 340, 359, 474, 571, 579, 582
column value 219, 390, 395, 582
command line
processor 137, 412
COMMENT 55, 408
components 296, 317, 416, 485
compression 3, 55, 64, 103, 225, 277, 518, 621
Compression dictionary 105
compression dictionary 103
condition 137, 251, 391, 604
CONNECT 137, 348
connected DB2 subsystem
IRLM parameters 331
conversion mode 57, 60, 98
parallelism enhancements 593
conversion mode (CM) 473, 475–476, 535–536, 538, 540, 545, 548, 550–551, 559–560, 563, 565, 569–570, 572, 574, 592–593

COPY 86, 105, 364, 478
corresponding history table
  table name 207
corresponding system-period temporal table
  table name 207
COUNT 137, 351
coupling facility (CF) 118
CPU overhead 549, 561
CPU reduction 518, 535, 587
CPU time 57, 559
  significant savings 570
  total cost 559
CREATE TABLE
  LIKE 367
CREATE_SECURE_OBJECT system privilege 376
CREATEIN 364
CREATOR 375
cross invalidation 114, 119
CS 132, 239, 405
CTHREAD 52, 503
CURRENT EXPLAIN MODE 373
CURRENT SQLID 362–363
CURRENT TIMESTAMP 138, 159, 174, 300
  time zone 174
customer table 250, 381, 390
  column access control 390, 395
  column mask objects 398
  following tasks 398
  row access control 391
  row policies 393

D

data 3, 68, 109, 125, 243, 309, 337, 426, 473, 615
data page 79, 83, 105, 115, 118, 558–559, 576
  different set 115
data set 53, 57, 61–62, 69, 73, 80, 274, 311, 418, 477, 561, 565
  maximum number 503
data sharing 55, 77, 110, 132, 354, 468, 474, 598
Data type 72, 88, 90, 132, 134, 154, 205–206, 340, 359, 485, 554, 571
data type
  returns information 166
DATAACCESS 339, 486
DATAACCESS authority 356, 367, 488
database administrator 204, 337
Database name 341
database object 68, 91, 338, 355, 385, 460
database system 154, 156, 183, 203–204
  existing applications 154
  timestamp data 165
DATASIZE 106, 584
DATE 132, 290, 362, 478, 583
date format 132
datetime constants 154
DB0B DSNWVCM1 345
DB2 10 12, 3, 51–52, 67–69, 109–110, 125, 203–204, 243, 337–338, 471, 475, 531, 575
  64-bit COMMON storage 597
  access control feature 385
  access plan stability support 229
  address 107, 354
  audit category 340
  authority 348
  base 484
  break 599
  BSAM buffers 572
  buffer pool enhancements 560
  catalogue 63, 489
  CDB table SYSIBM.USER Name 413
  Centimeters 458
  change 405, 413, 559
  CM9 489
  conversion mode 52, 475
  conversion mode (CM) 535–536, 538, 540, 545, 548, 550–551, 559–560, 563, 565, 569–570, 572, 574, 592–593
  conversion mode 8 476
  conversion mode 9 476
  Curium 107, 361, 458, 473
  data 57
  DB2 catalog 489
  disables compression 56
  early code 111
  ENFM 476
  enhancement 230
  environment 349, 362
  externalize 347, 599
  format 514
  function 520
  group 338
  identity propagation 417–418
  index scan 545
  installation CLIST 502
  Load 569
  make 240
  member consolidation 597
  memory mapping 599
  migration planning 474
  new BIND QUERY feature 498
  new column 359
  new column value 359
  new DSNZPARMs 505
  new enhancements 606
  new function 518
  new z/OS identity propagation 417
  new-function mode 118, 148, 166, 231, 476, 519
  NFM 62, 116, 360, 458, 490, 532
  object changes enhancement 497
  online changes 71
  packaging 480
  page fix 574
  policy 379
  premigration 52
  pre-migration health check job 473
  running 418
  ship 570
  SQL 383
  SQL function 530
  SQL functionality 530

supports DRDA 531
supports index 540
table spaces 499
terminate 474
tolerate 515, 518
track 590
use 565
utility 572
DB2 10 for z/OS
performance improvements 533
DB2 8 68, 103, 126, 460
DB2 9 7, 52, 55, 57, 68, 109–110, 130, 137, 139, 224, 243, 309, 312, 338, 383, 416, 468, 538, 545
big incentive 545
DB2 10 56–57, 71, 77, 130, 229, 315, 383, 468
described behavior 465
documentation 518
dynamic prefetch 546
IPv6 support 316
new driver 53
package 224, 227
partitioned table spaces 458
subsequent REBINDs 227
system 121, 473
DB2 9 XML additional functions 244
DB2 authorization 339
DB2 Catalog 500
DB2 catalogue 77, 85, 140, 163, 474, 484, 490, 559
SMS definitions 517
DB2 command
interface 348
DB2 family 125, 591
other products 129
DB2 issue 226, 347, 503
DB2 member 109–111
DB2 QMF
Enterprise Edition 482
DB2 startup 340, 574
Automatic policy start 340
automatic start 343
DB2 subsystem 1, 51, 54, 97, 110–111, 132, 354, 356, 464–465, 475, 536, 575
query tables 367
DB2 system 107, 215, 359, 480, 606, 610
DB0A 412–414
privilege 359
DB2 utility 61, 63, 381, 427, 468, 484
additional EAV support 63
DB2 V8 63, 68, 100, 120, 227, 312, 458, 473, 475
behavior 120
Curium 475
end of service 472
function 312
group attachment behavior 120
NFM 13, 475, 517
supported column type 515
system 474
DB2 V9 54
56-way system 54
DB2 Version
7 599
8 120, 458
8 Installation Guide 478
9 562, 601
DB2 Version 1 54, 575
DB2 Version 5 115, 553
DB2 Version 8 111–112
DB2R5.AUDE MP 350
Value 406
DB2R5.CUST OMER 392
PERMISSION RA02_CUSTOMERS 402
DB2R5.CUSTOMER 391
PERMISSION RA01_CUSTOMERS 393
DB2-supplied routine 523
Validate deployment 529
DBA toolbox 578
DBADM authority 363
DBADMIN column 367, 369
DBAT 315, 548, 603
DBCLOB 176, 280, 570
DBD01 121, 491
DBET 121, 465
DBID/OBID 405
DBM1 address space 52, 530, 536
virtual storage usage 599
DBMS 118, 204, 243
DBNAME 72, 76, 339
DBPROTOCOL 531
DBRM 233–234, 372, 478, 599
DDF 109, 414, 535
DDF command 117, 312–313, 549
DDF indoubt URs 117
DDF location 598
DDF restart light 117
DDL 55, 73, 91, 307, 385, 491, 584
DDLTOX 632
DECFLOAT 132, 149, 251, 579
decimal 132, 251
decomposition 246, 482
default value 57, 106, 130, 225, 231, 245, 361, 377, 502, 569, 572
deferred write 47
DEFINING SECTION (DS) 616
definition change 71–72, 74, 487
DEGREE 53, 131, 622
DEL_CFSTRUCTS_ON_RESTART 506, 618
DELETE 119, 278, 350, 466, 474, 569
delete 237, 282, 341, 496, 616
delete name 119–120
dependent privilege 383
cascading revokes 383
DEV Type 94
DFSMS 428, 473
DIS DB 72, 459–460
DIS LOG
command output 99
Output 98
dis thd 416
DIS Trace 352
-DISPLAY DDF 312

-DISPLAY GROUP DETAIL 111
-DISPLAY LOCATION 311
-DISPLAY LOCATION DETAIL 311
display profile 374
DISPLAY THREAD 416
Distributed 418
distributed data facility (DDF) 3, 52, 109, 308, 310
  availability 310
Domain Name 315
domain name 312, 315
DRDA 165, 262, 310, 373, 481
DS8000 3, 62
DSN_STRUCT_TABLE 404
DSN_XMLVALIDATE 266
DSN088I 313–314
DSN1COMP 105
DSN1COPY 116
DSN6SPRM 59, 225, 501
DSN6SYSP 501
DSNAME 94, 262
DSNDB01.SYSUTILX 468
DSNJU003 98
DSNJU004 312, 468, 477
DSNL519I 315
DSNL523I 315
DSNLEUSR 413
DSNT404I SQLCODE 75, 89, 382
DSNT408I 77, 90, 373
DSNT408I SQLCODE 77, 103
DSNT415I SQLERRP 75, 77
DSNT416I SQLERRD 75, 77
DSNT418I SQLSTATE 75, 77
DSNTEP2 137, 373
DSNTIJEN job 495
DSNTIJIN 99, 490
DSNTIJMV 528
DSNTIJPM 52, 473
DSNTIJTM 60
DSNTIJXA 510
DSNTIJXB 511
DSNTIJXC 513
DSNTIJXZ 520
DSNTIP7 306
DSNTIP9 58, 538
DSNTIPK 111
DSNTIPN 64
DSNTIPSV 520
DSNTSMFD 64
DSNTWFG 60
DSNTXTA 510
DSNTXTB 511
DSNUM 94
DSNWZP 381
DSNZPARM 59, 119, 225, 274, 344, 501, 538, 618
  ABIND 478
  CACHEDYN 240
  CHKFREQ 97
  CHKLOGR 97
  CHKMINS 97
  CHKTYPE 97
  FCCOPYDDN 427
  FLASHCOPY_COPY 426
  FLASHCOPY_PPRC 426
  IMPDSDEF 306
  LOB_INLINE_LENGTH 570
  PLANMGMT 225
  REVOKE_DEP_PRIV 383
  REVOKE_DEP_PRIVILEGES 383
  SECADM1 380
  SEPARATE_SECURITY 344, 361, 377
  SMFCOMP 64
  TCPALVER 413
DSNZPARM parameter
  ACCUMACC 601
  ACCUMUID 601
  PTASKROL 601
dsnzparm SEPARATE_SECURITY 365
DSNZPARM SPRMRRF 69
DSNZPARMs
  for the SECADM authority 360
DSSIZE 57–58, 60, 68, 89, 490, 564
duplicate LRSN value 119, 557
duplicate row 190
  alternative orderings 193
  possible orderings 193
Dynamic prefetch 545
Dynamic SQL
  access path 224
  query 221
dynamic SQL 131, 224, 350, 553, 620
  access path stability 230
  OMPE record trace 354
dynamic SQL statement 405
Dynamic statement cache 535, 616
Dynamic VIPA 315

E

EAV volume
  format 62
  list 62
EDMPOOL 56
efficiency 337
EIM 416
element 154, 247, 624
empty Lob 567
enable-new-function mode (ENFM) 476
end column 205
ENFM 468
enterprise identity mapping (EIM) 416
ENVID value 390
environment 53, 72, 77, 110, 115, 140, 223, 262, 348, 485, 537, 627
Equivalent SQL 164
error message 76, 89–90, 314, 373, 428, 452, 516, 529
EXEC SQL 233, 351, 555
EXPLAIN 132, 541
  QUERYNO 374
Explain 241, 482
EXPLAIN authority 371
EXPLAIN output 403, 541

Archived

explicit cast 149, 232
truncation semantics 151
expression 126, 232, 245, 366, 476, 540
extended address volume
Support 62
extended address volume (EAV) 62
extended addressability (EA) 516
Extended addressing space (EAS) 62
extended correlation token 316
extended format
data sets 516
extended format (EF) 62, 516, 571
extended indicator variable 231
extended indicator variables 230
extended timestamp precisions
SQLTYPE 392/393) 530
EXTERNAL Action 127
external length 157

F

FALLBACK SPE
APAR 474
Applied 474
Fallback SPE 474
fallback SPE 474
FCCOPYDDN 427
FETCH 275, 535
fetch 131, 274, 415, 540, 616
FICON 3
file reference variables 565–567, 569
FINAL TABLE 280
FlashCopy 12, 3, 22, 425, 535
FlashCopy enhancements 426
FlashCopy image copy 426, 444
with the COPY utility 426
flexibility 13, 100, 115, 517, 606
fn
contains 252
FOR BIT DATA 287
FOR SYSTEM_TIME AS OF timestamp-expression 213, 216
fractional second 155, 157
missing digits 158
function 3, 56–57, 107, 125, 204, 244, 361, 465
FUNCTION TRYXML 136
function version 133, 139

G

GB bar 52
data stores 53
virtual storage 52
GENERATED ALWAYS 277, 407
GENERATED BY DEFAULT 92
getpage 546
given table space
currently pending definition change 91
other running processes 461
XML data 566
GRANT ACCESSCTRL 380
GRANT DATAACCESS 363
GRANT option 359
graphic string 148
implicit cast 149
GRECP 120
group attach name 110
group attachment 110, 120
GRP 110

H

handle 51, 86, 120, 230, 449, 518, 567
hard disk drive (HDD) 590
Hash access
bad choice 578
perfect candidate 578
hash access 499, 536, 560
hash key 85, 577–578
new columns 580
unique hash overflow index 580
HASH space 81, 83, 578, 584
hash space 576
extra space 578
hash table 576
hash overflow index 583
other access path options 580
space 80, 85
table space 577, 580
heavy INSERT
application 92
environment 92, 115
HIGH DSNUM 94
High Performance Option (HPO) 486
HISTOGRAM 87
history table 205–206
corresponding columns 210
ROWID values 210
host variable 163, 231–232, 280, 415, 537, 540, 581
target column 231
host variables 231, 281, 415, 537

I

I/O 122, 502, 535, 544, 622
I/O parallelism 558
I/O subsystem 575
IBM DB2
Driver 482
server platform 338
Utility 484
IBMREQD 73, 76, 408
ICTYPE 87, 96, 451–452, 571
IDCAMS 62–63, 99
identity propagation 416
IEAOPTxx 35
IFC DESCRIPTION (ID) 338
IFCID 140 340
IFCID 141 341, 363
audit trace record 363
grant 339
IFCID 142 341

IFCID 143 341, 349
IFCID 144 351
IFCID 145 405
IFCID 148 539
IFCID 225 599, 618
  Duplicate data 599
IFCID 269 342
IFCID 270 341
IFCID 3 599
IFCID 306 55
IFCID 359 119, 598, 622
IFCID 361 348, 367
  audit trace record 369
  trace record 348
IFCID 362 342
  RECTRACE report 347
  trace record 347
IFCID 365
  record 602
  trace 602
  trace data 602
IFCID record 411, 599
IFICD 23 340
II10817 628
II14219 628
II14334 628
II1440 628
II14426 244, 628
II14438 635
II14441 628
II14464 628
II4474 628
II4477 628
image copy 93, 426, 519, 570
IMMEDWRITE 132, 226
IMPLICIT 77, 391
implicit cast 148–149
implicit casting 148
  Examples 150
implicit casting optimization 154
implicit time zone 170
  possible difference 175
IMS 235, 309, 484, 576
INCLUDE XML TABLESPACES 270
index 70, 109, 114, 163, 251, 340, 489, 622
index ORing 539
index page
  split 119, 557, 622
indicated time zone
  time zone value 176–177
Indicator variable 231
indicator variable 181, 230–231
  negative value 234
  special values 231
infix arithmetic operators 150
informational APARs 473
inline copy 96, 454
INLINE Length 570–571
inline LOB 569
inline Lob 569
inline SQL scalar function 127
  KM2MILES 136
  return 137
IN-LIST enhancement 540
IN-list predicate 541
  transitive closure predicates 541
inner workfile 593
input parameter 60
INSERT 92, 103, 109, 152, 231, 247, 343, 555
insert 53, 122, 152, 230, 247, 341, 496, 535, 616
installation 52, 131, 225, 306, 340, 471, 565
installation CLIST 60, 477, 502
  different name 521
installation job
  DSNTIJRT 413
  DSNTIJSG 606
INSTANCE 94, 348, 527
instrumentation facility interface (IFI) 603
Interactive Storage Management Facility (ISMF) 516
International Standards Organization (ISO) 155
IP address 310, 312, 317, 419–420, 488, 605, 607
IPLIST 310
IPNAMES 310, 413
IPv6 support 316
IRLM 118, 468, 473, 599, 618
IS 60–61, 77, 98, 114, 281, 346, 428, 475, 567, 622
ISO format 154–155
IX 114

J

Japanese industrial standard (JIS) 132
Java 156, 308–309, 482, 535
JCC 308, 554
JDBC 110, 280, 381, 482
JDBC driver 381, 485

K

KB 55, 75, 500, 504, 538
KB buffer pool 56, 545
KB chunk 561
KB page 548, 561
keyword 72, 75, 158, 261, 312, 342, 487, 554

L

LANGUAGE SQL 127, 136, 304
LARGE 75, 585
LDAP 416
left hand side
  UTC representation 172
LENGTH 157, 460, 570
LIKE 137, 280, 339
list 59, 111, 137, 232, 343, 535, 618
list prefetch 537
LOAD 116, 181, 237, 261, 368, 426, 501, 546
LOB 77, 149, 243, 408, 452, 481
LOB column 53, 79, 491, 566–567, 570
  inline length 571
  inline length attribute 571

Archive

inline portion 570
inlined piece 570
LOB object 77, 88
LOB table 70, 77, 271, 453, 489, 567
LOB table space 69–70, 79, 89, 307, 454–455, 458, 491, 497, 514, 567
REORG SHRLEVEL options 458
space reclamation 458
LOB/XML 306, 566
LOBs 56, 149, 271, 481, 489, 536
processing 308, 566
location alias name 311
LOCATIONS 310, 413
LOCK 118
locking 51, 55, 109, 279, 489, 583
locks 117, 273, 462, 504
LOG 73, 94, 487, 575
log record 54, 96, 503, 575
log record sequence number 118
log record sequence number (LRSN) 118
LOGGED 94, 492
logging 51, 67, 118
logical aggregation group 192
LOGICAL Part 94
logical partition 100, 102, 460, 624
logical unit of work (LUW) 338
Logical Unit of Work identifier 316
LOGICAL_PART 101–102
LOGRBA 353
long-running reader 106, 461, 504
LOW DSNUM 94
LPAR 111
LPL 121
LRSN 94, 118, 277, 557
LRSN spin avoidance 118
LUMODES 310

M

M 81, 154, 292, 341
machine- readable material (MRM) 73
MAINT mode 520
maintenance 119, 121, 128, 243, 332, 474, 627
map page 115
MASK/PERM 405
mass delete 239
materialization 74, 78, 308, 538
materialized query tables 400
maximum number 59–61, 69, 80, 165, 503, 538, 550
default value 504
MAXOFILR 52
MAXTEMPS 538
MB page
frame 561
MEMBER CLUSTER 74, 115, 558
MEMBER Cluster 74, 115, 536, 577
member cluster 92
MEMBER CLUSTER column 499
MEMBER CLUSTER for UTS 114
MEMBER Name 94, 110
MERGE 103, 231, 278, 604
Migration 381
MODIFY 311, 488, 549
-MODIFY DDF ALIAS 311
MONSIZE 503
moving aggregates 184
MSTR address space 107
msys for Setup 481
MVS LPAR 620

N

namespace 255
native SQL procedure 148, 301
native SQL stored procedures 300
new-function mode 55, 115, 473
DB2 10 57, 115, 126, 476
DB2 9 118, 148, 476
NFM 103, 224, 338, 535
node 252, 583
non-inline SQL scalar function 127
non-LOB 565–566
non-XML data 566
NORMAL Completion 98, 345, 460
NPAGESF 94
NPI 546
NULL 72, 126, 220, 245, 340, 514, 567
null value 130, 177, 213, 231, 397, 583
numeric value 149
Implicit cast 150
NUMPARTS 57, 116, 507

O

OA03148 634
OA17735. 35
OA24811 426
OA25903 262
OA31116 634
OA33106 634
OA33529 634
OA33702 634
OA34865 634
OA35057 635
OA35885 635
OA38419 633, 635
Object 72, 341
OBJECTNAME 339
OBJECTTYPE 339
OBJECTTYPE column 339
ODBC 53, 110, 165, 280, 484, 535
ODBC application 53, 165, 572
ODBC driver 53
OLAP specification 184
important concepts 184
OLAP specifications 184
Online Communications Database changes 310
Online DDF location alias name changes 311
online performance (OP) 539
online reorg 106, 108, 452, 490
online schema change 71, 93, 518
optimization 132, 223, 307, 535

Optimization Service Center (OSC) 482
options 12, 72–73, 127, 225, 271, 337, 434, 541
ORDER 56, 102, 184, 246, 398, 539
ORDER BY 188, 250, 544
ORDER BY clause 188, 250
ordering 188

P

package bind
owner 365
page access 545
page level sequential detection 547
page set 63, 71–72, 88, 93, 110, 114, 119, 353, 431, 548, 561
page size 56, 60, 68, 70, 77, 492, 500, 545, 564
different buffer pool 78
valid combinations 80
parameter marker 132, 165, 231, 537
PART 83
PARTITION 60–61, 86, 88, 180, 585, 622
partition-by-growth 68
partition-by-growth (PBG) 57, 490
partition-by-growth table space 69–70, 73
MAXPARTITIONS 73, 87
structure 73
partition-by-growth table spaces
in workfiles 57
PARTITIONED 103
partitioned data set (PDS) 477
partitioned data set extended (PDSE) 477
partitioned table space 116, 432, 506
disjoint partition ranges 107
Partitioning 187, 587
partitioning 184, 187, 277, 501, 577
partitions 57, 121, 132, 308, 458, 501, 577
password phrase 411
RACF user 412
PATH option 245
PBG table space 457, 492–494, 577, 579–580
PBR table space 453, 577, 579
pending changes 71
Performance 12, 17, 338, 486, 535
performance 1, 3, 57, 109, 115, 204, 243, 337, 485
performance improvement 57, 148, 557, 559
period, 204
PGFIX 561
PHASE Statistic 454, 586
physical aggregation
group 189
physical aggregation group 189
PIC X 163
PIT LRSN 94
PK28627 628
PK51537 258
PK51571 244, 250, 254
PK51572 244, 250, 254
PK51573 244, 250, 254
PK52522 224
PK52523 224
PK55585 258
PK55783 251
PK55831 258
PK55966 279
PK56922 474
PK57409 244
PK58291 63
PK58292 63
PK58914 244
PK62876 478
PK65772 511
PK70423 516
PK76100 628
PK79327 120
PK79925 478
PK80320 121
PK80375 55, 225
PK80732 252, 257
PK80735 252, 257
PK80925 122
PK81151 63
PK81260 251
PK82631 257
PK83072 53
PK83735 122, 558
PK85068 508
PK85833 478
PK85856 49
PK85889 49
PK90032 50, 262
PK90040 50, 262
PK94122 122
PK96558 252
PK99362 53
PM00068 628
PM01821 478, 628
PM02631 121
PM02658 331
PM03795 121
PM04968 474, 628
PM05255 121
PM05664 244
PM06324 121
PM06760 121
PM06933 121
PM06972 121
PM07357 121
PM11441 121
PM11446 121
PM11482 244
PM11941 332
PM12286 567
PM12819 467
PM13466 628
PM13467 628
PM13525 148, 480, 628
PM13631 148, 628
PM17542 9, 63, 628
PM18196 49, 628
PM18557 9, 629
PM19034 629

Archived

PM19584 629
PM21277 467, 629
PM21747 467, 629
PM22091 331
PM22628 635
PM23887 635
PM23888 635
PM23889 635
PM24082 635
PM24083 635
PM24721 565, 629
PM24723 629
PM24808 332, 629
PM24808. 501
PM24937 230, 629
PM25282 629
PM25357 629
PM25525 629
PM25635 332, 629
PM25648 629
PM25652 629
PM25679 629
PM26480 310, 629
PM26762 426, 629
PM26781 310, 629
PM26977 344, 629
PM27073 629
PM27099 630
PM27811 630
PM27828 630
PM27835 630
PM27872 64, 630
PM27973 630
PM28100 630
PM28296 344, 630
PM28458 630
PM28500 630
PM28543 344
PM28796 630
PM28925 630
PM29037 630
PM29124 480, 630
PM29900 630
PM29901 630
PM30394 344
PM30425 630
PM30468 47, 630
PM30991 630, 633
PM31003 112, 630
PM31004 112
PM31006 112
PM31007 112
PM31009 112
PM31243 631
PM31313 631
PM31314 631
PM31614 631
PM31641 631
PM31807 631
PM31813 631
PM32217 344
PM32638 635
PM32647 635
PM33501 631
PM33767 631
PM33991 631
PM35049 635
PM35190 631
PM35284 631
PM36177 631
PM37057 631
PM37112 631
PM37300 631
PM37622 631, 633
PM37625 631
PM37630 632
PM37647 632
PM37660 632
PM37816 632
PM38164 632
PM38417 632
PM39342 632
PM41447 632
PM42331 632
PM42528 632–633
PM42924 632
PM43292 632
PM43293 632
PM43597 632
PM43817 632
PM45650 632
PM45651 632–633
PM45810 633
PM47091 631, 633
PM47616 633
PM47871 636
PM51467 633, 635
PM51655 633
PM51945 632–633
PM52012 633
PM52327 633
PM52724 633
PM53237 633
PM53243 633
PM53254 633
PM54873 632–633
PM55051 633
PM55928 633
PM56631 633
PM56845 633
PM57206 633
PM57632 633
PM57878 633
PM58177 633
PM62797 634
PM66287 634
PM67544 634
PM67565 636
PM69522 634
PM70046 634

Archived

PM70270 634
PM70575 634
PM70645 636
PM71114 634
PM72526 634
PM80779 634
PM82301 634
PM84750) 634
PM86952 634
PM91930 634
PM94885 634
policy based audit 338
POLICY Integer 207
PORT 312–313, 413
POSITION 566
precision 12 157
  timestamp value 158
precision x 159
  time zone 173
precompiler 483
predicate 57, 196, 232, 250, 339
prefetch quantity 545
  Additional requests 545
prefix 110, 150
prior DB2 release 127
Private protocol 481, 529, 602
Procedural Language (PL) 126, 536
PROCESSING SYSIN 428, 517
progressive streaming 308, 573
PSRBD 81
PTF 262, 526, 600
pureXML 7, 243

Q

qualified name 268
QUALIFIER 141, 226, 367, 428
query 1, 52, 106, 116, 127, 224, 245, 317, 339, 476, 535
query performance 402, 485, 587
QUERYNO 241, 402, 623

R

RACF 25, 337, 428, 473
RACF group 362
RACF profile 367, 377
RACF profiles 363
RACF user
  DB2R51 419
  Id 416
RANDOM 111
RANDOM ATTACH 111
RANDOMATT 120
range-partitioned table space 69–71, 74
  SEGSIZE 89
range-partitioned table spaces 68
RBA 54, 94, 277, 449, 575
RBDP 71, 78, 580
re/bind 535
read stability (RS) 504
READS 127, 304, 382, 605
Real 57, 381
real storage 561, 620
  potential spike 597
reason code 262, 346, 531, 610, 618
REBIND PACKAGE 128, 225, 487
REBIND Package 128, 225, 374
  SWITCH option 227
rebuild pending 580
RECOVER 93, 116, 487, 570
Redbooks Web site
  Contact us xxxiii
referential integrity 500
REGION 349
relational data system (RDS) 404
RELCREATED 76, 90, 408
remote location 310, 412–413, 530–531, 602
remote server 309–310
RENAME INDEX 55, 92
REOPT 132, 537
reordered row format 277, 577
reordered row format (RRF) 69
REORG 79, 83, 115–116, 163, 274, 368, 426, 452, 481, 486, 535, 546
REORG Index 73, 78, 88
REORG TABLESPACE 73, 78, 86, 89, 452–453
  control statement 458
  DSN00063.PART TB 455
  job 458
  pending definition change 74
  ROTATE.TS1 2
    4 460
  SHRLEVEL Change 91, 458
  SHRLEVEL Reference 86, 458
  SHRLEVEL REFERENCE job output 86
  statement 94, 107
  syntax 454
  TESTDB.TEST TS SHRLEVEL Reference 453
  TESTDB.TS1 PART 3 459
  utility 88, 91, 462
  utility execution 95
REORG utility 86, 104–105, 116, 274, 453, 570, 579
REORP 71
REPAIR 95, 368
repeatable read (RR) 504
REPORT 93–94, 269, 313, 349, 468
REPORT RECOVERY utility 469
repository 263, 488
requirements 116, 231, 243, 309–310, 337, 472, 536
  system software 473
RESET 102
resource unavailable 579
RESTART 98, 100, 475, 621
restart light 117, 340
RESTRICT 72, 146, 382
return 54, 87, 89, 138, 245, 312, 392, 428, 537
RETURN Code 75, 428, 456, 517, 527
RETURNS VARCHAR 140
Revoke statement syntax 383
RID 59, 465, 504, 621
RIDs 59, 538

Archived

ROLLBACK 100, 137
root anchor point (RAP) 577
ROUND_HALF_UP mode 152
row access
control 380
control activation 391
control activation DB2 insert 390
control rule 393
control search condition 394
rule 385, 393
row and column access control 384
row format 277, 569
row length 56
row level sequential detection 547
Row level sequential detection (RLSD) 546
Row permission 360, 376, 389, 497
DB2 description 408
enforcement 385
new SQL statements 411
object 386
row permissions 209, 364–365
row-begin column 205
row-end column 208
maximum timestamp value 212
ROWID 73, 408
RRSAF 110, 235
RTS 83, 104, 558
RUNSTATS 86, 145, 332, 374, 476

S

same page 73, 77, 118, 121, 547
next row 547
same way 109, 149, 361, 567
SCA 118, 506, 618
scalar aggregate functions 184
scalar functions 184
SCHEMA 240, 347, 497
Schema 92, 263, 341, 514
schema 67–68, 127, 221, 250, 339, 482
SCOPE PENDING 269
SCOPE XMLSCHEMAONLY 270
SDSNLINK 63
SECADM authority 339, 344, 359, 488
auth ID 390
SECADM DSNZPARMs 361
second ALTER
change 91
TABLESPACE command 91
SECURITY 423
segment size 58–59, 76, 506
segmented table space 58, 69, 507
SEGSIZE 58, 74, 115, 492
segsize 74, 87, 115
SEGSIZE value 115
SELECT CURRENT_TIMESTAMP 160
SELECT Id 189
SELECT Policy 211
SELECT TRYTMSPROMO 171
SEPARATE_SECURITY 377
Separation of duties 354
SEQ 94
Server 286, 315, 337, 473
SESSION TIME ZONE 175
SESSION TIMEZONE 175
alternative syntax 175
SET 90, 111, 137, 234, 260, 315, 343, 428, 478, 544, 622
-SET SYSPARM command 111
SGRP 110
shared communications area 118
SHR LVL 94
SHRLEVEL 78, 272, 481, 558
SHRLEVEL Change 74, 426
SHRLEVEL None 85
SHRLEVEL Reference 74, 431, 458
Shutdown and restart times 64
side 79, 101, 172, 286, 542
simple table space 69, 87
SKIP LOCKED DATA 238
skip-level migration 472
SMF 9, 64, 338, 506, 618
SMS definition 517
SNA 316
solid state drive (SSD) 590
sort key 56, 189, 593
data type 196
length 56
ordered sequence 190
sort merge join (SMJ) 593
sort record 56, 564
row length 56
sort key length 56
source control data set (SCDS) 516
source text 73, 408
space map 89, 105, 115
page 105, 115
sparse index 593
spin avoidance 118
SPT01 481
SPUFI 137, 301
SQL 55, 75, 116–117, 203, 244, 317, 337, 339, 460, 476, 535, 553, 620
SQL access
path 224
plan 373, 537
SQL Data 127, 382
SQL DDL
interface 387
statement 387
SQL DML
attempt 373, 395
operation 384, 386, 391
operation reference 397
privilege 369, 373
statement 350, 373, 384
SQL interface 330, 358
SQL PL 126, 278, 591
SQL procedure 148, 301, 532, 535, 591
SQL procedures 137, 224, 490, 535
SQL Reference 126, 342
DB2 10 128, 359

SQL scalar 126, 250
function 126, 302
function body 126
function PRIVET 141
function processing 147
function TRYTMSPROMO 170
function version 139
SQL scalar function 126, 302
ALTER FUNCTION 133
examples 136
syntax changes 141
versioning 138
SQL scalar functions 126
SQL statement 52, 56, 81, 89, 126–127, 147, 206, 230–231, 250, 341–342, 485, 517, 545–546, 620
authorization requirement 366
DISTINCT specifications 56
metadata information 380
original source CCSID 604
semantics checks 372
syntax check 374
time information 349
trace records 349, 556
SQL statement text 349, 556
SQL stored procedure 301
SQL stored procedures 300
SQL table 126
SQL table function
ALTER FUNCTION 146
syntax changes 148
SQL table functions 144
SQL variable 126, 132, 302, 591
SQL/XML 246
SQL0217W 375
SQLADM 339
SQLADM authority 356, 358, 371
query table 376
SQLCODE 56, 72, 75, 116, 153, 165, 262, 343, 362, 497, 564
SQLCODE +610 75
SQLCODE -104 365, 368
SQLCODE –20497 170
SQLERROR 128, 372, 486
SQLID DB0B 399
SQL DML statement 400
SQLJ 240, 280, 482
SQLJ applications 287, 482
SQLSTATE 56, 75, 165, 375, 497, 584
SQLSTATE 22007 170
SSID 60
SSL 312
START LRSN 94–95
START Trace 338, 488, 597
startup procs 528
statement 52, 56, 115, 126, 247, 317, 535, 616
Static SQL
COBOL program 351
static SQL 132, 224, 350–351, 540, 553
DB2 9 VSCR 224
host variables 553
last bind time 604
OMPE record trace 353
statistics 57, 86–87, 145, 332, 374, 476
STATUS 102, 313, 347, 460, 517, 537
STATUS VARCHAR 207
STMTID 241
STOP keyword 312
STOSPACE 365
straw model 595–596, 623
string representation 132, 149, 155, 169
standard formats 155
time zone value 170, 172
string value 149, 248
STYPE 87, 451, 571
subgroup attach 110
subgroup attach name 110
subquery predicate 543
SUBSTR 151, 308, 391, 570
SUBSYSTEM Id 514
synchronous I/O 546
Syntax Diagram 128, 383
SYSADM 61, 337, 520
SYSADM authority 343–344, 348, 361
Audit use 343
SECADM authority 355, 379
security administration 355
security administrative duties 379
SQL access 380
SYSCOLUMNS 86, 163, 389, 493, 582
SYSCOPY 86–87, 500, 571
SYSCOPY entry 94
SYSCTRL 340
SYSFUN.DSN_XMLVALIDATE 262
SYSIBM.DSN_PROFILE_ATTRIBUTES 317, 605
SYSIBM.DSN_PROFILE_TABLE 317, 605
SYSIBM.IPLIST 316
SYSIBM.SYSA UDITPOLICY 340
SYSIBM.SYSAUDITPOLICY 340, 343
audit policy 340
catalog table columns 342
policy row 343
SYSIBM.SYSC OLUMNS 206, 390, 516
DEFAULT column 206
PERIOD column 206
SYSIBM.SYSC ONTROLS 390, 497
SYSIBM.SYSC OPY 93, 432
corresponding row 451
entry 94
Record 452
table 451
SYSIBM.SYSCOLUMNS 390, 407, 499
SYSIBM.SYSCONTROLS 389, 497
SYSIBM.SYSCOPY 87, 426
SYSIBM.SYSD EPENDENCIES 390, 394
SYSIBM.SYSD UMMY1 140, 375
SYSIBM.SYSDBAUTH 517
SYSIBM.SYSDEPENDENCIES 389, 409
SYSIBM.SYSDUMMY1 138, 293, 402
SELECT SESSION TIME ZONE 176
SELECT SESSION TIMEZONE 176

sysibm.sysdummy1
select 1 375
SYSIBM.SYSENVIRONMENT 390
SYSIBM.SYSINDEXSPACESTATS 499, 584
SYSIBM.SYSOBJROLEDEP 409
SYSIBM.SYSP ENDINGDDL 76, 497
new row 78
SYSIBM.SYSPACKDEP 406
SYSIBM.SYSPENDINGDDL 72, 116, 497
SYSIBM.SYSQ UERY 498
SYSIBM.SYSQUERY 221, 498
SYSIBM.SYSROUTINES 127, 380, 409, 499
SYSIBM.SYST ABLES 207, 390
column access control 390
SYSIBM.SYST ABLESPACE 83
updates column DATAPAGES 84
SYSIBM.SYSTABLEPART 101, 307, 499
SYSIBM.SYSTABLES 389, 410, 499
SYSIBM.SYSTABLESPACE 74, 116, 499
SYSIBM.SYSTABLESPACESTATS 106, 499, 584
SYSIBM.SYSTRIGGERS 410
SYSIBM.SYSU SERAUTH 359
SYSIBM.SYSUSERAUTH 359, 410, 499
SYSIN 94, 99, 237, 349, 428, 517
SYSLGRNX 449, 495
SYSOPR 340, 520
SYSPACKAGE 53, 127, 220, 239, 493
SYSPACKSTMT 350, 491, 604
SYSPARM command 111
SYSPRINT 61, 99, 237, 428, 521
SYSTABLEPART 86, 101, 307, 493, 571
system authorities and privileges 354
System DBADM 339, 341, 349
auth ID 365
System Level
Backup 468
system parameter 62, 342, 501, 605
SEPARATE_SECURITY 378
system period 205
System z 3, 51, 472, 485, 533
System z9 600
SYSTEM_TIME period 205–206
end column 209
system-period data 205
temporal table 209
system-period data versioning 205
system-period temporal table 205
new period-specification clauses 213

T

table expression 542
TABLE OBID 353
table owner 373, 379, 384
table row 249, 343, 559
table space
AREOR state 87
automation 96
buffer pool 75
CHKP status 465
CONSIST.TS1 465
copy information 469
data 590
data pages 79, 115, 561
data set 74
DB1.TS1 89–90
DB2 data compression 105
definition 72, 74, 497, 546
DFSMS data class 502
DSNDB06.SYST SCTD 497
execution 102
fixed part 582
FREESPACE percentage 585
GBP-dependent status 121
HASHDB.HASH TS 81
intent 548
level 116, 549
lock 239
LOCKSIZE definition 583
logical end 100–101
name 492, 499
NOAREORPEND state 96
option 72, 92, 306, 578
organization 68, 83, 582
original SEGSIZE 74
other tables 211
page 561
page set 76, 432
page size 75
partition 79
pending definition changes 72, 86, 91
REORG TABLESPACE 82–83, 86
restrictive states 108
scan 537
segment size 78
structure 68
SYSPACKAGE 242
tiny clusters 81
type 564
type conversion 69
unconditional separation 59
user data sets 86
table space (TS) 61, 68, 73, 78, 106, 109, 115, 209, 339, 432, 482, 504, 536, 545
table space scans 545
tables 55, 61, 80, 115, 130, 246, 310, 317, 337, 474, 538
TABLESPACE statement 57, 68, 75, 115–116, 449, 506
TABLESPACE TESTDB.TEST TS
PART 1 454
PART 2 454, 456
PART 3 454
TBNAME 382, 497, 540
TCP/IP 312, 423
temporal 204
TEXT 628, 634–635
th
tm 166
TIME 86, 290, 348, 600
time stamp data 290
TIME Zone 129, 155, 166, 168, 205
2-byte representation 168

Alternative spelling 167
C1 TIMESTAMP 175
C2 TIMESTAMP 175
CURRENT TIMESTAMP 174
hour component 179
minute component 166
new data type TIMESTAMP 166
nullable timestamp 182
precision 0 168
precision 12 173
precision 6 173
second byte 168
SELECT CURRENT_TIMESTAMP 174
TMSTZ_DFLT TIMESTAMP 167
valid range 168
valid string representation 178
time zone value 168–170
times DB2 583, 585, 616
TIMESTAMP 73, 86, 153, 166, 204, 290, 340, 408, 478, 622
timestamp column 158, 205–206
timestamp data 153, 156–157, 163, 530
type 156–157
timestamp data type 156–157
timestamp precision 156–157
input data 165
timestamp value 155–156, 159, 212
character-string or graphic-string representations 176
String representation 157
TIMESTAMP WITH TIME ZONE 166
TOTALROWS 584
TRACE privilege 348
Trace record
OMPE formatted audit trace 363
trace record 339, 347, 504, 506, 556, 598, 618
traces 338, 468, 488, 603
transitive closure 541
triggers 204, 242, 360
TRUNCATE 364, 566
Trusted context 422
TRYIMPCAST Value 152
TS 102, 453
TYPE 60, 74, 116, 127, 249, 346, 460, 516, 570
Type 2 583
Type 4 308, 573

U

UA07148 634
UA55970 634
UA56174 634
UA57243 634
UA57254 634
UA57704 634
UA58937 635
UA60823 635
UA66416 635
UA66420 635
UDF 366, 381–382
UDFs 300, 365
UK33456 279
UK37397 628
UK44120 628
UK50918 53
UK52302 53
UK53480/1 628
UK56305 628
UK56306 628
UK58204 628
UK59100 467
UK59101 467
UK59680 567
UK59887 629
UK60466 467, 629
UK60887 628
UK61093 635
UK61094 635
UK61139 635
UK61142 635
UK61213 629
UK61317 635
UK62150 629
UK62201 628
UK62326 628
UK62328 628
UK63087 629
UK63215 629
UK63366 629
UK63457 629
UK63818 629
UK63820 629
UK63890 630
UK63971 629
UK64370 629
UK64389 630
UK64423 47, 630
UK64424 630
UK64588 629
UK64597 64, 630
UK65205 629
UK65253 630
UK65325 635
UK65379 629
UK65385 630
UK65399 635
UK65412 635
UK65632 630
UK65637 630
UK65750 630
UK65859 629
UK65920 631
UK65924 635
UK65951 630
UK66046 630
UK66087 629
UK66327 630
UK66374 631
UK66376 630
UK66379 630
UK66476 630

Archived

UK66610 629
UK66964 631
UK67132 630
UK67267 628
UK67512 630
UK67578 630
UK67634 631
UK67637 630
UK67639 631
UK67958 630
UK68097 629
UK68098 631
UK68364 630
UK68476 628
UK68652 629
UK68659 632
UK68743 631
UK68801 632
UK69029 631
UK69030 632
UK69055 631
UK69199 632
UK69286 630
UK69377 631
UK69735 631
UK70215 631
UK70233 629
UK70302 630
UK70310 629
UK70483 632
UK70647 630
UK70844 632
UK71128 631, 633
UK71333 632
UK71412 631
UK71437 632
UK71458 631
UK71459 633
UK71467 632
UK71875 632
UK72212 632
UK72447 633
UK72590 636
UK73139 633
UK73180 631
UK73426 632
UK73478 631
UK73630 632
UK73864 633
UK74175 632
UK74381 633
UK74981 633
UK75324 633
UK76650 633
UK77490 633
UK77500 633
UK77584 632
UK77739 633
UK78208 633
UK78229 633
UK78514 633
UK78632 633
UK79243 633
UK79406 633
UK80113 633
UK81047 634
UK8112 636
UK81124 636
UK81520 634
UK82555 634
UK82633 634
UK82732 634
UK83168 634
UK83171 634
UK90325 632
UK91203 634
UK95407 634
unary arithmetic operators 150
Unicode 149, 277, 473
UNIQUE 81, 570
unique index 80, 82, 85, 215, 535, 578
  additional non-key columns 586
unit of recovery (UR) 238, 450
Universal table space
  support 74
universal table space 116, 274, 339, 558
universal table space (UTS) 115, 425, 559
universal table spaces (UTS) 68
UNLOAD 95, 181, 287, 364, 454, 565
UPDATE 86, 115, 119, 133, 231, 261, 343, 474, 569
URI 261
USAGE 182, 281, 368
use PLANMGMT 227
user data 87–89, 166, 338, 356
user DB2R53 362
user-defined function 126, 176, 268, 360, 474, 523
  registers different types 126
  special registers 176
user-defined functions
  types 126
UTC representation 172
UTF-8 128, 277
UTILITY Execution 86, 105, 428, 456, 517
UTS 115, 339, 507
UTSERIAL lock 61, 468

V

VALUE 152, 281, 542, 617
VALUES 152, 233, 265, 343, 428, 555
VARCHAR 72, 140, 233, 245, 340, 502, 567
variable 83, 114, 126, 250, 390, 618
VERSION 94, 127
Version 54, 58, 107, 111, 535
VERSION V1 140
VERSION V2 140
VERSIONING_SCHEMA 207
VERSIONING_TABLE 207
versions 3, 51, 55, 106, 115, 138, 204, 259, 316, 361, 470
virtual storage

Archive

constraint relief 51, 503
consumption 569
relief 51–52, 503
use 52
VPSIZE 76, 561

W

well-formed XML 258
WFDBSEP 59
whitespace 262
window specification 187
WITH 104, 106, 231, 340, 428, 474, 616
WLM 3, 130, 262, 312, 366, 481, 561
WLM environment 131, 523, 526
address space procs 528
minimal set 526
work file 56, 538
parallelism improvements 593
work load manager (WLM) 565
workfile 56, 273, 477, 535, 623
workfile database 57
available space 57
in-memory workfiles 57
KB table spaces 58
table space 57
workfile record 56
workfile table space 56, 59
space calculations 60
workfiles 57
Workload Manager (WLM) 130–131, 312, 481, 565

X

XA recover processing 117
XML 1, 243, 481, 535
XML and LOB streaming 243
XML column 245, 341
XML columns 259, 339, 565
XML data 136, 243, 565
XML data type 7, 136, 243, 579
XML documents 245, 566
XML index 251
XML multi-versioning 274
XML record 522
XML schema 250
XML schema repository 266
XML schema validation 262
XML support 244, 535
XML System Services 262
XML type modifier 258, 263
XMLEXISTS 250
XMLEXISTS predicate 250
XMLMODIFY function 266
xmlns 246
XMLPARSE 262
XMLQUERY 250
XMLQUERY function 250
XMLTABLE table function 244
XMLXSROBJECTID scalar function 268
XPath 244
XPath expression 245
XPath functions 258
XPATH performance enhancement 252
XPath scan 257
XSR 263

Z

z/Architecture 3–4
z/OS 1, 51, 110, 243
z/OS enhancement 572
z/OS Installation 381, 518
DB2 10 381
z/OS Security Server
identity propagation 416
V1R11 416
V1R8 416
z/OS Security Server V1R11 416
z/OS V1R10 9, 337, 473
z/OS V1R11 9, 337, 417, 473
zIIP 262, 536

Archived

Archived

Redbooks

DB2 10 for z/OS Technical Overview

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM CONFIDENTIAL

Archived

Archived

DB2 10 for z/OS Technical Overview

Explore the new system and application functions

Obtain information about expected performance improvements

Decide how and when to migrate to DB2 10

IBM DB2 Version 10.1 for z/OS (DB2 10 for z/OS or just *DB2 10* throughout this book) is the fourteenth release of DB2 for MVS. It brings improved performance and synergy with the System z hardware and more opportunities to drive business value in the following areas:

- Cost savings and compliance through optimized innovations
- Business insight innovations
- Business resiliency innovations

The DB2 10 environment is available either for brand new installations of DB2, or for migrations from DB2 9 for z/OS or from DB2 UDB for z/OS Version 8 subsystems.

This IBM Redbooks publication introduces the enhancements made available with DB2 10 for z/OS. The contents help you understand the new functions and performance enhancements, start planning for exploiting the key new capabilities, and justify the investment in installing or migrating or skip migrating to DB2 10.

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

SG24-7892-00 ISBN 0738435112

Archive