



Administration Guide



Administration Guide

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” at the end of this information.

Fourth edition (December 2008)

This edition applies to DB2 Version 9.1 for z/OS (DB2 V9.1 for z/OS), product number 5635-DB2, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

© Copyright International Business Machines Corporation 1982, 2008.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this information	xvii
Who should read this information	xvii
DB2 Utilities Suite	xvii
Terminology and citations	xviii
Accessibility features for DB2 Version 9.1 for z/OS	xviii
How to send your comments	xix
How to read syntax diagrams	xix

Part 1. Designing a database 1

Chapter 1. Database objects and relationships 3

Logical database design with the entity-relationship model	3
Modeling your data	3
Recommendations for logical data modeling	5
Practical examples of data modeling	5
Entities for different types of relationships	5
Entity attributes	7
Entity normalization	9
Logical database design with Unified Modeling Language	13
Physical database design	15
Denormalization of tables	15
Views as a way to customize what data users see	17
Indexes on table columns	18

Chapter 2. Implementing your database design 19

Implementing DB2 databases	19
Creating DB2 databases	19
Dropping DB2 databases	20
Implementing DB2 storage groups	20
Advantages of storage groups	20
Creating DB2 storage groups	22
Enabling SMS to control DB2 storage groups	22
Deferring allocation of DB2-managed data sets	22
How DB2 extends data sets	23
DB2 space allocation	24
Managing DB2 data sets with DFSMSHsm	28
Managing your own data sets	32
Defining index space storage	35
Creating EA-enabled table spaces and index spaces	36
Implementing DB2 table spaces	37
Creating a table space explicitly	37
Guidelines and recommendations for table spaces	38
Examples of table space definitions	45
Implementing DB2 tables	47
Creating base tables	47
Guidelines for table names	48
Creating temporary tables	48
Creating materialized query tables	52
Creating tables that use table-controlled partitioning	53
Creating tables that use index-controlled partitioning	56
Creating a clone table	56
Implementing DB2 views	57
Creating DB2 views	57
Guidelines for view names	59

How DB2 inserts and updates data through views	59
Dropping DB2 views	60
Implementing DB2 indexes	60
Creating DB2 indexes	60
Guidelines for defining indexes.	60
How DB2 implicitly creates an index	62
Recommendations for index page size	62
Index versions	63
Compressing indexes	64
Implementing DB2 schemas	65
Creating a schema by using the schema processor	65
Processing schema definitions	66
Loading data into DB2 tables	66
Loading data with the LOAD utility	67
Loading data by using the INSERT SQL statement	70
Loading data from DL/I	72
Implementing DB2 stored procedures.	72
Creating stored procedures	72
Deleting stored procedures	72
Implementing DB2 user-defined functions	72
Creating user-defined functions.	73
Deleting user-defined functions.	73
Estimating disk storage for user data	73
General approach to estimating storage	73
Calculating the space required for a table	75
Calculating the space required for an index.	78
Saving space with data compression	83
Compressing data	83
Calculating the space that is required for a dictionary	83

Chapter 3. Altering your database design 87

Altering DB2 databases	87
ALTER DATABASE options	87
Altering DB2 storage groups.	88
Letting SMS manage your DB2 storage groups	88
Adding or removing volumes from a DB2 storage group	88
Altering table spaces	89
Changing the logging attribute	90
Changing the space allocation for user-managed data sets	91
Dropping, recreating, or converting a table space	92
Rebalancing data in partitioned table spaces	93
Altering a page set to contain DB2-defined extents	94
Altering DB2 tables.	94
Adding a new column to a table	95
Specifying a default value when altering a column	96
Altering the data type of a column	97
Altering a table for referential integrity.	102
Adding or dropping table check constraints	105
Adding a partition	106
Altering partitions.	108
Adding XML columns	114
Altering materialized query tables	115
Altering the assignment of a validation routine	117
Altering a table for capture of changed data	118
Changing an edit procedure or a field procedure	119
Altering the subtype of a string column	119
Altering the attributes of an identity column	119
Changing data types by dropping and re-creating the table	120
Moving a table to a table space of a different page size	123
Altering DB2 views	124
Altering views by using the INSTEAD OF trigger	124

Altering DB2 indexes	125
Adding a column to an index when you add the column to a table	126
Altering how varying-length index columns are stored	126
Altering the clustering of an index	127
Dropping and redefining a DB2 index	128
Reorganizing indexes	128
Recycling index version numbers	128
Altering stored procedures	129
Altering user-defined functions	130
Altering implicitly created XML objects	130
Changing the high-level qualifier for DB2 data sets	131
Defining a new integrated catalog alias	132
Changing the qualifier for system data sets	132
Changing qualifiers for other databases and user data sets	135
Tools for moving DB2 data	138
Moving DB2 data	140
Moving a DB2 data set	141
Scenario: Moving from index-controlled to table-controlled partitioning	142
<hr/>	
Part 2. Security and auditing	145
Chapter 4. Getting started with DB2 security	147
DB2 security solutions	147
What's new in DB2 Version 9.1 security?	147
DB2 data access control	148
ID-based privilege control within DB2	149
Role-based privilege control within DB2	150
Privileges and authorities	150
Ownership-based privilege control within DB2	151
Multilevel security	151
Authorization control through exit routines	151
DB2 subsystem access control	151
Managing access requests from local applications	152
Managing access requests from remote applications	152
Data set protection	152
RACF for data protection	152
Data encryption	153
Scenario: Securing data access at Spiffy Computer	153
Determining security objectives	153
Securing manager access to employee data	154
Securing access to payroll operations and management	157
Managing access privileges of other authorities	160
Chapter 5. Managing access through authorization IDs or roles.	165
Authorization IDs and roles	166
Authorization IDs	166
Roles in a trusted context	166
Privileges and authorities	167
Explicit privileges	167
Implicit privileges through object ownership	172
Administrative authorities	173
Utility authorities for DB2 catalog and directory	179
Privileges by authorization ID and authority	180
Managing explicit privileges	185
Granting privileges to a role	186
Granting privileges to the PUBLIC ID	186
Granting privileges to remote users	187
Granting privileges through views	187
Granting privileges with the GRANT statement	188
Revoking privileges with the REVOKE statement	193

Managing implicit privileges	201
Managing implicit privileges through object ownership	202
Managing implicit privileges through plan or package ownership	204
Managing implicit privileges through routines	211
Retrieving privilege records in the DB2 catalog	224
Catalog tables with privilege records	224
Retrieving all authorization IDs or roles with granted privileges	225
Retrieving multiple grants of the same privilege.	225
Retrieving all authorization IDs or roles with the DBADM authority	226
Retrieving all IDs or roles with access to the same table	226
Retrieving all IDs or roles with access to the same routine	227
Retrieving tables or views accessible by an ID	228
Retrieving plans or packages with access to the same table	228
Retrieving privilege information through views	229
Implementing multilevel security with DB2	229
Multilevel security.	230
Mandatory access checking.	233
Implementing multilevel security at the object level	235
Implementing multilevel security with row-level granularity	236
Restricting access to the security label column	239
Managing data in a multilevel-secure environment	239
Implementing multilevel security in a distributed environment.	247

Chapter 6. Managing access through RACF 249

Establishing RACF protection for DB2	249
Defining DB2 resources to RACF.	250
Permitting RACF access	252
Protecting stored procedures	259
Protecting connection requests that use the TCP/IP protocol	261
Establishing Kerberos authentication through RACF	262
Implementing DB2 support for enterprise identity mapping	263
Configuring the z/OS LDAP server	264
Setting up RACF for the z/OS LDAP server	265
Setting up the EIM domain controller	266
Adding the SAF user mapping plug-in data set to LNKLIST	267
Managing connection requests from local applications.	267
Processing of connection requests	268
Using secondary IDs for connection requests	269
Processing of sign-on requests.	270
Using secondary IDs for sign-on requests	272
Using sample connection and sign-on exit routines for CICS transactions	273
Managing connection requests from remote applications	273
Security mechanisms for DRDA and SNA	273
Communications database for the server	274
Enabling change of user passwords	277
Authorization failure code	277
Managing inbound SNA-based connection requests	277
Managing inbound TCP/IP-based connection requests	284
Managing denial-of-service attacks	286
Managing outbound connection requests	287
Translating outbound IDs	296
Sending passwords	298

Chapter 7. Managing access through trusted contexts. 301

Trusted contexts	301
Trusted connections	301
Defining trusted contexts	302
Creating local trusted connections	303
Establishing remote trusted connections by DB2 for z/OS requesters	303
Establishing remote trusted connections to DB2 for z/OS servers	304

Switching users of a trusted connection	305
Reusing a local trusted connection through the DSN command processor and DB2I	305
Reusing a remote trusted connection by DB2 for z/OS requesters	306
Reusing a remote trusted connection through DB2 for z/OS servers	306
Reusing a local trusted connection through RRSAF	306
Reusing a local trusted connection through the SQL CONNECT statement	307
Defining external security profiles	307
Enabling users to perform actions on behalf of others	308
Performing tasks on objects for other users	308
Chapter 8. Managing access through data definition control	311
Data definition statements	311
Data definition control support	311
Registration tables	312
ART columns	312
ORT columns	313
Installing data definition control support	314
Enabling data definition control	314
Controlling data definition by application name	314
Controlling data definition by application name with exceptions	315
Controlling data definition by object name	317
Controlling data definition by object name with exceptions	318
Registering object sets	319
Disabling data definition control	320
Managing registration tables and indexes	321
Creating registration tables and indexes	321
Naming registration tables and indexes	322
Dropping registration tables and indexes	322
Creating table spaces for registration tables	322
Adding columns to registration tables	322
Updating registration tables	323
Chapter 9. Protecting data through encryption and RACF	325
Encrypting your data through DB2 built-in functions	325
Defining columns for encrypted data	325
Defining column-level encryption	326
Defining value-level encryption	327
Using predicates for encrypted data	329
Optimizing performance of encrypted data	329
Encrypting your data with Secure Socket Layer support	331
AT-TLS configuration	331
Configuring the DB2 server for SSL	332
Configuring the DB2 requester for SSL	333
Protecting data sets through RACF	334
Adding groups to control DB2 data sets	334
Creating generic profiles for data sets	334
Authorizing DB2 IDs to use data set profiles	336
Enabling DB2 IDs to create data sets	336
Chapter 10. Auditing access to DB2.	337
DB2 audit trace	337
Authorization IDs traced by auditing	338
Audit classes	338
Limitations of the audit trace	339
Auditing in a distributed data environment	340
Audit trace records	340
Audit trace reports	340
Additional sources of audit information	341
Determining active security measures	342
Using DB2 audit trace and trace records	342

Starting the audit trace	342
Stopping the audit trace	343
Auditing specific tables	343
Auditing specific IDs or roles	344
Determining ID privileges and authorities	345
Collecting audit trace records	345
Formatting audit trace records	345
Managing data accuracy and consistency	345
Ensuring data presence and uniqueness	346
Protecting data integrity	346
Tracking data changes	347
Checking for lost and incomplete transactions	347
Ensuring data consistency	348
Using referential integrity for data consistency	348
Using locks for data consistency	348
Checking data consistency	349
<hr/>	
Part 3. Operation and recovery	353
Chapter 11. DB2 basic operational concepts	355
Recommendations for entering commands	355
DB2 operator commands	355
Where DB2 commands are entered	358
Where command responses go	360
Authorities for DB2 commands	361
DB2 message identifiers	362
Unsolicited DB2 messages	363
Operational control options	363
Chapter 12. Starting and stopping DB2	365
Starting DB2	365
Messages at start	365
Options at start	366
Restricting access to data	366
Ending the wait state at start	367
Restart options after an abend	367
Stopping DB2	368
Chapter 13. Submitting work to DB2	369
Submitting work by using DB2I	369
Running TSO application programs	369
DSN subcommands for TSO environments	370
Sources that DB2 uses to find authorization access by the application program	371
Running IMS application programs	371
Running CICS application programs	372
Running batch application programs	372
Running application programs using CAF	373
Running application programs using RRSAF	373
Chapter 14. Scheduling administrative tasks	375
Interacting with the administrative task scheduler	375
Adding a task	375
Listing scheduled tasks	387
Listing the last execution status of scheduled tasks	391
Removing a scheduled task	395
Manually starting the administrative task scheduler	397
Manually stopping the administrative task scheduler	398
Synchronization between administrative task schedulers in a data sharing environment	398
Troubleshooting the administrative task scheduler	399

	Architecture of the administrative task scheduler	402
	The lifecycle of the administrative task scheduler	403
	Scheduler task lists	404
	Architecture of the administrative task scheduler in a data sharing environment	405
	Security guidelines for the administrative task scheduler	406
	User roles in the administrative task scheduler	407
	Protection of the interface of the administrative task scheduler	407
	Protection of the resources of the administrative task scheduler	408
	Secure execution of tasks in the administrative task scheduler	408
	Execution of scheduled tasks in the administrative task scheduler	409
	Multi-threading in the administrative task scheduler	409
	Scheduled execution of a stored procedure	410
	How the administrative task scheduler works with Unicode	411
	Scheduled execution of a JCL job	411
	Execution of scheduled tasks in a data sharing environment	412

Chapter 15. Monitoring and controlling DB2 and its connections 413

	Controlling DB2 databases and buffer pools	413
	Starting databases	413
	Monitoring databases	415
	Obtaining information about application programs	417
	Obtaining information about and handling pages in error	419
	Using the STOP DATABASE command to make objects available	422
	Altering buffer pools	423
	Monitoring buffer pools	423
	Controlling user-defined functions	424
	Starting user-defined functions	424
	Monitoring user-defined functions	425
	Stopping user-defined functions	425
	Controlling DB2 utilities	426
	Starting online utilities	426
	Monitoring and changing online utilities	426
	Controlling DB2 stand-alone utilities	427
	Controlling the IRLM	428
	z/OS commands that operate on IRLM	429
	Starting the IRLM	430
	Stopping the IRLM	430
	Monitoring threads	431
	Types of threads	431
	Output of the DISPLAY THREAD command	432
	Displaying information about threads	432
	Monitoring all DBMSs in a transaction	436
	Controlling connections	438
	Controlling TSO connections	439
	Controlling CICS connections	442
	Controlling IMS connections	447
	Controlling RRS connections	456
	Controlling connections to remote systems	460
	Controlling traces	474
	Types of DB2 traces	474
	Diagnostic traces for attachment facilities	474
	Controlling the DB2 trace	475
	Diagnostic trace for the IRLM	476
	Controlling the resource limit facility (governor)	476
	Changing subsystem parameter values	477

Chapter 16. Managing the log and the bootstrap data set 479

	How database changes are made	479
	Units of recovery and points of consistency	479
	How DB2 rolls back work	480

How the initial DB2 logging environment is established	481
How DB2 creates log records	481
How DB2 writes the active log	481
How DB2 writes (offloads) the archive log	482
How DB2 retrieves log records	487
Managing the log	488
Quiescing activity before offloading	488
Archiving the log	489
Dynamically changing the checkpoint frequency.	490
Setting limits for archive log tape units.	491
Monitoring the system checkpoint	491
Displaying log information	491
Resetting the log RBA	492
Log RBA range	492
Resetting the log RBA value in a data sharing environment	493
Resetting the log RBA value in a non-data sharing environment	493
Canceling and restarting an offload	494
Displaying the status of an offload	494
Discarding archive log records.	494
Locating archive log data sets	495
Management of the bootstrap data set	498
Restoring dual mode BSDS.	498
BSDS copies with archive log data sets	499
Recommendations for changing the BSDS log inventory	499

Chapter 17. Restarting DB2 after termination 501

Methods of restarting.	501
Types of termination	501
Normal restart and recovery	502
Automatic restart	507
Restart in a data sharing environment	507
Restart implications for table spaces that are not logged	507
Conditional restart	508
Terminating DB2 normally	509
Restarting automatically.	509
Deferring restart processing	510
Deferral of restart	511
Performing conditional restart	511
Options for recovery operations after conditional restart	512
Conditional restart records	512
Resolving postponed units of recovery	512
RECOVER POSTPONED command	513
Recovering from an error during RECOVER POSTPONED processing	514

Chapter 18. Maintaining consistency across multiple systems 515

Multiple system consistency	515
Two-phase commit process	515
Commit coordinator and multiple participants	517
Illustration of multi-site update	518
Termination for multiple systems.	519
Consistency after termination or failure.	520
Normal restart and recovery for multiple systems	521
Multiple-system restart with conditions.	522
Heuristic decisions about whether to commit or abort an indoubt thread	522
Resolving indoubt units of recovery	522
Resolution of IMS indoubt units of recovery	523
Resolution of CICS indoubt units of recovery.	524
Resolution of RRS indoubt units of recovery	524
Resolving WebSphere Application Server indoubt units of recovery	525
Resolving remote DBMS indoubt units of recovery	527

Determining the coordinator's commit or abort decision	528
Recovering indoubt threads	528
Resetting the status of an indoubt thread	529

Chapter 19. Backing up and recovering your data. 531

Plans for backup and recovery	531
Plans for recovery of distributed data	531
Plans for extended recovery facility toleration	532
Plans for recovery of indexes	533
Preparation for recovery: a scenario	533
Events that occur during recovery	534
Tips for maximizing data availability during backup and recovery	539
Where to find recovery information	542
How to report recovery information	543
How to discard SYSCOPY and SYSLGRNX records	544
Preparations for disaster recovery	545
Recommendations for more effective recovery from inconsistency	547
How to recover multiple objects in parallel	549
Automatic fast log apply during RECOVER	550
Recovery of page sets and data sets	550
Recovery of data to a prior point of consistency	556
Preparing to recover an entire DB2 subsystem to a prior point in time using image copies or object-level backups	565
Creating essential disaster recovery elements	565
Resolving problems with a user-defined work file data set	567
Resolving problems with DB2-managed work file data sets	567
Recovering error ranges for a work file table space	568
Recovering after a conditional restart of DB2	568
Regenerating missing identity column values	569
Recovering a table space and all of its indexes	570
Removing various pending states from LOB and XML table spaces	574
Restoring data by using DSN1COPY	575
Backing up and restoring data with non-DB2 dump and restore	575
Recovering accidentally dropped objects	575
Recovering your DB2 system to a given point in time by using the RESTORE SYSTEM utility	581
Recovering by using DB2 restart recovery	590
Recovering by using FlashCopy backups	591
Making catalog definitions consistent with your data after recovery to a prior point in time	591
Performing remote site recovery from a disaster at a local site	594
Backup and recovery involving clone tables	595
Data restore of an entire system	595

Chapter 20. Recovering from different DB2 for z/OS problems 597

Recovering from IRLM failure	597
Recovering from z/OS or power failure	597
Recovering from disk failure	598
Recovering from application errors	600
Backing out incorrect application changes (with a quiesce point)	600
Backing out incorrect application changes (without a quiesce point)	601
Recovering from IMS-related failures	601
Recovering from IMS control region failure	602
Recovering from IMS indoubt units of recovery	602
Recovering from IMS application failure	605
Recovering from a DB2 failure in an IMS environment	605
Recovering from CICS-related failure	606
Recovering from CICS application failures	606
Recovering DB2 when CICS is not operational	607
Recovering DB2 when the CICS attachment facility cannot connect to DB2	608
Recovering CICS indoubt units of recovery	608
Recovering from CICS attachment facility failure	611

Recovering from subsystem termination	611
Recovering from temporary resource failure	612
Recovering from active log failures	613
Recovering from being out of space in active logs	613
Recovering from a write I/O error on an active log data set	614
Recovering from a loss of dual active logging	615
Recovering from I/O errors while reading the active log	616
Recovering from archive log failures.	618
Recovering from allocation problems with the archive log	618
Recovering from write I/O errors during archive log offload	618
Recovering from read I/O errors on an archive data set during recovery	619
Recovering from insufficient disk space for offload processing	620
Recovering from BSDS failures	621
Recovering from an I/O error on the BSDS	621
Recovering from an error that occurs while opening the BSDS	622
Recovering from unequal timestamps on BSDSs	622
Recovering the BSDS from a backup copy	623
Recovering from BSDS or log failures during restart	626
Recovering from failure during log initialization or current status rebuild	628
Recovering from a failure during forward log recovery	639
Recovering from a failure during backward log recovery	645
Recovering from a failure during a log RBA read request.	647
Recovering from unresolvable BSDS or log data set problem during restart.	648
Recovering from a failure resulting from total or excessive loss of log data	651
Resolving inconsistencies resulting from a conditional restart	655
Recovering from DB2 database failure	660
Recovering a DB2 subsystem to a prior point in time	661
Recovering from a down-level page set problem.	662
Recovering from a problem with invalid LOBs	664
Recovering from table space input-output errors	665
Recovering from DB2 catalog or directory I/O errors	666
Recovering from integrated catalog facility failure	667
Recovering VSAM volume data sets that are out of space or destroyed	667
Recovering from out-of-disk-space or extent-limit problems	668
Recovering from referential constraint violation	672
Recovering from distributed data facility failure	673
Recovering from conversation failure	673
Recovering from communications database failure	674
Recovering from database access thread failure	675
Recovering from VTAM failure	676
Recovering from TCP/IP failure	676
Recovering from remote logical unit failure	677
Recovering from an indefinite wait condition.	677
Recovering database access threads after security failure	678
Performing remote-site disaster recovery	679
Recovering from a disaster by using system-level backups	679
Restoring data from image copies and archive logs.	679
Recovering from disasters by using a tracker site	694
Using data mirroring for disaster recovery	702
Scenarios for resolving problems with indoubt threads	708
Scenario: Recovering from communication failure	710
Scenario: Making a heuristic decision about whether to commit or abort an indoubt thread	711
Scenario: Recovering from an IMS outage that results in an IMS cold start	713
Scenario: Recovering from a DB2 outage at a requester that results in a DB2 cold start	715
Scenario: What happens when the wrong DB2 subsystem is cold started	719
Scenario: Correcting damage from an incorrect heuristic decision about an indoubt thread	720
Chapter 21. Reading log records	723
Contents of the log	723
Unit of recovery log records	723
Checkpoint log records	727

Database page set control records	728
Other exception information	728
The physical structure of the log	728
Physical and logical log records	728
The log record header	729
The log control interval definition (LCID)	730
Log record type codes	732
Log record subtype codes	733
Interpreting data change log records.	735
Reading log records with IFI	735
Reading log records into a buffer.	735
Reading specific log records (IFCID 0129)	736
Reading complete log data (IFCID 0306)	737
Reading log records with OPEN, GET, and CLOSE	739
JCL DD statements for DB2 stand-alone log services	740
Data sharing members that participate in a read.	742
Registers and return codes	743
Stand-alone log OPEN request.	744
Stand-alone log GET request	745
Stand-alone log CLOSE request	747
Sample application that uses stand-alone log services	748
Reading log records with the log capture exit routine	749

Part 4. Appendixes 751

Appendix A. Writing exit routines. 753

Connection routines and sign-on routines	753
Specifying connection and sign-on routines	754
Sample connection and sign-on routines	754
When connection and sign-on routines are taken	755
EXPL for connection and sign-on routines	756
Exit parameter list for connection and sign-on routines	756
Authorization ID parameter list for connection and sign-on routines	757
Input values for connection routines.	758
Input values for sign-on routines	759
Expected output for connection and sign-on routines	759
Processing in the sample connection and sign-on routines	760
Performance considerations for connection and sign-on routines	761
Debugging connection and sign-on routines	762
Session variables in connection and sign-on routines	763
Access control authorization exit routine	764
Specifying the access control authorization routine	766
The default access control authorization routine	766
When the access control authorization routine is taken	766
Considerations for the access control authorization routine	767
Parameter list for the access control authorization routine	770
Expected output for the access control authorization routine.	778
Debugging the access control authorization routine.	780
Determining whether the access control authorization routine is active	781
Edit routines	781
Specifying the edit routine	782
When the edit routine is taken	782
Parameter lists for the edit routine	782
Processing requirements for edit routines	783
Incomplete rows and edit routines	783
Expected output for edit routines.	784
Validation routines	785
Specifying the validation routine	785
When the validation routine is taken	785
Parameter lists for the validation routine	786

Processing requirements for validation routines	786
Incomplete rows and validation routines	787
Expected output for validation routines	787
Date and time routines	788
Specifying the date and time routine	789
When date and time routines are taken.	789
Parameter lists for date and time routines	790
Expected output for date and time routines	791
Conversion procedures	792
Specifying a conversion procedure	792
When conversion procedures are taken.	793
Parameter lists for conversion procedures	793
Expected output for conversion procedures	793
Field procedures	795
Field definition for field procedures	796
Specifying the field procedure.	796
When field procedures are taken	797
Control blocks for execution of field procedures.	798
Field-definition (function code 8)	801
Field-encoding (function code 0)	803
Field-decoding (function code 4)	805
Log capture routines	807
Specifying the log capture routine	807
When log capture routines are taken	808
Parameter lists for log capture routines.	808
Routines for dynamic plan selection in CICS	809
Routine for CICS transaction invocation stored procedure	810
General guidelines for writing exit routines	810
Coding rules for exit routines	810
Modifying exit routines	811
Execution environment for exit routines	811
Registers at invocation for exit routines.	811
Parameter lists for exit routines	812
Row formats for edit and validation routines.	813
Column boundaries for edit and validation routines	813
Null values for edit procedures, field procedures, and validation routines	813
Fixed-length rows for edit and validation routines	814
Varying-length rows for edit and validation routines	814
Varying-length rows with nulls for edit and validation routines	815
EDITPROCs and VALIDPROCs for handling basic and reordered row formats	815
Converting basic row format table spaces with edit and validation routines to reordered row format	816
Dates, times, and timestamps for edit and validation routines	816
Parameter list for row format descriptions.	817
DB2 codes for numeric data in edit and validation routines	818
RACF access control module	820
Appendix B. Stored procedures for administration	821
DSNACICS stored procedure	821
The DSNACICX user exit routine.	826
DSNLEUSR stored procedure	829
DSNAIMS stored procedure	832
DSNAIMS2 stored procedure	836
ADMIN_COMMAND_DB2 stored procedure.	841
ADMIN_COMMAND_DSN stored procedure	850
ADMIN_COMMAND_UNIX stored procedure	852
ADMIN_DS_BROWSE stored procedure	856
ADMIN_DS_DELETE stored procedure.	859
ADMIN_DS_LIST stored procedure	862
ADMIN_DS_RENAME stored procedure	868
ADMIN_DS_SEARCH stored procedure	871
ADMIN_DS_WRITE stored procedure	873

ADMIN_INFO_HOST stored procedure	877
ADMIN_INFO_SSID stored procedure	881
ADMIN_JOB_CANCEL stored procedure	882
ADMIN_JOB_FETCH stored procedure.	885
ADMIN_JOB_QUERY stored procedure	888
ADMIN_JOB_SUBMIT stored procedure	892
ADMIN_UTL_SCHEDULE stored procedure	895
ADMIN_UTL_SORT stored procedure	904
Common SQL API stored procedures	911
GET_CONFIG stored procedure	912
GET_MESSAGE stored procedure	922
GET_SYSTEM_INFO stored procedure	932
XPath expressions for filtering output	948
Information resources for DB2 for z/OS and related products	949
How to obtain DB2 information.	955
How to use the DB2 library	959
Notices	963
Programming Interface Information	965
General-use Programming Interface and Associated Guidance Information	965
Product-sensitive Programming Interface and Associated Guidance Information	965
Trademarks	965
Glossary	967
Index	1011

About this information

This information provides guidance information that you can use to perform a variety of administrative tasks with DB2® for z/OS® (DB2).

Information about DB2 concepts, which was present in previous editions of this book, is now provided in *Introduction to DB2 for z/OS*.

This information assumes that your DB2 subsystem is running in Version 9.1 new-function mode. Generally, new functions that are described, including changes to existing functions, statements, and limits, are available only in new-function mode. Two exceptions to this general statement are new and changed utilities and optimization enhancements, which are also available in conversion mode unless stated otherwise.

Who should read this information

This information is primarily intended for system and database administrators. It assumes that the user is familiar with:

- The basic concepts and facilities of DB2
- Time Sharing Option (TSO) and Interactive System Productivity Facility (ISPF)
- The basic concepts of Structured Query Language (SQL)
- The basic concepts of Customer Information Control System (CICS®)
- The basic concepts of Information Management System (IMS™)
- How to define and allocate z/OS data sets using job control language (JCL).

Certain tasks require additional skills, such as knowledge of Transmission Control Protocol/Internet Protocol (TCP/IP) or Virtual Telecommunications Access Method (VTAM®) to set up communication between DB2 subsystems, or knowledge of the IBM® System Modification Program (SMP/E) to install IBM licensed programs.

DB2 Utilities Suite

Important: In this version of DB2 for z/OS, the DB2 Utilities Suite is available as an optional product. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

The DB2 Utilities Suite is designed to work with the DFSORT™ program, which you are licensed to use in support of the DB2 utilities even if you do not otherwise license DFSORT for general use. If your primary sort product is not DFSORT, consider the following informational APARs mandatory reading:

- II14047/II14213: USE OF DFSORT BY DB2 UTILITIES
- II13495: HOW DFSORT TAKES ADVANTAGE OF 64-BIT REAL ARCHITECTURE

These informational APARs are periodically updated.

Related information

DB2 utilities packaging (Utility Guide)

Terminology and citations

In this information, DB2 Version 9.1 for z/OS is referred to as "DB2 for z/OS." In cases where the context makes the meaning clear, DB2 for z/OS is referred to as "DB2." When this information refers to titles of DB2 for z/OS books, a short title is used. (For example, "See *DB2 SQL Reference*" is a citation to *IBM DB2 Version 9.1 for z/OS SQL Reference*.)

When referring to a DB2 product other than DB2 for z/OS, this information uses the product's full name to avoid ambiguity.

The following terms are used as indicated:

DB2 Represents either the DB2 licensed program or a particular DB2 subsystem.

OMEGAMON®

Refers to any of the following products:

- IBM Tivoli® OMEGAMON XE for DB2 Performance Expert on z/OS
- IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS
- IBM DB2 Performance Expert for Multiplatforms and Workgroups
- IBM DB2 Buffer Pool Analyzer for z/OS

C, C++, and C language

Represent the C or C++ programming language.

CICS Represents CICS Transaction Server for z/OS.

IMS Represents the IMS Database Manager or IMS Transaction Manager.

MVS™ Represents the MVS element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

RACF®

Represents the functions that are provided by the RACF component of the z/OS Security Server.

Accessibility features for DB2 Version 9.1 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including DB2 Version 9.1 for z/OS. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size

Tip: The Information Management Software for z/OS Solutions Information Center (which includes information for DB2 Version 9.1 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

You can access DB2 Version 9.1 for z/OS ISPF panel functions by using a keyboard or keyboard shortcut keys.

For information about navigating the DB2 Version 9.1 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Related accessibility information

Online documentation for DB2 Version 9.1 for z/OS is available in the Information Management Software for z/OS Solutions Information Center, which is available at the following Web site: <http://publib.boulder.ibm.com/infocenter/dzichelp>

IBM and accessibility

See the *IBM Accessibility Center* at <http://www.ibm.com/able> for more information about the commitment that IBM has to accessibility.

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 for z/OS documentation. You can use the following methods to provide comments:

- Send your comments by e-mail to db2zinfo@us.ibm.com and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title or a help topic title).
- You can send comments from the Web. Visit the DB2 for z/OS - Technical Resources Web site at:

<http://www.ibm.com/support/docview.wss?&uid=swg27011656>

This Web site has an online reader comment form that you can use to send comments.

- You can also send comments by using the feedback link at the footer of each page in the Information Management Software for z/OS Solutions Information Center at <http://publib.boulder.ibm.com/infocenter/db2zhelp>.

How to read syntax diagrams

Certain conventions apply to the syntax diagrams that are used in IBM documentation.

Apply the following rules when reading the syntax diagrams that are used in DB2 for z/OS documentation:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
 - The ►— symbol indicates the beginning of a statement.
 - The —→ symbol indicates that the statement syntax is continued on the next line.
 - The ►— symbol indicates that a statement is continued from the previous line.
 - The —▶ symbol indicates the end of a statement.
- Required items appear on the horizontal line (the main path).



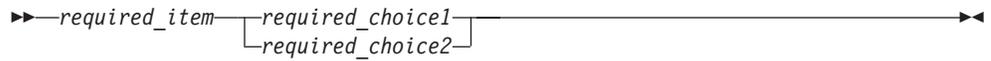
- Optional items appear below the main path.



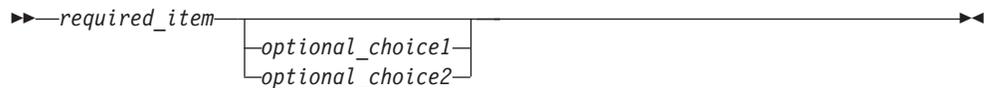
If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.



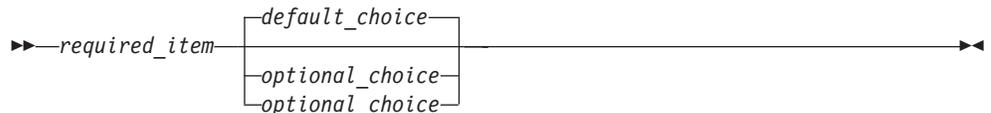
- If you can choose from two or more items, they appear vertically, in a stack. If you *must* choose one of the items, one item of the stack appears on the main path.



If choosing one of the items is optional, the entire stack appears below the main path.



If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.



If the repeat arrow contains a comma, you must separate repeated items with a comma.



A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.

|
|
|

►►—*required_item*—| fragment-name |—————►►

fragment-name:

|—*required_item*—|—————|
 └—*optional_name*—┘

- With the exception of XPath keywords, keywords appear in uppercase (for example, FROM). Keywords must be spelled exactly as shown. XPath keywords are defined as lowercase names, and must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Part 1. Designing a database

Chapter 1. Database objects and relationships

The two general tasks necessary to design a database are logical data modeling and physical data modeling.

In logical data modeling, you design a model of the data without paying attention to specific functions and capabilities of the DBMS that will store the data. In fact, you could even build a logical data model without knowing which DBMS you will use.

Physical data modeling begins when you move closer to a physical implementation. The primary purpose of the physical design stage is to optimize performance while ensuring the integrity of the data.

Logical database design with the entity-relationship model

Before you implement a database, you should plan or design the database so that it satisfies all requirements.

Designing and implementing a successful database, one that satisfies the needs of an organization, requires a logical data model. Logical data modeling is the process of documenting the comprehensive business information requirements in an accurate and consistent format. Analysts who do data modeling define the data items and the business rules that affect those data items. The process of data modeling acknowledges that business data is a vital asset that the organization needs to understand and carefully manage. This section contains information that was adapted from Handbook of Relational Database Design.

Consider the following business facts that a manufacturing company needs to represent in its data model:

- Customers purchase products
- Products consist of parts
- Suppliers manufacture parts
- Warehouses store parts
- Transportation vehicles move the parts from suppliers to warehouses and then to manufacturers

These are all business facts that a manufacturing company's logical data model needs to include. Many people inside and outside the company rely on information that is based on these facts. Many reports include data about these facts.

Any business, not just manufacturing companies, can benefit from the task of data modeling. Database systems that supply information to decision makers, customers, suppliers, and others are more successful if their foundation is a sound data model.

Modeling your data

Data analysts can perform the task of data modeling in a variety of ways.

Many data analysts follow these steps:

1. Build critical user views.
 - a. Carefully examining a single business activity or function.
 - b. Develop a user view, which is the model or representation of critical information that the business activity requires.

This initial stage of the data modeling process is highly interactive. Because data analysts cannot fully understand all areas of the business that they are modeling, they work closely with the actual users. Working together, analysts and users define the major entities (significant objects of interest) and determine the general relationships between these entities.

In a later stage, the analyst combines each individual user view with all the other user views into a consolidated logical data model.
2. Add keys to user views

Key business rules affect insert, update, and delete operations on the data. For example, a business rule might require that each customer entity have at least one unique identifier. Any attempt to insert or update a customer identifier that matches another customer identifier is not valid. In a data model, a unique identifier is called a primary key.
3. Add detail to user views and validate them.
 - a. Add other descriptive details that are less vital.
 - b. Associate these descriptive details, called attributes, to the entities.

For example, a customer entity probably has an associated phone number. The phone number is a non-key attribute of the customer entity.
 - c. Validate all the user views

To validate the views, analysts use the normalization process and process models. Process models document the details of how the business will use the data.
4. Determine additional business rules that affect attributes.
 - a. Clarify the data-driven business rules.

Data-driven business rules are constraints on particular data values. These constraints need to be true, regardless of any particular processing requirements.

The advantage to defining data-driven business rules during the data design stage, rather than during application design is that programmers of many applications don't need to write code to enforce these business rules. For example, Assume that a business rule requires that a customer entity have a phone number, an address, or both. If this rule doesn't apply to the data itself, programmers must develop, test, and maintain applications that verify the existence of one of these attributes. Data-driven business requirements have a direct relationship with the data, thereby relieving programmers from extra work.
5. Integrate user views.
 - a. Combine into a consolidated logical data model the newly created different user views.
 - b. Integrate other data models that already exist in the organization with the new consolidated logical data model.

At this stage, analysts also strive to make their data model flexible so that it can support the current business environment and possible future changes. For example, assume that a retail company operates in a single country and that business plans include expansion to other countries. Armed with knowledge of these plans, analysts can build the model so that it is flexible enough to support expansion into other countries.

Recommendations for logical data modeling

To build sound data models, analysts follow a well-planned methodology.

Follow these recommendation for building quality data models:

- Work interactively with the users as much as possible.
- Use diagrams to represent as much of the logical data model as possible.
- Build a data dictionary to supplement the logical data model diagrams.

A *data dictionary* is a repository of information about an organization's application programs, databases, logical data models, users, and authorizations. A data dictionary can be manual or automated.

Practical examples of data modeling

Use this scenario of real-life data modeling to better understand the key activities necessary for creating sound data models.

You begin by defining your entities, the significant objects of interest. Entities are the things about which you want to store information. For example, you might want to define an entity, called EMPLOYEE, for employees because you need to store information about everyone who works for your organization. You might also define an entity, called DEPARTMENT, for departments.

Next, you define primary keys for your entities. A primary key is a unique identifier for an entity. In the case of the EMPLOYEE entity, you probably need to store a large amount of information. However, most of this information (such as gender, birth date, address, and hire date) would not be a good choice for the primary key. In this case, you could choose a unique employee ID or number (EMPLOYEE_NUMBER) as the primary key. In the case of the DEPARTMENT entity, you could use a unique department number (DEPARTMENT_NUMBER) as the primary key.

After you have decided on the entities and their primary keys, you can define the relationships that exist between the entities. The relationships are based on the primary keys. If you have an entity for EMPLOYEE and another entity for DEPARTMENT, the relationship that exists is that employees are assigned to departments. You can read more about this topic in the next section.

After defining the entities, their primary keys, and their relationships, you can define additional attributes for the entities. In the case of the EMPLOYEE entity, you might define the following additional attributes:

- Birth date
- Hire date
- Home address
- Office phone number
- Gender
- Resume

Lastly, you normalize the data.

Entities for different types of relationships

In a relational database, you can express several types of relationships.

Consider the possible relationships between employees and departments. If a given employee can work in only one department, this relationship is one-to-one for employees. One department usually has many employees; this relationship is one-to-many for departments. Relationships can be one-to-many, many-to-one, one-to-one, or many-to-many.

Subsections:

- “One-to-one relationships”
- “One-to-many and many-to-one relationships”
- “Many-to-many relationships” on page 7
- “Business rules for relationships” on page 7

The type of a given relationship can vary, depending on the specific environment. If employees of a company belong to several departments, the relationship between employees and departments is many-to-many.

You need to define separate entities for different types of relationships. When modeling relationships, you can use diagram conventions to depict relationships by using different styles of lines to connect the entities.

One-to-one relationships

When you are doing logical database design, one-to-one relationships are bidirectional relationships, which means that they are single-valued in both directions. For example, an employee has a single resume; each resume belongs to only one person. The previous figure illustrates that a one-to-one relationship exists between the two entities. In this case, the relationship reflects the rules that an employee can have only one resume and that a resume can belong to only one employee.

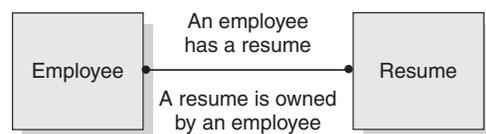


Figure 1. Assigning one-to-one facts to an entity

One-to-many and many-to-one relationships

A one-to-many relationship occurs when one entity has a multivalued relationship with another entity. In the following figure, you see that a one-to-many relationship exists between the two entities—employee and department. This figure reinforces the business rules that a department can have many employees, but that each individual employee can work for only one department.

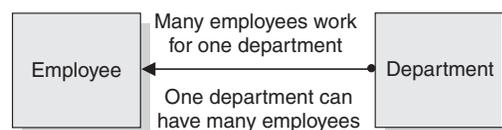


Figure 2. Assigning many-to-one facts to an entity

Many-to-many relationships

A many-to-many relationship is a relationship that is multivalued in both directions. The following figure illustrates this kind of relationship. An employee can work on more than one project, and a project can have more than one employee assigned.

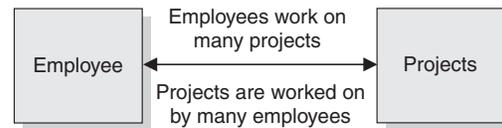


Figure 3. Assigning many-to-many facts to an entity

Business rules for relationships

Whether a given relationship is one-to-one, one-to-many, many-to-one, or many-to-many, your relationships need to make good business sense. Therefore, database designers and data analysts can be more effective when they have a good understanding of the business. If they understand the data, the applications, and the business rules, they can succeed in building a sound database design.

When you define relationships, you have a big influence on how smoothly your business runs. If you don't do a good job at this task, your database and associated applications are likely to have many problems, some of which may not manifest themselves for years.

Entity attributes

When you define attributes for the entities, you generally work with the data administrator (DA) to decide on names, data types, and appropriate values for the attributes.

Attribute names

Most organizations have naming conventions. In addition to following these conventions, Data analysts also base attribute definitions on class words.

A *class word* is a single word that indicates the nature of the data that the attribute represents.

The class word NUMBER indicates an attribute that identifies the number of an entity. Attribute names that identify the numbers of entities should therefore include the class word of NUMBER. Some examples are EMPLOYEE_NUMBER, PROJECT_NUMBER, and DEPARTMENT_NUMBER.

When an organization does not have well-defined guidelines for attribute names, the DAs try to determine how the database designers have historically named attributes. Problems occur when multiple individuals are inventing their own naming schemes without consulting one another.

Data types of attributes

You must specify a data type for each attribute. Most organizations have well-defined guidelines for using the different data types.

Example: You might use the following data types for attributes of the EMPLOYEE entity:

- EMPLOYEE_NUMBER: CHAR(6)
- EMPLOYEE_LAST_NAME: VARCHAR(15)
- EMPLOYEE_HIRE_DATE: DATE
- EMPLOYEE_SALARY_AMOUNT: DECIMAL(9,2)

The data types that you choose are business definitions of the data type. During physical database design you might need to change data type definitions or use a subset of these data types. The database or the host language might not support all of these definitions, or you might make a different choice for performance reasons.

For example, you might need to represent monetary amounts, but DB2 and many host languages do not have a data type MONEY. In the United States, a natural choice for the SQL data type in this situation is DECIMAL(10,2) to represent dollars. But you might also consider the INTEGER data type for fast, efficient performance.

For more information on data types of attributes, see the “CREATE TABLE” topic in the *DB2 SQL Reference*.

Appropriate values for attributes

When you design a database, you need to decide what values are acceptable for the various attributes of an entity.

For example, you would not want to allow numeric data in an attribute for a person’s name. The data types that you choose limit the values that apply to a given attribute, but you can also use other mechanisms. These other mechanisms are domains, null values, and default values.

Subsections:

- “Domain”
- “Null values” on page 9
- “Default values” on page 9

Domain

A *domain* describes the conditions that an attribute value must meet to be a valid value. Sometimes the domain identifies a range of valid values. By defining the domain for a particular attribute, you apply business rules to ensure that the data will make sense.

Example 1: A domain might state that a phone number attribute must be a 10-digit value that contains only numbers. You would not want the phone number to be incomplete, nor would you want it to contain alphabetic or special characters and thereby be invalid. You could choose to use either a numeric data type or a character data type. However, the domain states the business rule that the value must be a 10-digit value that consists of numbers.

Example 2: A domain might state that a month attribute must be a 2-digit value from 01 to 12. Again, you could choose to use datetime, character, or numeric data types for this value, but the domain demands that the value must be in the range of 01 through 12. In this case, incorporating the month into a datetime data type is probably the best choice. This decision should be reviewed again during physical database design.

Null values

When you are designing attributes for your entities, you will sometimes find that an attribute does not have a value for every instance of the entity. For example, you might want an attribute for a person's middle name, but you can't require a value because some people have no middle name. For these occasions, you can define the attribute so that it can contain null values.

A *null value* is a special indicator that represents the absence of a value. The value can be absent because it is unknown, not yet supplied, or nonexistent. The DBMS treats the null value as an actual value, not as a zero value, a blank, or an empty string.

Just as some attributes should be allowed to contain null values, other attributes should not contain null values.

Example: For the EMPLOYEE entity, you might not want to allow the attribute EMPLOYEE_LAST_NAME to contain a null value.

Default values

In some cases, you may not want a given attribute to contain a null value, but you don't want to require that the user or program always provide a value. In this case, a default value might be appropriate.

A *default value* is a value that applies to an attribute if no other valid value is available.

Example: Assume that you don't want the EMPLOYEE_HIRE_DATE attribute to contain null values and that you don't want to require users to provide this data. If data about new employees is generally added to the database on the employee's first day of employment, you could define a default value of the current date.

Entity normalization

After you define entities and decide on attributes for the entities, you normalize entities to avoid redundancy.

An entity is normalized if it meets a set of constraints for a particular normal form, which this section describes. Normalization helps you avoid redundancies and inconsistencies in your data. This section summarizes rules for first, second, third, and fourth normal forms of entities, and it describes reasons why you should or shouldn't follow these rules.

Subsections:

- "First normal form" on page 10
- "Second normal form" on page 10
- "Third normal form" on page 11
- "Fourth normal form" on page 13

The rules for normal form are cumulative. In other words, for an entity to satisfy the rules of second normal form, it also must satisfy the rules of first normal form. An entity that satisfies the rules of fourth normal form also satisfies the rules of first, second, and third normal form.

In this section, you will see many references to the word *instance*. In the context of logical data modeling, an instance is one particular occurrence. An instance of an entity is a set of data values for all of the attributes that correspond to that entity.

Example: The following figure shows one instance of the EMPLOYEE entity.

Employee

EMPLOYEE _NUMBER	EMPLOYEE _FIRST _NAME	EMPLOYEE _LAST _NAME	DEPARTMENT _NUMBER	EMPLOYEE _HIRE_DATE
000010	CHRISTINE	HAAS	A00	1975-01-01

Figure 4. The EMPLOYEE entity

First normal form

A relational entity satisfies the requirement of first normal form if every instance of an entity contains only one value, never multiple repeating attributes. Repeating attributes, often called a repeating group, are different attributes that are inherently the same. In an entity that satisfies the requirement of first normal form, each attribute is independent and unique in its meaning and its name.

Example: Assume that an entity contains the following attributes:

EMPLOYEE_NUMBER
 JANUARY_SALARY_AMOUNT
 FEBRUARY_SALARY_AMOUNT
 MARCH_SALARY_AMOUNT

This situation violates the requirement of first normal form, because JANUARY_SALARY_AMOUNT, FEBRUARY_SALARY_AMOUNT, and MARCH_SALARY_AMOUNT are essentially the same attribute, EMPLOYEE_MONTHLY_SALARY_AMOUNT.

Second normal form

An entity is in second normal form if each attribute that is not in the primary key provides a fact that depends on the entire key. A violation of the second normal form occurs when a nonprimary key attribute is a fact about a subset of a composite key.

Example: An inventory entity records quantities of specific parts that are stored at particular warehouses. The following figure shows the attributes of the inventory entity.

----- Key -----			
PART	WAREHOUSE	QUANTITY	WAREHOUSE_ADDRESS

Figure 5. Entity in violation of the second normal form

Here, the primary key consists of the PART and the WAREHOUSE attributes together. Because the attribute WAREHOUSE_ADDRESS depends only on the value of WAREHOUSE, the entity violates the rule for second normal form. This design causes several problems:

- Each instance for a part that this warehouse stores repeats the address of the warehouse.
- If the address of the warehouse changes, every instance referring to a part that is stored in that warehouse must be updated.
- Because of the redundancy, the data might become inconsistent. Different instances could show different addresses for the same warehouse.
- If at any time the warehouse has no stored parts, the address of the warehouse might not exist in any instances in the entity.

To satisfy second normal form, the information in the previous figure would be in two entities, as the following figure shows.

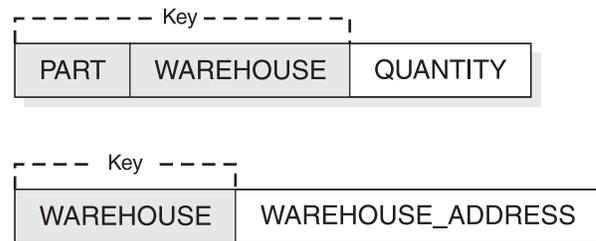


Figure 6. Entities that satisfy the second normal form

Third normal form

An entity is in third normal form if each nonprimary key attribute provides a fact that is independent of other non-key attributes and depends only on the key. A violation of the third normal form occurs when a nonprimary attribute is a fact about another non-key attribute.

Employee_Department table before updating

Key				
EMPLOYEE_NUMBER	EMPLOYEE_FIRST_NAME	EMPLOYEE_LAST_NAME	DEPARTMENT_NUMBER	DEPARTMENT_NAME
000200	DAVID	BROWN	D11	MANUFACTURING
000320	RAMAL	MEHTA	E21	SOFTWARE
000220	JENIFER	LUTZ	D11	MANUFACTURING

Employee_Department table after updating

Key				
EMPLOYEE_NUMBER	EMPLOYEE_FIRST_NAME	EMPLOYEE_LAST_NAME	DEPARTMENT_NUMBER	DEPARTMENT_NAME
000200	DAVID	BROWN	D11	INSTALLATION
000320	RAMAL	MEHTA	E21	SOFTWARE
000220	JENIFER	LUTZ	D11	MANUFACTURING

Figure 7. Results of an update in a table that violates the third normal form

Example: The first entity in the previous figure contains the attributes `EMPLOYEE_NUMBER` and `DEPARTMENT_NUMBER`. Suppose that a program or user adds an attribute, `DEPARTMENT_NAME`, to the entity. The new attribute depends on `DEPARTMENT_NUMBER`, whereas the primary key is on the `EMPLOYEE_NUMBER` attribute. The entity now violates third normal form.

Changing the `DEPARTMENT_NAME` value based on the update of a single employee, David Brown, does not change the `DEPARTMENT_NAME` value for other employees in that department. The updated version of the entity as shown in the previous figure illustrates the resulting inconsistency. Additionally, updating the `DEPARTMENT_NAME` in this table does not update it in any other table that might contain a `DEPARTMENT_NAME` column.

You can normalize the entity by modifying the `EMPLOYEE_DEPARTMENT` entity and creating two new entities: `EMPLOYEE` and `DEPARTMENT`. The following figure shows the new entities. The `DEPARTMENT` entity contains attributes for `DEPARTMENT_NUMBER` and `DEPARTMENT_NAME`. Now, an update such as changing a department name is much easier. You need to make the update only to the `DEPARTMENT` entity.

Employee table

--- Key ---

<code>EMPLOYEE_NUMBER</code>	<code>EMPLOYEE_FIRST_NAME</code>	<code>EMPLOYEE_LAST_NAME</code>
000200	DAVID	BROWN
000320	RAMAL	MEHTA
000220	JENIFER	LUTZ

Department table

--- Key ---

<code>DEPARTMENT_NUMBER</code>	<code>DEPARTMENT_NAME</code>
D11	MANUFACTURING
E21	SOFTWARE

Employee_Department table

--- Key ---

<code>DEPARTMENT_NUMBER</code>	<code>EMPLOYEE_NUMBER</code>
D11	000200
D11	000220
E21	000329

Figure 8. Employee and department entities that satisfy the third normal form

Fourth normal form

An entity is in fourth normal form if no instance contains two or more independent, multivalued facts about an entity.

----- Key -----				
EMPID	SKILL_CODE	LANGUAGE_CODE	SKILL_PROFICIENCY	LANGUAGE_PROFICIENCY

Figure 9. Entity in violation of the fourth normal form

Example: Consider the EMPLOYEE entity. Each instance of EMPLOYEE could have both SKILL_CODE and LANGUAGE_CODE. An employee can have several skills and know several languages. Two relationships exist, one between employees and skills, and one between employees and languages. An entity is not in fourth normal form if it represents both relationships, as the previous figure shows.

Instead, you can avoid this violation by creating two entities that represent both relationships, as the following figure shows.

----- Key -----		
EMPID	SKILL_CODE	SKILL_PROFICIENCY

----- Key -----		
EMPID	LANGUAGE_CODE	LANGUAGE_PROFICIENCY

Figure 10. Entities that satisfy the fourth normal form

If, however, the facts are interdependent (that is, the employee applies certain languages only to certain skills), you should not split the entity.

You can put any data into fourth normal form. A good rule to follow when doing logical database design is to arrange all the data in entities that are in fourth normal form. Then decide whether the result gives you an acceptable level of performance. If the performance is not acceptable, denormalizing your design is a good approach to improving performance.

Logical database design with Unified Modeling Language

Another model that you can use for entity-relationship model of database design is Unified Modeling Language (UML).

The Object Management Group is a consortium that created the UML standard. UML modeling is based on object-oriented programming principals. The basic difference between the entity-relationship model and the UML model is that, instead of designing entities as this chapter illustrates, you model objects. UML defines a standard set of modeling diagrams for all stages of developing a software system. Conceptually, UML diagrams are like the blueprints for the design of a software development project.

Some examples of UML diagrams are listed below:

Class Identifies high-level entities, known as classes. A class describes a set of objects that have the same attributes. A class diagram shows the relationships between classes.

Use case

Presents a high-level view of a system from the user's perspective. A use case diagram defines the interactions between users and applications or between applications. These diagrams graphically depict system behavior. You can work with use-case diagrams to capture system requirements, learn how the system works, and specify system behavior.

Activity

Models the workflow of a business process, typically by defining rules for the sequence of activities in the process. For example, an accounting company can use activity diagrams to model financial transactions.

Interaction

Shows the required sequence of interactions between objects. Interaction diagrams can include sequence diagrams and collaboration diagrams.

- Sequence diagrams show object interactions in a time-based sequence that establishes the roles of objects and helps determine class responsibilities and interfaces.
- Collaboration diagrams show associations between objects that define the sequence of messages that implement an operation or a transaction.

Component

Shows the dependency relationships between components, such as main programs and subprograms.

Developers can graphically represent the architecture of a database and how it interacts with applications using one of many available UML modeling tools. Similarities exist between components of the entity-relationship model and UML diagrams. For example, the class structure corresponds closely to the entity structure.

The logical data model provides an overall view of the captured business requirements as they pertain to data entities. The data model diagram graphically represents the physical data model. The physical data model applies the logical data model's captured requirements to specific DBMS languages. Physical data models also capture the lower-level detail of a DBMS database.

Database designers can customize the data model diagram from other UML diagrams, which allows them to work with concepts and terminology, such as columns, tables, and relationships, with which they are already familiar. Developers can also transform a logical data model into a physical data model.

Because the data model diagram includes diagrams for modeling an entire system, it allows database designers, application developers, and other development team members to share and track business requirements throughout development. For example, database designers can capture information, such as constraints, triggers, and indexes, directly on the UML diagram. Developers can also transfer between object and data models and use basic transformation types such as many-to-many relationships.

Physical database design

After completing the logical design of your database, you now move to the physical design. The purpose of building a physical design of your database is to optimize performance while ensuring data integrity by avoiding unnecessary data redundancies.

During physical design, you transform the entities into tables, the instances into rows, and the attributes into columns. You and your colleagues must decide on many factors that affect the physical design, some of which are listed below.

- How to translate entities into physical tables
- What attributes to use for columns of the physical tables
- Which columns of the tables to define as keys
- What indexes to define on the tables
- What views to define on the tables
- How to denormalize the tables
- How to resolve many-to-many relationships

| Physical design is the time when you abbreviate the names that you chose during
| logical design. For example, you can abbreviate the column name that identifies
| employees, EMPLOYEE_NUMBER, to EMPNO. In previous versions of DB2, you
| needed to abbreviate column and table names to fit the physical constraint of an
| 18-byte limit. Beginning with DB2 version 8, the column name size was increased
| to a 30- byte maximum and the table names size was increased to a 128-byte
| maximum.

The task of building the physical design is a job that truly never ends. You need to continually monitor the performance and data integrity characteristics of the database as time passes. Many factors necessitate periodic refinements to the physical design.

DB2 lets you change many of the key attributes of your design with ALTER SQL statements. For example, assume that you design a partitioned table so that it will store 36 months' worth of data. Later you discover that you need to extend that design to hold 84 months' worth of data. You can add or rotate partitions for the current 36 months to accommodate the new design.

| The remainder of this chapter includes some valuable information that can help
| you as you build and refine your database's physical design.

Denormalization of tables

During physical design, analysts transform the entities into tables and the attributes into columns.

Denormalization is a key step in the task of building a physical relational database design. It is the intentional duplication of columns in multiple tables, and the consequence is increased data redundancy. For more information, see *Introduction to DB2 for z/OS*.

The warehouse address column first appears as part of a table that contains information about parts and warehouses. To further normalize the design of the table, analysts remove the warehouse address column from that table. Analysts also define the column as part of a table that contains information only about warehouses.

Normalizing tables is generally the recommended approach. What if applications require information about both parts and warehouses, including the addresses of warehouses? The premise of the normalization rules is that SQL statements can retrieve the information by joining the two tables. The problem is that, in some cases, performance problems can occur as a result of normalization. For example, some user queries might view data that is in two or more related tables; the result is too many joins. As the number of tables increases, the access costs can increase, depending on the size of the tables, the available indexes, and so on. For example, if indexes are not available, the join of many large tables might take too much time. You might need to denormalize your tables. Denormalization is the intentional duplication of columns in multiple tables, and it increases data redundancy.

Example: Consider the design in which both tables have a column that contains the addresses of warehouses. If this design makes join operations unnecessary, it could be a worthwhile redundancy. Addresses of warehouses do not change often, and if one does change, you can use SQL to update all instances fairly easily.

Tip: Do not automatically assume that all joins take too much time. If you join normalized tables, you do not need to keep the same data values synchronized in multiple tables. In many cases, joins are the most efficient access method, despite the overhead they require. For example, some applications achieve 44-way joins in subsecond response time.

When you are building your physical design, you and your colleagues need to decide whether to denormalize the data. Specifically, you need to decide whether to combine tables or parts of tables that are frequently accessed by joins that have high performance requirements. This is a complex decision about which this book cannot give specific advice. To make the decision, you need to assess the performance requirements, different methods of accessing the data, and the costs of denormalizing the data. You need to consider the trade-off: is duplication, in several tables, of often-requested columns less expensive than the time for performing joins?

Recommendations:

- Do not denormalize tables unless you have a good understanding of the data and the business transactions that access the data. Consult with application developers before denormalizing tables to improve the performance of users' queries.
- When you decide whether to denormalize a table, consider all programs that regularly access the table, both for reading and for updating. If programs frequently update a table, denormalizing the table affects performance of update programs because updates apply to multiple tables rather than to one table.

In the following figure, information about parts, warehouses, and warehouse addresses appears in two tables, both in normal form.



Figure 11. Two tables that satisfy second normal form

The following figure illustrates the denormalized table.

Key			
PARTNO	WRHS_NO	PART_QTY	WRHS_ADDRESS

Figure 12. The denormalized table

Resolving many-to-many relationships is a particularly important activity because doing so helps maintain clarity and integrity in your physical database design. To resolve many-to-many relationships, you introduce associative tables, which are intermediate tables that you use to tie, or associate, two tables to each other.

Example: Employees work on many projects. Projects have many employees. In the logical database design, you show this relationship as a many-to-many relationship between project and employee. To resolve this relationship, you create a new associative table, EMPLOYEE_PROJECT. For each combination of employee and project, the EMPLOYEE_PROJECT table contains a corresponding row. The primary key for the table would consist of the employee number (EMPNO) and the project number (PROJNO).

Another decision that you must make relates to the use of repeating groups.

Example: Assume that a heavily used transaction requires the number of wires that are sold by month in a given year. Performance factors might justify changing a table so that it violates the rule of first normal form by storing repeating groups. In this case, the repeating group would be: MONTH, WIRE. The table would contain a row for the number of sold wires for each month (January wires, February wires, March wires, and so on).

Recommendation: If you decide to denormalize your data, document your denormalization thoroughly. Describe, in detail, the logic behind the denormalization and the steps that you took. Then, if your organization ever needs to normalize the data in the future, an accurate record is available for those who must do the work.

Views as a way to customize what data users see

A view offers an alternative way of describing data that exists in one or more tables.

Some users might find that no single table contains all the data they need; rather, the data might be scattered among several tables. Furthermore, one table might contain more data than users want to see or more than you want to authorize them to see. For those situations, you can create views.

You might want to use views for a variety of reasons:

- To limit access to certain kinds of data

You can create a view that contains only selected columns and rows from one or more tables. Users with the appropriate authorization on the view see only the information that you specify in the view definition.

Example: You can define a view on the EMP table to show all columns except SALARY and COMM (commission). You can grant access to this view to people who are not managers because you probably don't want them to have access to salary and commission information.

- To combine data from multiple tables

You can create a view that uses UNION or UNION ALL operators to logically combine smaller tables, and then query the view as if it were one large table.

Example: Assume that three tables contain data for a period of one month. You can create a view that is the UNION ALL of three fullselects, one for each month of the first quarter of 2004. At the end of the third month, you can view comprehensive quarterly data.

You can create a view any time after the underlying tables exist. The owner of a set of tables implicitly has the authority to create a view on them. A user with administrative authority at the system or database level can create a view for any owner on any set of tables. If they have the necessary authority, other users can also create views on a table that they didn't create.

Related concepts

"DB2 views" (Introduction to DB2 for z/OS)

Indexes on table columns

If you are involved in the physical design of a database, you will be working with other designers to determine what columns you should index.

You will use process models that describe how different applications are going to be accessing the data. This information is important when you decide on indexing strategies to ensure adequate performance.

The main purposes of an index are as follows:

- To optimize data access.

In many cases, access to data is faster with an index than without an index. If the DBMS uses an index to find a row in a table, the scan can be faster than when the DBMS scans an entire table.

- To ensure uniqueness.

A table with a unique index cannot have two rows with the same values in the column or columns that form the index key.

Example: If payroll applications use employee numbers, no two employees can have the same employee number.

- To enable clustering.

A clustering index keeps table rows in a specified sequence, to minimize page access for a set of rows. When a table space is partitioned, a special type of clustering occurs; rows are clustered within each partition.

The following example applies to table-based partitioning but not to index-based partitioning because the partitioning index must be the clustering index.

Example: If the partitioning scheme on the month and the clustering index is on the name, the rows will be clustered on the name within the month.

In general, users of the table are unaware that an index is in use. DB2 decides whether to use the index to access the table.

Chapter 2. Implementing your database design

Implementing your database design involves implementing DB2 objects, loading and managing data, and altering your design as necessary.

Tip:  You can simplify your database implementation by letting DB2 implicitly create certain objects for you. On a CREATE TABLE statement, if you do not specify a database name, DB2 will use an existing implicitly created database. If an implicitly created database does not exist, DB2 will create one by using the naming convention of DSNxxxxx (from DSN00001 to DSN60000). If the table space is implicitly created, DB2 will create all of the required system objects for you, including:

- Enforcing the primary key index and the unique key index
- Creating the ROWID index (if the ROWID column is defined as GENERATED BY DEFAULT)
- Creating the LOB table spaces, the auxiliary tables, and the auxiliary indexes



Implementing DB2 databases

Use DB2 databases to collect and control data.

Creating DB2 databases

You can create a DB2 database by defining a database at the current server.

 Issue the CREATE DATABASE SQL statement to define a database.

The CREATE DATABASE statement allows you to define the following clauses:

STOGROUP

Lets you change the name of the default storage group to support disk space requirements for table spaces and indexes within the database. The new default DB2 storage group is used only for new table spaces and indexes; existing definitions do not change.

BUFFERPOOL

Lets you change the name of the default buffer pool for table spaces and indexes within the database. Again, it applies only to new table spaces and indexes; existing definitions do not change.

INDEXBP

Lets you change the name of the default buffer pool for the indexes within the database. The new default buffer pool is used only for new indexes; existing definitions do not change.



Related concepts

"DB2 databases" (Introduction to DB2 for z/OS)

Dropping DB2 databases

You can drop a DB2 database by removing the object at the current server.

GUIP Issue the DROP DATABASE SQL statement to remove the database.

GUIP

Related concepts

"DB2 databases" (Introduction to DB2 for z/OS)

Implementing DB2 storage groups

DB2 uses storage groups to allocate storage for table spaces and indexes and to define, extend, alter, and delete VSAM data sets.

You have the following options for creating storage groups and managing DB2 data sets:

- You can let DB2 manage the data sets. This option means less work for DB2 database administrators.
- You can let SMS manage some or all of the data sets, either when you use DB2 storage groups or when you use data sets that you have defined yourself. This option offers a reduced workload for DB2 database administrators and storage administrators. For more information, see "Enabling SMS to control DB2 storage groups" on page 22.
- You can define and manage your own data sets using VSAM Access Method Services. This option gives you the most control over the physical storage of tables and indexes.

Advantages of storage groups

Allowing DB2 to manage your data sets by using DB2 storage groups offers several advantages.

The following list describes some of the things that DB2 does for you in managing your auxiliary storage requirements:

- When a table space is created, DB2 defines the necessary VSAM data sets using VSAM Access Method Services. After the data sets are created, you can process them with access method service commands that support VSAM control-interval (CI) processing (for example, IMPORT and EXPORT).

Exception: **GUIP** You can defer the allocation of data sets for table spaces and index spaces by specifying the DEFINE NO clause on the associated statement (CREATE TABLESPACE and CREATE INDEX), which also must specify the USING STOGROUP clause. **GUIP**

- When a table space is dropped, DB2 automatically deletes the associated data sets.
- When a data set in a segmented or simple table space reaches its maximum size of 2 GB, DB2 might automatically create a new data set. The primary data set allocation is obtained for each new data set.
- When needed, DB2 can extend individual data sets.
- When you create or reorganize a table space that has associated data sets, DB2 deletes and then redefines them, reclaiming fragmented space. However, when

you run REORG with the REUSE option and SHRLEVEL NONE, REORG resets and reuses DB2-managed data sets without deleting and redefining them. If the size of your table space is not changing, using the REUSE parameter could be more efficient.

Exception:  When reorganizing a LOB table space with the SHRLEVEL NONE option, DB2 does not delete and redefine the first data set that was allocated for the table space. If the REORG results in empty data sets beyond the first data set, DB2 deletes those empty data sets. 

- When you want to move data sets to a new volume, you can alter the volumes list in your storage group. DB2 automatically relocates your data sets during the utility operations that build or rebuild a data set (LOAD REPLACE, REORG, REBUILD, and RECOVER).

Restriction: If you use the REUSE option, DB2 does not delete and redefine the data sets and therefore does not move them.

For a LOB table space, you can alter the volumes list in your storage group, and DB2 automatically relocates your data sets during the utility operations that build or rebuild a data set (LOAD REPLACE and RECOVER).

To move user-defined data sets, you must delete and redefine the data sets in another location.

Control interval sizing

A *control interval* is a fixed-length area or disk in which VSAM stores records and creates distributed free space. A control interval is the unit of information that VSAM transmits to or from disk.

DB2 page sets are defined as VSAM linear data sets. Prior to Version 8, DB2 defined all data sets with VSAM control intervals that were 4 KB in size. Beginning in Version 8, DB2 can define data sets with variable VSAM control intervals. One of the biggest benefits of this change is an improvement in query processing performance.

The VARY DS CONTROL INTERVAL parameter on installation panel DSNTIP7 allows you to control whether DB2-managed data sets have variable VSAM control intervals:

- A value of YES indicates that a DB2-managed data set is created with a VSAM control interval that corresponds to the size of the buffer pool that is used for the table space. This is the default value.
- A value of NO indicates that a DB2-managed data set is created with a fixed VSAM control interval of 4 KB, regardless of the size of the buffer pool that is used for the table space.

The following table shows the default and compatible control interval sizes for each table space page size. For example, a table space with pages 16 KB in size can have a VSAM control interval of 4 KB or 16 KB. Control interval sizing has no impact on indexes; index pages are always 4 KB in size.

Table 1. Default and compatible control interval sizes

Table space page size	Default control interval size	Compatible control interval sizes
4 KB	4 KB	4 KB
8 KB	8 KB	4 KB, 8 KB

Table 1. Default and compatible control interval sizes (continued)

Table space page size	Default control interval size	Compatible control interval sizes
16 KB	16 KB	4 KB, 16 KB
32 KB	32 KB	4 KB, 32 KB

Creating DB2 storage groups

You can create DB2 storage groups by using the SQL statement CREATE STOGROUP.

GUIP To create a DB2 storage group, complete the following steps:

1. Issue the SQL statement CREATE STOGROUP.
2. Specify the storage group name.

DB2 storage group names are unqualified identifiers of up to 128 characters. A DB2 storage group name cannot be the same as any other storage group name in the DB2 catalog. **GUIP**

After you define a storage group, DB2 stores information about it in the DB2 catalog. (This catalog is not the same as the integrated catalog facility catalog that describes DB2 VSAM data sets). The catalog table SYSIBM.SYSSTOGROUP has a row for each storage group, and SYSIBM.SYSVOLUMES has a row for each volume. With the proper authorization, you can retrieve the catalog information about DB2 storage groups by using SQL statements.

Enabling SMS to control DB2 storage groups

Using the Storage Management Subsystem (SMS) family of products to manage your data sets can result in a reduced workload for DB2 database administrators and storage administrators.

GUIP To enable SMS to control DB2 storage groups:

1. Issue a CREATE STOGROUP SQL statement to define a DB2 storage group. You can specify SMS classes when you create a storage group.
2. Indicate how you want SMS to control the allocation of volumes in one of the following ways:
 - Specify an asterisk (*) for the VOLUMES attribute.
 - Specify the DATACLAS, MGMTCLAS, or STORCLAS keywords. **GUIP**

If you use DB2 to allocate data to specific volumes, you must assign an SMS storage class with guaranteed space, and you must manage free space for each volume to prevent failures during the initial allocation and extension. Using guaranteed space reduces the benefits of SMS allocation, requires more time for space management, and can result in more space shortages. You should only use guaranteed space when space needs are relatively small and do not change.

Deferring allocation of DB2-managed data sets

When you execute a CREATE TABLESPACE statement with the USING STOGROUP clause, DB2 generally defines the necessary VSAM data sets for the table space. In some cases, however, you might want to define a table space without immediately allocating the associated data sets.

For example, you might be installing a software program that requires that many table spaces be created, but your company might not need to use some of those table spaces; you might prefer not to allocate data sets for the table spaces you will not be using.

The deferral of allocating data sets is recommended when:

- Performance of the CREATE TABLESPACE statement is important.
- Disk resource is constrained.

GUIP To defer the physical allocation of DB2-managed data sets, complete the following steps:

1. Issue a CREATE TABLESPACE SQL statement.

2. Specify the DEFINE NO clause. **GUIP**

The table space is created, but DB2 does not allocate (that is, define) the associated data sets until a row is inserted or loaded into a table in that table space. The DB2 catalog table SYSIBM.SYSTABLESPART contains a record of the created table space and an indication that the data sets are not yet allocated.

Restriction: The DEFINE NO clause is not allowed for LOB table spaces, for table spaces in a work file database, or for user-defined data sets. (In the case of user-defined data sets, the table space is created with the USING VCAT clause of the CREATE TABLESPACE statement).

Do not use the DEFINE NO clause on a table space if you plan to use a tool outside of DB2 to propagate data into a data set in the table space. When you use DEFINE NO, the DB2 catalog indicates that the data sets have not yet been allocated for that table space. Then, if data is propagated from a tool outside of DB2 into a data set in the table space, the DB2 catalog information does not reflect the fact that the data set has been allocated. The resulting inconsistency causes DB2 to deny application programs access to the data until the inconsistency is resolved.

How DB2 extends data sets

When a data set is created, DB2 allocates a primary allocation space on a volume that has space available and that is specified in the DB2 storage group. Any extension to a data set always gets a secondary allocation space.

If new extensions reach the end of the volume, DB2 accesses all candidate volumes from the DB2 storage group and issues the Access Method Services command ALTER ADDVOLUMES to add these volumes to the integrated catalog facility (ICF) catalog as candidate volumes for the data set. DB2 then makes a request to allocate a secondary extent on any one of the candidate volumes that has space available. After the allocation is successful, DB2 issues the command ALTER REMOVEVOLUMES to remove all candidate volumes from the ICF catalog for the data set.

DB2 extends data sets when either of the following conditions occurs:

- The requested space exceeds the remaining space in the data set.
- 10% of the secondary allocation space (but not over 10 allocation units, based on either tracks or cylinders) exceeds the remaining space.

If DB2 fails to extend a data set with a secondary allocation space because of insufficient available space on any single candidate volume of a DB2 storage

group, DB2 tries again to extend with the requested space if the requested space is smaller than the secondary allocation space. Typically, DB2 requests only one additional page. In this case, a small amount of two units (tracks or cylinders, as determined by DFSMS™ based on the SECQTY value) is allocated. To monitor data set extension activity, use IFCID 258 in statistics class 3.

Nonpartitioned spaces

For a nonpartitioned table space or a nonpartitioned index space, DB2 defines the first piece of the page set starting with a primary allocation space, and extends that piece by using secondary allocation spaces. When the end of the first piece is reached, DB2 defines a new piece (which is a new data set) and extends that new piece starting with a primary allocation space.

Exception: In the case where the OPTIMIZE EXTENT SIZING parameter (MGEXTSZ) on installation panel DSNTIP7 is set to YES and the SECQTY value for the table space or index space is greater than zero, the primary space allocation of each subsequent data set is the larger of the SECQTY setting and the value that is derived from a sliding scale algorithm. See “Secondary space allocation” on page 25 for information about the sliding scale algorithm.

Partitioned spaces

For a partitioned table space or a partitioned index space, each partition is a data set. Therefore, DB2 defines each partition with the primary allocation space and extends each partition’s data set by using a secondary allocation space, as needed.

Extension failures

If a data set uses all possible extents, DB2 cannot extend that data set. For a partitioned page set, the extension fails only for the particular partition that DB2 is trying to extend. For nonpartitioned page sets, DB2 cannot extend to a new data set piece, which means that the extension for the entire page set fails.

To avoid extension failures, allow DB2 to use the default value for primary space allocation and to use a sliding scale algorithm for secondary extent allocations.

DB2 space allocation

Primary and secondary space allocation sizes are the main factors that affect the amount of disk space that DB2 uses.

In general, the primary space allocation must be large enough to handle the storage needs that you anticipate. The secondary space allocation must be large enough for your applications to continue operating until the data set is reorganized.

If the secondary space allocation is too small, the data set might have to be extended more times to satisfy those activities that need a large space.

Primary space allocation

DB2 uses default values for primary space allocation of DB2-managed data sets.

The default values are:

- 1 cylinder (720 KB) for non-LOB table spaces
- 10 cylinders for LOB table spaces

- 1 cylinder for indexes

GUIP To indicate that you want DB2 to use the default values for primary space allocation of table spaces and indexes, specify a value of 0 for the following parameters on installation panel DSNTIP7, as shown in the following table.

Table 2. DSNTIP7 parameter values for managing space allocations

Installation panel DSNTIP7 parameter	Recommended value
TABLE SPACE ALLOCATION	0
INDEX SPACE ALLOCATION	0

Thereafter:

- On CREATE TABLESPACE and CREATE INDEX statements, do not specify a value for the PRIQTY option.
- On ALTER TABLESPACE and ALTER INDEX statements, specify a value of -1 for the PRIQTY option.

Primary space allocation quantities do not exceed DSSIZE or PIECESIZE clause values.

Exception: If the OPTIMIZE EXTENT SIZING parameter (MGEXTSZ) on installation panel DSNTIP7 is set to YES and the table space or index space has a SECQTY setting of greater than zero, the primary space allocation of each subsequent data set is the larger of the SECQTY setting and the value that is derived from a sliding scale algorithm. See “Secondary space allocation” for information about the sliding scale algorithm.

For those situations in which the default primary quantity value is not large enough, you can specify a larger value for the PRIQTY option when creating or altering table spaces and indexes. DB2 always uses a PRIQTY value if one is explicitly specified.

If you want to prevent DB2 from using the default value for primary space allocation of table spaces and indexes, specify a non-zero value for the TABLE SPACE ALLOCATION and INDEX SPACE ALLOCATION parameters on

installation panel DSNTIP7. **GUIP**

Secondary space allocation

DB2 can calculate the amount of space to allocate to secondary extents by using a sliding scale algorithm.

The first 127 extents are allocated in increasing size, and the remaining extents are allocated based on the initial size of the data set:

- For 32 GB and 64 GB data sets, each extent is allocated with a size of 559 cylinders.
- For data sets that range in size from less than 1 GB to 16 GB, each extent is allocated with a size of 127 cylinders.

This approach has several advantages:

- It minimizes the potential for wasted space by increasing the size of secondary extents slowly at first.

- It prevents very large allocations for the remaining extents, which would likely cause fragmentation.
- It does not require users to specify SECQTY values when creating and altering table spaces and index spaces.
- It is theoretically possible to reach maximum data set size without running out of secondary extents.

In the case of severe DASD fragmentation, it can take up to 5 extents to satisfy a logical extent request. In this situation, the data set does not reach the theoretical data set size.

If you installed DB2 on the operating system z/OS Version 1 Release 7, or later, then you can modify the Extent Constraint Removal option. By setting the Extent Constraint Removal option to YES in the SMS data class, the maximum number of extents can be up to 7257. However, the limits of 123 extents per volume and a maximum volume count of 59 per data set remain valid. For more information, see "Using VSAM extent constraint removal" in the z/OS V1R7 guide *"DFSMS: Using the New Functions"* (order number SC26-7473-02).

Maximum allocation is shown in the following table. This table assumes that the initial extent that is allocated is one cylinder in size.

Table 3. Maximum allocation of secondary extents

Maximum data set size, in GB	Maximum allocation, in cylinders	Extents required to reach full size
1	127	54
2	127	75
4	127	107
8	127	154
16	127	246
32	559	172
64	559	255

GUIP DB2 uses a sliding scale for secondary extent allocations of table spaces and indexes when:

- You do not specify a value for the SECQTY option of a CREATE TABLESPACE or CREATE INDEX statement
- You specify a value of -1 for the SECQTY option of an ALTER TABLESPACE or ALTER INDEX statement.

Otherwise, DB2 always uses a SECQTY value for secondary extent allocations, if one is explicitly specified. **GUIP**

Exception: For those situations in which the calculated secondary quantity value is not large enough, you can specify a larger value for the SECQTY option when creating or altering table spaces and indexes. However, in the case where the OPTIMIZE EXTENT SIZING parameter is set to YES and you specify a value for the SECQTY option, DB2 uses the value of the SECQTY option to allocate a secondary extent only if the value of the option is larger than the value that is derived from the sliding scale algorithm. The calculation that DB2 uses to make this determination is:

Actual secondary extent size = max (min (ss_extent, MaxAlloc), SECQTY)

In this calculation, *ss_extent* represents the value that is derived from the sliding scale algorithm, and *MaxAlloc* is either 127 or 559 cylinders, depending on the maximum potential data set size. This approach allows you to reach the maximum page set size faster. Otherwise, DB2 uses the value that is derived from the sliding scale algorithm.

If you do not provide a value for the secondary space allocation quantity, DB2 calculates a secondary space allocation value equal to 10% of the primary space allocation value and subject to the following conditions:

- The value cannot be less than 127 cylinders for data sets that range in initial size from less than 1 GB to 16 GB, and cannot be less than 559 cylinders for 32 GB and 64 GB data sets.
- The value cannot be more than the value that is derived from the sliding scale algorithm.

The calculation that DB2 uses for the secondary space allocation value is:

Actual secondary extent size = max (0.1 × PRIQTY, min (ss_extent, MaxAlloc))

In this calculation, *ss_extent* represents the value that is derived from the sliding scale algorithm, and *MaxAlloc* is either 127 or 559 cylinders, depending on the maximum potential data set size.

Secondary space allocation quantities do not exceed DSSIZE or PIECESIZE clause values.

If you do not want DB2 to extend a data set, you can specify a value of 0 for the SECQTY option. Specifying 0 is a useful way to prevent DSNDB07 work files from growing out of proportion.

If you want to prevent DB2 from using the sliding scale for secondary extent allocations of table spaces and indexes, specify a value of NO for the OPTIMIZE EXTENT SIZING parameter on installation panel DSNTIP7.

Example of primary and secondary space allocation

The following table shows how primary and secondary quantities are affected by a CREATE statement and two subsequent ALTER statements.

This example assumes a maximum data set size of less than 32 GB, and the following parameter values on installation panel DSNTIP7:

- TABLE SPACE ALLOCATION = 0
- INDEX SPACE ALLOCATION = 0
- OPTIMIZE EXTENT SIZING = YES

Table 4. Example of specified and actual space allocations

Action	Specified PRIQTY	Actual primary quantity allocated	Specified SECQTY	Actual secondary quantity allocated
CREATE TABLESPACE	100 KB	100 KB	1000 KB	2 cylinders
ALTER TABLESPACE	-1	1 cylinder	2000 KB	3 cylinders
ALTER TABLESPACE		1 cylinder	-1	1 cylinder

Managing DB2 data sets with DFSMSHsm

You can use the Hierarchical Storage Management functional component (DFSMSHsm™) of DFSMS to manage space and data availability among the storage devices in your system.

You can also use DFSMSHsm to move data sets that have not been recently used to slower, less expensive storage devices. Moving the data sets helps to ensure that disk space is managed efficiently.

Related information

 z/OS Library Center

Migrating to DFSMSHsm

If you decide to use DFSMSHsm for your DB2 data sets, you should develop a migration plan with your system administrator.

With user-managed data sets, you can specify DFSMSHsm classes on the Access Method Services DEFINE command. With DB2 storage groups, you need to develop automatic class selection routines.

Restriction: If you use the BACKUP SYSTEM utility to create system-level backups, do not use DFSMSHsm to migrate DB2 table spaces and indexes. You can use DFSMSHsm to migrate or recall archive log data sets.

GUPI To enable DFSMSHsm to manage your DB2 storage groups, complete the following steps:

1. Issue either a CREATE STOGROUP or ALTER STOGROUP SQL statement.
2. Specify one or more asterisks as *volume-ID* in the VOLUMES option.

The following example causes all database data set allocations and definitions to use nonspecific selection through DFSMSHsm filtering services.

```
CREATE STOGROUP G202
VOLUMES ('*')
VCAT vcat name
DATACLAS dataclass name
MGMTCLAS management class name
STORCLAS storage class name;
```

3. Define the data classes for your table space data sets and index data sets.
4. Code the SMS automatic class selection (ACS) routines to assign indexes to one SMS storage class and to assign table spaces to a different SMS storage class.
5. Use the system parameters SMSDCFL and SMSDCIX to assign table spaces and indexes to different DFSMSHsm data classes.
 - SMSDCFL specifies a DFSMSHsm data class for table spaces. If you assign a value to SMSDCFL, DB2 specifies that value when it uses Access Method Services to define a data set for a table space.
 - SMSDCIX specifies a DFSMSHsm data class for indexes. If you assign a value to SMSDCIX, DB2 specifies that value when it uses Access Method Services to define a data set for an index.

Important: If you specified the DATACLAS keyword in the CREATE STOGROUP statement, the DATACLAS value overrides the SMSDCFL and SMSDCIX values.

GUI

The following example shows how to create a storage group in a SMS managed subsystem:

```
CREATE STOGROUP SG0S0101
  VCAT REGSMS
  DATACLAS REGSMSDC
  MGMTCLAS REGSMSMC
  STORCLAS REGSMSSC;
```

How archive logs are recalled by DFSMSHsm

DFSMSHsm can automatically migrate and recall archive log and image copy data sets. If DB2 needs an archive log data set or an image copy data set that DFSMSHsm has migrated, a recall begins automatically and DB2 waits for the recall to complete before continuing.

For processes that read more than one archive log data set, such as the RECOVER utility, DB2 anticipates a DFSMSHsm recall of migrated archive log data sets. When a DB2 process finishes reading one data set, it can continue with the next data set without delay, because the data set might already have been recalled by DFSMSHsm.

If you accept the default value YES for the RECALL DATABASE parameter on the Operator Functions panel (DSNTIPO), DB2 also recalls migrated table spaces and index spaces. At data set open time, DB2 waits for DFSMSHsm to perform the recall. You can specify the amount of time DB2 waits while the recall is being performed with the RECALL DELAY parameter, which is also on panel DSNTIPO. If RECALL DELAY is set to zero, DB2 does not wait, and the recall is performed asynchronously.

You can use System Managed Storage (SMS) to archive DB2 subsystem data sets, including the DB2 catalog, DB2 directory, active logs, and work file databases (DSNDB07 in a non-data-sharing environment). However, before starting DB2, you should recall these data sets by using DFSMSHsm. Alternatively, you can avoid migrating these data sets by assigning them to a management class that prevents migration.

If a volume has a STOGROUP specified, you must recall that volume only to volumes of the same device type as others in the STOGROUP.

In addition, you must coordinate the DFSMSHsm automatic purge period, the DB2 log retention period, and MODIFY utility usage. Otherwise, the image copies or logs that you might need during a recovery could already have been deleted.

The RECOVER utility and the DFSMSdss RESTORE command

The RECOVER utility can run the DFSMSdss™ RESTORE command, which generally uses extensions that are larger than the data set's primary and secondary space allocation values.

The RECOVER utility runs this command if the point of recovery is defined by an image copy that was taken by using the CONCURRENT option of the COPY utility.

When the RECOVER utility chooses a system-level backup for object-level recovery, DFSMShsm is used to restore the data sets from the system-level backup.

The DFSMSdss RESTORE command extends a data set differently than DB2, so after this command runs, you must alter the page set to contain extents that are defined by DB2.

Restrictions associated with the BACKUP SYSTEM utility and DFSMShsm

If you plan to use the DB2 BACKUP SYSTEM online utility to take volume-level copies of the data and logs of a non-data-sharing DB2 subsystem or a DB2 data sharing group, all of the DB2 data sets must reside on volumes that are managed by DFSMSsms.

Restriction: If you use the BACKUP SYSTEM utility to create system-level backups, do not use DFSMShsm to migrate DB2 table spaces and indexes.

The BACKUP SYSTEM utility uses copy pools. A *copy pool* is a named set of storage groups that can be backed up and restored as a unit; DFSMShsm processes the storage groups collectively for fast replication. Each DB2 subsystem has up to two copy pools, one for databases and one for logs.

Copy pools are also referred to as source storage groups. Each source storage group contains the name of an associated copy-pool backup storage group, which contains eligible volumes for the backups. The storage administrator must define both the source and target storage groups, and use the following DB2 naming convention:

DSN\$locn-name\$cp-type

The variables that are used in this naming convention are described in the following table.

Table 5. Naming convention variables

Variable	Meaning
DSN	The unique DB2 product identifier
\$	A delimiter. You must use the dollar sign (\$) character.
locn-name	The DB2 location name
cp-type	The copy pool type. Use DB for database and LG for log.

The DB2 BACKUP SYSTEM and RESTORE SYSTEM utilities invoke DFSMShsm to back up and restore the copy pools. DFSMShsm interacts with DFSMSsms to determine the volumes that belong to a given copy pool so that the volume-level backup and restore functions can be invoked.

Tip: The BACKUP SYSTEM utility can dump the copy pools to tape automatically if you specify the options that enable that function.

Related reference

"BACKUP SYSTEM" (DB2 Utility Guide and Reference)

"RESTORE SYSTEM" (DB2 Utility Guide and Reference)

Incremental system-level backups

You can use the DB2 BACKUP SYSTEM online utility to take incremental FlashCopy® backups of the data of a non-data-sharing DB2 subsystem or a DB2 data sharing group. All of the DB2 data sets must reside on volumes that are managed by DFSMSsms.

An incremental FlashCopy relationship is established for each source volume in the copy pool with corresponding target volumes. Each source volume can have only one incremental relationship. Therefore, multiple incremental FlashCopy backup versions are not supported.

The incremental FlashCopy backup feature is supported by the z/OS Version 1 Release 8, or later operating system. To support this feature the following keywords were added to the syntax of the BACKUP SYSTEM utility:

- **ESTABLISH FCINCREMENTAL**: Specifies that a persistent incremental FlashCopy relationship is to be established, if none exists for source copy volumes in the database copy pool. Use this keyword once to establish the persistent incremental FlashCopy relationships. Subsequent invocations of BACKUP SYSTEM (without this keyword) will automatically process the persistent incremental FlashCopy relationship.
- **END FCINCREMENTAL**: Specifies that a last incremental FlashCopy backup be taken and for the persistent incremental FlashCopy relationship to be withdrawn for all of the volumes in the database copy pool. Use this keyword only if no further incremental FlashCopy backups of the database copy pool are desired.

The first time that you use the ESTABLISH FCINCREMENTAL keyword in an invocation of the BACKUP SYSTEM utility the persistent incremental FlashCopy relationship is established. The incremental FlashCopy relationship exists until you withdraw it by specifying the END FCINCREMENTAL keyword in the utility control statement.

For the first invocation of BACKUP SYSTEM that specifies the ESTABLISH FCINCREMENTAL keyword, all of the tracks of each source volume are copied to their corresponding target volumes. For subsequent BACKUP SYSTEM requests, only the changed tracks are copied to the target volumes.

If you keep more than one DASD FlashCopy version of the database copy pool, you need to create full-copy backups for versions other than the incremental version.

For example, you decide to keep two DASD FlashCopy versions of your database copy pool. You invoke the BACKUP SYSTEM utility with the ESTABLISH FCINCREMENTAL keyword. A full-copy of each volume is created, because the incremental FlashCopy relationship is established for the first time. You invoke the BACKUP SYSTEM utility the next day. This request creates the second version of the backup. This version is a full-copy backup, because the incremental FlashCopy relationship is established with the target volumes in the first version. The following day you run the BACKUP SYSTEM utility again, but without the ESTABLISH FCINCREMENTAL keyword. The incremental version is the oldest version, so the incremental version is used for the FlashCopy backup. This time only the tracks that have changed are copied. The result is a complete copy of the source volume.

DFSMSshm allows multiple versions of FlashCopy backups for a copy pool. However, only one incremental FlashCopy backup is supported. If the database

copy pool has two versions of FlashCopy backups, every other copy is an incremental copy since the oldest copy is replaced.

Managing your own data sets

You might choose to manage your own VSAM data sets for several reasons.

For example:

- You have a large linear table space on several data sets. If you manage your own data sets, you can better control the placement of individual data sets on the volumes (although you can keep a similar type of control by using single-volume DB2 storage groups).
- You want to prevent deleting a data set within a specified time period, by using the TO and FOR options of the Access Method Services DEFINE and ALTER commands. You can create and manage the data set yourself, or you can create the data set with DB2 and use the ALTER command of Access Method Services to change the TO and FOR options.
- You are concerned about recovering dropped table spaces. Your own data set is not automatically deleted when a table space is dropped, making it easier to reclaim the data.

Tip: As table spaces and index spaces expand, you might need to provide additional data sets. To take advantage of parallel I/O streams when doing certain read-only queries, consider spreading large table spaces over different disk volumes that are attached on separate channel paths.

Defining data sets

DB2 checks whether you have defined your data sets correctly.

You must define a data set for each of the following items:

- A simple or segmented table space
- A partition of a partitioned table space
- A partition of a partitioned index

GUPI You must define the data sets before you can issue the CREATE TABLESPACE, CREATE INDEX, or ALTER TABLE ADD PARTITION SQL statements.

If you create a partitioned table space, you must create a separate data set for each partition, or you must allocate space for each partition by using the PARTITION option of the NUMPARTS clause in the CREATE TABLESPACE statement.

If you create a partitioned secondary index, you must create a separate data set for each partition. Alternatively, for DB2 to manage your data sets, you must allocate space for each partition by using the PARTITIONED option of the CREATE INDEX statement.

If you create a partitioning index that is partitioned, you must create a separate data set for each partition. Alternatively, for DB2 to manage your data sets, you must allocate space for each partition by using the PARTITIONED option or the PARTITION ENDING AT clause of the CREATE INDEX statement in the case of index-controlled partitioning. **GUPI**

To define and manage VSAM data sets yourself, complete the following steps:

1. Issue a DEFINE CLUSTER statement to create the data set.
2. Give each data set a name that complies with the following format:
catname.DSNDBx.dbname.psname.y0001.znnn

For more information, see “Data set naming conventions” on page 34.

3. In the DEFINE CLUSTER statement, specify the size of the primary and secondary extents of the VSAM cluster. If you specify zero for the secondary extent size, data set extension does not occur.
4. Specify that the data sets be LINEAR. Do not use RECORDSIZE; this attribute is invalid. Use the CONTROLINTERVALSIZE attribute if you are using variable-sized control intervals.
5. Specify the REUSE option. You must define the data set as REUSE before running the DSN1COPY utility.
6. Use SHAREOPTIONS(3,3).

Example:

The following example code shows an example of the DEFINE CLUSTER command, which defines a VSAM data set for the SYSUSER table space in database DSNDB06. Assume that an integrated catalog facility catalog named DSNCAT is already defined.

```
DEFINE CLUSTER -
  (NAME(DSNCAT.DSNDBC.DSNDB06.SYSUSER.I0001.A001) -
   LINEAR -
   REUSE -
   VOLUMES(DSNV01) -
   RECORDS(100 100) -
   SHAREOPTIONS(3 3) ) -
  DATA -
  (NAME(DSNCAT.DSNDBD.DSNDB06.SYSUSER.I0001.A001) -
   CATALOG(DSNCAT))
```

For user-managed data sets, you must pre-allocate shadow data sets prior to running the following against the table space:

- REORG with SHRLEVEL CHANGE
- REORG with SHRLEVEL REFERENCE
- CHECK INDEX with SHRLEVEL CHANGE
- CHECK DATA with SHRLEVEL CHANGE
- CHECK LOB with SHRLEVEL CHANGE

You can specify the MODEL option for the DEFINE CLUSTER command so that the shadow is created like the original data set, as shown in the following example code.

```
DEFINE CLUSTER -
  (NAME('DSNCAT.DSNDBC.DSNDB06.SYSUSER.x0001.A001') -
   MODEL('DSNCAT.DSNDBC.DSNDB06.SYSUSER.y0001.A001')) -
  DATA -
  (NAME('DSNCAT.DSNDBD.DSNDB06.SYSUSER.x0001.A001') -
   MODEL('DSNCAT.DSNDBD.DSNDB06.SYSUSER.y0001.A001')) -
```

In the previous example, the instance qualifiers *x* and *y* are distinct and are equal to either **I** or **J**. You must determine the correct instance qualifier to use for a shadow data set by querying the DB2 catalog for the database and table space.

The DEFINE CLUSTER command has many optional parameters that do not apply when DB2 uses the data set. If you use the parameters SPANNED, EXCEPTIONEXIT, BUFFERSPACE, or WRITECHECK, VSAM applies them to your data set, but DB2 ignores them when it accesses the data set.

The value of the OWNER parameter for clusters that are defined for storage groups is the first SYSADM authorization ID specified at installation.

When you drop indexes or table spaces for which you defined the data sets, you must delete the data sets unless you want to reuse them. To reuse a data set, first commit, and then create a new table space or index with the same name. When DB2 uses the new object, it overwrites the old information with new information, which destroys the old data.

Likewise, if you delete data sets, you must drop the corresponding table spaces and indexes; DB2 does not drop these objects automatically.

Related concepts

“Advantages of storage groups” on page 20

Data set naming conventions:

When you define a data set, you must give each data set a name that complies with the correct format.

The correct format for the name of a data set is as follows:

catname.DSNDBx.dbname.psname.y0001.znnn

catname

Integrated catalog name or alias (up to eight characters). Use the same name or alias here as in the USING VCAT clause of the CREATE TABLESPACE and CREATE INDEX statements.

x C (for VSAM clusters) or D (for VSAM data components).

dbname

DB2 database name. If the data set is for a table space, *dbname* must be the name given in the CREATE TABLESPACE statement. If the data set is for an index, *dbname* must be the name of the database containing the base table. If you are using the default database, *dbname* must be DSNDB04.

psname

Table space name or index name. This name must be unique within the database.

You use this name on the CREATE TABLESPACE or CREATE INDEX statement. (You can use a name longer than eight characters on the CREATE INDEX statement, but the first eight characters of that name must be the same as in the data set's *psname*.)

y0001 Instance qualifier for the data set.

Define one data set for the table space or index with a value of I for y if you *do not* plan to run the following:

- REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE
- CHECK DATA with SHRLEVEL REFERENCE
- CHECK INDEX with SHRLEVEL REFERENCE
- CHECK LOB with SHRLEVEL REFERENCE

Define two data sets if you plan to run the following:

- REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE
- CHECK DATA with SHRLEVEL CHANGE
- CHECK INDEX with SHRLEVEL CHANGE
- CHECK LOB with SHRLEVEL CHANGE

Define one data set with a value of I for *y*, and one with a value of J for *y*.

znnn Data set number. The first digit *z* of the data set number is represented by the letter A, B, C, D, or E, which corresponds to the value 0, 1, 2, 3, or 4 as the first digit of the partition number.

For partitioned table spaces, if the partition number is less than 1000, the data set number is *Axxx* in the data set name (for example, A999 represents partition 999). For partitions 1000 to 1999, the data set number is *Bxxx* (for example, B000 represents partition 1000). For partitions 2000 to 2999, the data set number is *Cxxx*. For partitions 3000 to 3999, the data set number is *Dxxx*. For partitions 4000 up to a maximum of 4096, the data set number is *Exxx*.

The naming convention for data sets that you define for a partitioned index is the same as the naming convention for other partitioned objects.

For simple or segmented table spaces, the number is 001 (preceded by A) for the first data set. When little space is available, DB2 issues a warning message. If the size of the data set for a simple or a segmented table space approaches the maximum limit, define another data set with the same name as the first data set and the number 002. The next data set will be 003, and so on.

You can reach the VSAM extent limit for a data set before you reach the size limit for a partitioned or a nonpartitioned table space. If this happens, DB2 does not extend the data set.

Extending user-managed data sets

A user-managed data set is allocated by using only volumes defined for that data set in the ICF catalog. Before the current volume runs out of space, you must extend the data set.

To extend a user-managed data set:

Issue the Access Method Services commands ALTER ADDVOLUMES or ALTER REMOVEVOLUMES for candidate volumes.

Deleting user-managed data sets

If you manage the data sets of a storage structure yourself, at some point you might need to delete data sets.

To delete a user-managed data set:

Issue the DELETE CLUSTER command for candidate volumes.

Defining index space storage

Generally, the CREATE INDEX statement creates an index space in the same DB2 database that contains the table on which the index is defined. This is true even if you defer building the index.

Exceptions:

- If you specify the USING VCAT clause, you create and manage the data sets yourself.
- If you specify the DEFINE NO clause on a CREATE INDEX statement that uses the USING STOGROUP clause, DB2 defers the allocation of the data sets for the index space.

GUIP To define your index space storage, complete the following steps:

1. Issue a CREATE INDEX statement.
2. Specify the USING clause.

When you specify USING, you declare whether you want DB2-managed or user-managed data sets. For DB2-managed data sets, you specify the primary and secondary space allocation parameters on the CREATE INDEX statement. If you do not specify USING, DB2 assigns the index data sets to the default storage groups using default space attributes.

You can specify the USING clause to allocate space for the entire index, or if the index is a partitioned index, you can allocate space for each partition.

GUIP

Information about space allocation for the index is kept in the SYSIBM.SYSINDEXPART table of the DB2 catalog. Other information about the index is in SYSIBM.SYSINDEXES.

Creating EA-enabled table spaces and index spaces

DFSMS has an extended-addressability function, which is necessary to create data sets that are larger than 4 GB. Therefore, the term for page sets that are enabled for extended addressability is *EA-enabled*.

You must use EA-enabled table spaces or index spaces if you specify a DSSIZE that is larger than 4 GB in the CREATE TABLESPACE statement.

To create EA-enabled page sets, you must:

1. Use SMS to manage the data sets that are associated with the EA-enabled page sets.
2. Associate the data sets with a *data class* (an SMS construct) that specifies the extended format and extended addressability options.

To make this association between data sets and the data class, use an automatic class selection (ACS) routine to assign the DB2 data sets to the relevant SMS data class. The ACS routine does the assignment based on the data set name. No performance penalty occurs for having non-EA-enabled DB2 page sets assigned to this data class, too, if you would rather not have two separate data classes for DB2.

For user-managed data sets, you can use ACS routines or specify the appropriate data class on the DEFINE CLUSTER command when you create the data set.

3. Create the partitioned or LOB table space with a DSSIZE of 8 GB or greater. The partitioning index for the partitioned table space takes on the EA-enabled attribute from its associated table space.

After a page set is created, you cannot use the ALTER TABLESPACE statement to change the DSSIZE. You must drop and re-create the table space.

Also, you cannot change the data sets of the page set to turn off the extended addressability or extended format attributes. If someone modifies the data class

to turn off the extended addressability or extended format attributes, DB2 issues an error message the next time that it opens the page set.

Implementing DB2 table spaces

DB2 table spaces are storage structures that store one or more data sets, which store one or more tables. You should understand the advantages and disadvantages of each type of table space in order to implement the table space that best suits your needs.

Creating a table space explicitly

Explicitly create a table space to define a segmented, partitioned, universal or LOB table space at the current server.

GUPI To explicitly create a table space, complete the following steps:

1. Issue a CREATE TABLESPACE SQL statement.
2. Specify the attributes of the table space. You can create segmented, partitioned, universal and LOB table spaces.

Tip: You can alter table spaces after they have been created, but the application of some statements, such as ALTER MAXPARTITIONS, prevent access to the database until they have finished. Consider future growth when defining new table spaces.

Example definition for a segmented table space: The following CREATE TABLESPACE statement creates a segmented table space with 32 pages in each segment:

```
CREATE TABLESPACE MYTS
  IN MYDB
  USING STOGROUP MYSTOGRP
  PRIQTY 30720
  SECQTY 10240
  SEGSIZE 32
  LOCKSIZE TABLE
  BUFFERPOOL BP0
  CLOSE NO;
```

Example definition for an EA-enabled partitioned table space: The following CREATE TABLESPACE statement creates an EA-enabled table space, SALESHX. Assume that a large query application uses this table space to record historical sales data for marketing statistics. The first USING clause establishes the MYSTOGRP storage group and space allocations for all partitions:

```
CREATE TABLESPACE SALESHX
  IN MYDB
  USING STOGROUP MYSTOGRP
  PRIQTY 4000
  SECQTY 130
  ERASE NO
  DSSIZE 16G
  NUMPARTS 48
  (PARTITION 46
   COMPRESS YES,
  PARTITION 47
   COMPRESS YES,
  PARTITION 48
```

```
        COMPRESS YES)
LOCKSIZE PAGE
BUFFERPOOL BP1
CLOSE NO;
```

Generally, when you use the CREATE TABLESPACE statement with the USING STOGROUP clause, DB2 allocates data sets for the table space. However, if you also specify the DEFINE NO clause, you can defer the allocation of data sets until data is inserted or loaded into a table in the table space.



Guidelines and recommendations for table spaces

This information provides guidelines and recommendations to help you with naming a table space, coding a table space, and determining the page size for a table space.

General naming guidelines for table spaces

A table space name is an identifier of up to eight characters. You can qualify a table space name with a database name.

- If you do not qualify an explicit table space with a database name, the default database name is database DSNDB04.
- If you do not explicitly specify a table space, DB2 implicitly creates the table space, where the name is derived based on the name of the table that is being created.
- If a database name is not explicitly specified for an implicit table space, and DB2 is operating in conversion mode, DB2 uses database DSNDB04.
- If DB2 is operating in new-function mode, DB2 either implicitly creates a new database for the table space, or uses an existing implicitly created database.

In conversion mode, the table space type for implicitly created table spaces is segmented. In new-function mode, the table space type is partition-by-growth.

A typical name is:

Object Name

Table space

MYDB.MYTS

Related reference

“Examples of table space definitions” on page 45

Coding guidelines for explicitly defined table spaces

You can use the CREATE TABLESPACE statement to create a table space explicitly. This statement lets you specify the attributes of the table space.

The following list introduces some of the clauses of the CREATE TABLESPACE statement that you will read about in this topic.

LOB

Indicates that the table space is to be a large object (LOB) table space.

DSSIZE

Indicates the maximum size, in GB, for each partition or, for LOB table spaces, for each data set.

FREEPAGE *integer*

Specifies how often DB2 is to leave a page of free space when the table space or partition is loaded or reorganized. You specify that DB2 is to set aside one free page for every *integer* number of pages. Using free pages can improve performance for applications that perform high-volume inserts or that update variable-length columns.

PCTFREE *integer*

Indicates the percentage (*integer*) of each page that DB2 should leave as free space when the table is loaded or reorganized. Specifying PCTFREE can improve performance for applications that perform high-volume inserts or that update variable-length columns.

COMPRESS

Specifies that data is to be compressed. You can compress data in a table space and thereby store more data on each data page.

BUFFERPOOL *bpname*

Identifies the buffer pool that this table space is to use and determines the page size of the table space. The buffer pool is a portion of memory in which DB2 temporarily stores data for retrieval.

LOCKSIZE

Specifies the size of locks that DB2 is to use within the table space. DB2 uses locks to protect data integrity. Use of locks results in some overhead processing costs, so choose the lock size carefully.

You can create segmented, partitioned, and LOB table spaces.

DB2 stores the names and attributes of all table spaces in the SYSIBM.SYSTABLESPACE catalog table, regardless of whether you define the table spaces explicitly or implicitly.

Related reference

“Examples of table space definitions” on page 45

Coding guidelines for implicitly defined table spaces

In new-function mode, DB2 implicitly creates a partition-by-growth table space for small tables when you use the CREATE TABLE statement to create a table and do not specify an existing table space name. In conversion mode, DB2 implicitly creates a segmented table space.

When DB2 defines a table space implicitly, DB2 performs the following tasks:

- Generates a table space for you.
- Derives a table space name from the name of your table.
- Uses default values for space allocation and other table space attributes.
- Creates the required LOB objects and XML objects.
- Enforces the UNIQUE constraint.
- Creates the primary key index.
- Creates the ROWID index, if the ROWID column is defined as GENERATED BY DEFAULT.

One or more tables are created for segmented table spaces.

You also need to create a table space when you define a declared temporary table.

DB2 stores the names and attributes of all table spaces in the SYSIBM.SYSTABLESPACE catalog table, regardless of whether you define the table spaces explicitly or implicitly.

Related reference

“Examples of table space definitions” on page 45

Recommendations for page size

DB2 provides many options for data page sizes.

The size of the data page is determined by the buffer pool in which you define the table space. For example, a table space that is defined in a 4-KB buffer pool has 4-KB page sizes, and one that is defined in an 8-KB buffer pool has 8-KB page sizes. (Indexes must be defined in a 4-KB buffer pool.)

Data in table spaces is stored and allocated in 4-KB record segments. Thus, an 8-KB page size means two 4-KB records, and a 32-KB page size means eight 4-KB records. A good starting point is to use the default of 4-KB page sizes when access to the data is random and only a few rows per page are needed. If row sizes are very small, using the 4-KB page size is recommended.

However, there are situations in which larger page sizes are needed or recommended:

- **When the size of individual rows is greater than 4 KB.** In this case, you must use a larger page size. When considering the size of work file table spaces, remember that some SQL operations, such as joins, can create a result row that does not fit in a 4-KB page. Therefore, having at least one work file that has 32-KB pages is recommended. (Work files cannot use 8-KB or 16-KB pages.)
- **When you can achieve higher density on disk by choosing a larger page size.** For example, only one 2100-byte record can be stored in a 4-KB page, which wastes almost half of the space. However, storing the record in a 32-KB page can significantly reduce this waste. The downside with this approach is the potential of incurring higher buffer pool storage costs or higher I/O costs—if you only touch a small number of rows, you are bringing a bigger chunk of data from disk into the buffer pool.

Using 8-KB or 16-KB page sizes can let you store more data on your disk with less impact on I/O and buffer pool storage costs. If you use a larger page size and access is random, you might need to go back and increase the size of the buffer pool to achieve the same read-hit ratio you do with the smaller page size.

- **When a larger page size can reduce data sharing overhead.** One way to reduce the cost of data sharing is to reduce the number of times the coupling facility must be accessed. Particularly for sequential processing, larger page sizes can reduce this number. More data can be returned on each access of the coupling facility, and fewer locks must be taken on the larger page size, further reducing coupling facility interactions.

If data is returned from the coupling facility, each access that returns more data is more costly than those that return smaller amounts of data, but, because the total number of accesses is reduced, coupling facility overhead is reduced.

For random processing, using an 8-KB or 16-KB page size instead of a 32-KB page size might improve the read-hit ratio to the buffer pool and reduce I/O resource consumption.

The maximum number of partitions for a table space depends on the page size and on the DSSIZE. The size of the table space depends on how many partitions are in the table space and on the DSSIZE.

Related reference

“Examples of table space definitions” on page 45

Factors for determining table space page size

DB2 provides many options for data page sizes. The size of the data page is determined by the buffer pool in which you define the table space.

For example, a table space that is defined in a 4-KB buffer pool has 4-KB page sizes, and one that is defined in an 8-KB buffer pool has 8-KB page sizes.

Data in table spaces is stored and allocated in record segments. Any record segment can be 4 KB in size, or it can be or the size determined by the buffer pool (4 KB, 8, KB, 16 KB, or 32 KB). In a table space with 4-KB record segments, an 8-KB page size requires two 4-KB records, and a 32-KB page size requires eight 4-KB records. A good starting point is the default of 4-KB page sizes when access to the data is random and only a few rows per page are needed. If row sizes are very small, using the 4-KB page size is recommended.

However, in some situations, larger page sizes are needed or recommended:

- **When the size of individual rows is greater than 4 KB.** In this case, you must use a larger page size. When considering the size of work file table spaces, remember that some SQL operations, such as joins, can create a result row that does not fit in a 4-KB page. Therefore, having at least one work file that has 32-KB pages is recommended. (Work files cannot use 8-KB or 16-KB pages.)
- **When you can achieve higher density on disk by choosing a larger page size.** For example, only one 2100-byte record can be stored in a 4-KB page, which wastes almost half of the space. However, storing the record in a 32-KB page can significantly reduce this waste. The disadvantage of this approach is the potential of incurring higher buffer pool storage costs or higher I/O costs; if you touch only a small number of rows, you bring a larger set of data from disk into the buffer pool.

Using 8-KB or 16-KB page sizes can let you store more data on your disk with less impact on I/O and buffer pool storage costs. If you use a larger page size and access is random, you might need to increase the size of the buffer pool to achieve the same read-hit ratio that you do with the smaller page size.

- **When a larger page size can reduce data sharing overhead.** One way to reduce the cost of data sharing is to reduce the number of times that the coupling facility must be accessed. Particularly for sequential processing, larger page sizes can reduce this number. More data can be returned on each access of the coupling facility, and fewer locks need to be taken on the larger pages, which reduces coupling facility interactions.

If data is returned from the coupling facility, each access that returns more data is more costly than those that return smaller amounts of data. However, because the total number of accesses is reduced, coupling facility overhead is reduced.

For random processing, using an 8-KB or 16-KB page size instead of a 32-KB page size might improve the read-hit ratio to the buffer pool and reduce I/O resource consumption.

The maximum number of partitions for a table space depends on the page size and on the DSSIZE. The size of the table space depends on how many partitions are in the table space and on the DSSIZE. The maximum number of partitions for a partition-by-growth table space depends on the value that is specified for the MAXPARTITIONS option of the CREATE TABLESPACE or ALTER TABLESPACE statement.

Related reference

“Examples of table space definitions” on page 45

Recommendations for LOB page size

Choosing a page size for LOBs (in the LOB table space) is a tradeoff between minimizing the number of getpages (maximizing performance) and not wasting space.

With LOB table spaces, no more than one LOB value is ever stored in a given page in a LOB table space. Space that is not used by the LOB value in the last page that is occupied by the LOB remains unused. DB2 also uses additional space for control information. The smaller the LOB, the greater the proportion of space for this “non-data” is used.

For example, if you have a 17-KB LOB, the 4-KB page size is the most efficient for storage. A 17-KB LOB requires five 4-KB pages for a total of 20 KB of storage space. Pages that are 8 KB, 16 KB, and 32 KB in size waste more space, because they require 24 KB, 32 KB, and 32 KB, respectively, for the LOB.

The following table shows that the number of data pages is lower for larger page sizes, but larger page sizes might have more unused space.

Table 6. Relationship between LOB size and data pages based on page size

LOB size	Page size	LOB data pages	% Non-LOB data or unused space
262 144 bytes	4 KB	64	1.6
	8 KB	32	3.0
	16 KB	16	5.6
	32 KB	8	11.1
4 MB	4 KB	1029	0.78
	8 KB	513	0.39
	16 KB	256	0.39
	32 KB	128	0.78
33 MB	4 KB	8234	0.76
	8 KB	4106	0.39
	16 KB	2050	0.19
	32 KB	1024	0.10

Choosing a page size based on average LOB size

If you know that all of your LOBs are not the same size, you can still make an estimate of what page size to choose. To estimate the average size of a LOB, you need to add a percentage to account for unused space and control information. To estimate the average size of a LOB value, use the following formula:

$$\text{LOB size} = (\text{average LOB length}) \times 1.05$$

The following table contains some suggested page sizes for LOBs with the intent to reduce the amount of I/O (getpages).

Table 7. Suggested page sizes based on average LOB length

Average LOB size (n)	Suggested page size
$n \leq 4$ KB	4 KB
$4 \text{ KB} < n \leq 8$ KB	8 KB
$8 \text{ KB} < n \leq 16$ KB	16 KB
$16 \text{ KB} < n$	32 KB

The estimates in the previous table mean that a LOB value of 17 KB can mean 15 KB of unused space. Again, you must analyze your data to determine what is best.

General guidelines for LOBs of same size

If your LOBs are all the same size, you can fairly easily choose a page size that uses space efficiently without sacrificing performance. For LOBs that are all the same size, consider the alternative in the following table to maximize your space savings.

Table 8. Suggested page sizes when LOBs are the same size

LOB size (y)	Suggested page size
$y \leq 4$ KB	4 KB
$4 \text{ KB} < y \leq 8$ KB	8 KB
$8 \text{ KB} < y \leq 12$ KB	4 KB
$12 \text{ KB} < y \leq 16$ KB	16 KB
$16 \text{ KB} < y \leq 24$ KB	8 KB
$24 \text{ KB} < y \leq 32$ KB	32 KB
$32 \text{ KB} < y \leq 48$ KB	16 KB
$48 \text{ KB} < y$	32 KB

Related reference

“Examples of table space definitions” on page 45

Factors for determining LOB table space page size

Choosing a page size for LOBs (in the LOB table space) is a trade-off between minimizing the number of getpages (maximizing performance) and not wasting space.

With LOB table spaces, no more than one LOB value is ever stored in a given page in a LOB table space. Space that is not used by the LOB value in the last page that is occupied by the LOB remains unused. DB2 also uses additional space for control information. The smaller the LOB is, the greater the proportion of space is for this “non-data”.

Example: If you have a 17-KB LOB, the 4-KB page size is the most efficient for storage. A 17-KB LOB requires five 4-KB pages for a total of 20 KB of storage space. Pages that are 8 KB, 16 KB, and 32 KB in size result in more wasted space because they require 24 KB, 32 KB, and 32 KB, respectively, for the LOB.

The following table shows that the number of data pages is lower for larger page sizes, but larger page sizes might have more unused space.

Table 9. Relationship between LOB size and data pages based on page size

LOB size	Page size	LOB data pages	% Non-LOB data or unused space
262 144 bytes	4 KB	64	1.6
	8 KB	32	3.0
	16 KB	16	5.6
	32 KB	8	11.1
4 MB	4 KB	1029	0.78
	8 KB	513	0.39
	16 KB	256	0.39
	32 KB	128	0.78
33 MB	4 KB	8234	0.76
	8 KB	4106	0.39
	16 KB	2050	0.19
	32 KB	1024	0.10

Choosing a page size based on the average LOB size

If you know that all of your LOBs are not the same size, you can use the average LOB size to choose the best page size. To estimate the average size of a LOB, you need to add a percentage to account for unused space and control information. To estimate the average size of a LOB value, use the following formula:

$$\text{LOB size} = (\text{average LOB length}) \times 1.05$$

The following table suggests page sizes for LOBs with the intent to reduce the amount of I/O (getpages).

Table 10. Suggested page sizes based on average LOB length

Average LOB size (n)	Suggested page size
$n \leq 4 \text{ KB}$	4 KB
$4 \text{ KB} < n \leq 8 \text{ KB}$	8 KB
$8 \text{ KB} < n \leq 16 \text{ KB}$	16 KB
$16 \text{ KB} < n$	32 KB

The estimates in the previous table mean that a LOB value of 17 KB can mean 15 KB of unused space. Again, you need to analyze your data to determine what is best.

General guidelines for LOBs of the same size

If your LOBs are all the same size, you can fairly easily choose a page size that uses space efficiently without sacrificing performance. For LOBs that are all the same size, consider the alternatives presented in the following table to maximize your space savings.

Table 11. Suggested page sizes when LOBs are the same size

LOB size (y)	Suggested page size
$y \leq 4 \text{ KB}$	4 KB

Table 11. Suggested page sizes when LOBs are the same size (continued)

LOB size (y)	Suggested page size
4 KB < y ≤ 8 KB	8 KB
8 KB < y ≤ 12 KB	4 KB
12 KB < y ≤ 16 KB	16 KB
16 KB < y ≤ 24 KB	8 KB
24 KB < y ≤ 32 KB	32 KB
32 KB < y ≤ 48 KB	16 KB
48 KB < y	32 KB

Related reference

“Examples of table space definitions”

Examples of table space definitions

When you define a table space, referring to examples of different types of table space definitions can be helpful.

This information describes several examples of table space definitions. The following clauses are used in the examples:

IN Identifies the database in which DB2 is to create the table space. If this clause is not specified, the default database, DSNDB04, is used.

USING STOGROUP

Indicates that you want DB2 to define and manage the data sets for this table space. If you specify the DEFINE NO clause, you can defer allocation of data sets until data is inserted or loaded into a table in the table space.

PRIQTY *integer*

Specifies the minimum primary space allocation for a DB2-managed data set. This parameter applies only to table spaces that use storage groups. The *integer* represents the number of kilobytes.

SECQTY *integer*

Specifies the minimum secondary space allocation for a DB2-managed data set. This parameter applies only to table spaces that use storage groups. The *integer* represents the number of kilobytes.

Example definition for a segmented table space

The following CREATE TABLESPACE statement creates a segmented table space with 32 pages in each segment:

```
CREATE TABLESPACE MYTS
  IN MYDB
  USING STOGROUP MYSTOGRP
  PRIQTY 30720
  SECQTY 10240
  SEGSIZE 32
  LOCKSIZE TABLE
  BUFFERPOOL BP0
  CLOSE NO;
```

Example definition for an EA-enabled partitioned table space

The following CREATE TABLESPACE statement creates an EA-enabled table space, SALESXH. Assume that a large query application uses this table space to record historical sales data for marketing statistics. The first USING clause establishes the MYSTOGRP storage group and space allocations for all partitions:

```
CREATE TABLESPACE SALESXH
  IN MYDB
  USING STOGROUP MYSTOGRP
    PRIQTY 4000
    SECQTY 130
    ERASE NO
  DSSIZE 16G
  NUMPARTS 48
  (PARTITION 46
    COMPRESS YES,
  PARTITION 47
    COMPRESS YES,
  PARTITION 48
    COMPRESS YES)
  LOCKSIZE PAGE
  BUFFERPOOL BP1
  CLOSE NO;
```

Example definitions for a partition-by-growth universal table space

The following examples show how to create a partition-by-growth universal table space. 

Example 1: In the following SQL statement, the universal table space is implicitly created by a CREATE TABLE statement.

```
CREATE TABLE TEST02TB(
  C1 SMALLINT,
  C2 DECIMAL(9,2),
  C3 CHAR(4))
PARTITIONING BY SIZE EVERY 4G
IN TEST02DB;
COMMIT;
```

Example 2: In the following SQL statement, the partition-by-growth universal table space has a maximum size of 2 GB for each partition, 4 pages per segment, with a maximum of 24 partitions for table space.

```
CREATE TABLESPACE TEST01TS IN TEST01DB USING STOGROUP SG1
DSSIZE 2G
MAXPARTITIONS 24
LOCKSIZE ANY
SEGSIZE 4;
COMMIT;
```



Example definitions for a range-partitioned universal table space

The following examples show how to create a range-partitioned universal table space (UTS). 

Example 1: The following SQL statement defines a range-partitioned universal table space with 16 pages per segment and 55 partitions. This universal table space uses a storage group SG1 and has LOCKSIZE ANY.

```
CREATE TABLESPACE TS1 IN DB1 USING STOGROUP SG1
NUMPARTS 55 SEGSIZE 16
LOCKSIZE ANY;
```

Example 2: The following SQL statement defines a range-partitioned universal table space with 64 pages per segment and 7 defer-defined partitions. This universal table space uses a storage group SG1 and compresses every odd-numbered partition.

```
CREATE TABLESPACE TS1 IN DB1 USING STOGROUP SG1
NUMPARTS 7
(
PARTITION 1 COMPRESS YES,
PARTITION 3 COMPRESS YES,
PARTITION 5 COMPRESS YES,
PARTITION 7 COMPRESS YES
)
SEGSIZE 64
DEFINE NO;
```

GUPI

Related concepts

- “General naming guidelines for table spaces” on page 38
- “Coding guidelines for explicitly defined table spaces” on page 38
- “Coding guidelines for implicitly defined table spaces” on page 39
- “Recommendations for page size” on page 40
- “Factors for determining table space page size” on page 41
- “Recommendations for LOB page size” on page 42
- “Factors for determining LOB table space page size” on page 43

Implementing DB2 tables

Use the columns and rows of DB2 tables as logical structures for storing data.

Related reference

“Creation of tables” (Introduction to DB2)

Creating base tables

When you create a table, DB2 records a definition of the table in the DB2 catalog.

Creating a table does not store the application data. You can put data into the table by using several methods, such as the LOAD utility or the INSERT statement.

To create a base table that you designed:

Issue the CREATE TABLE statement.

Example: The following CREATE TABLE statement creates the EMP table, which is in a database named MYDB and in a table space named MYTS:

```
CREATE TABLE EMP
(EMPNO      CHAR(6)          NOT NULL,
FIRSTNAME  VARCHAR(12)     NOT NULL,
LASTNAME   VARCHAR(15)     NOT NULL,
```

```

DEPT      CHAR(3)           ,
HIREDATE  DATE           ,
JOB       CHAR(8)        ,
EDL       SMALLINT      ,
SALARY    DECIMAL(9,2)   ,
COMM      DECIMAL(9,2)   ,
PRIMARY KEY (EMPNO))
IN MYDB.MYTS;

```

This CREATE TABLE statement shows the definition of multiple columns.

Guidelines for table names

Most organizations have naming conventions to ensure that objects are named in a consistent manner. This information describes the basic requirements for table names.

The table name is an identifier of up to 128 characters. You can qualify the table name with an SQL identifier, which is a schema. When you define a table that is based directly on an entity, these factors also apply to the table names.

Creating temporary tables

Temporary tables are useful when you need to sort or query intermediate result tables that contain large numbers of rows and identify a small subset of rows to store permanently. The two types of temporary tables are created temporary tables and declared temporary tables.

To create a temporary table:

1. Determine the type of temporary table that you want to create.
 - If you need a permanent, sharable description of a table but need to store data only for the life of an application process, define and use a created temporary table.
 - If you need to store data for the life of an application process, but you don't need a permanent, sharable description of the table, define and use a declared temporary table.
2. Issue the appropriate SQL statement for the type of temporary table that you want to create.
 - To define a created temporary table, issue the CREATE GLOBAL TEMPORARY TABLE statement. For more information, see "Creating created temporary tables."
 - To define a declared temporary table, issue the DECLARE GLOBAL TEMPORARY TABLE statement. For more information, see "Creating declared temporary tables" on page 49.

Creating created temporary tables

If you need a permanent, sharable description of a table but need to store data only for the life of an application process, you can define and use a created temporary table.

DB2 does not log operations that it performs on created temporary tables; therefore, SQL statements that use created temporary tables can execute more efficiently. Each application process has its own instance of the created temporary table.

To create a created temporary table:

Issue the CREATE GLOBAL TEMPORARY TABLE statement.

Example: The following statement defines a created temporary table that is named TEMPPROD.

```
CREATE GLOBAL TEMPORARY TABLE TEMPPROD
(SERIALNO CHAR(8) NOT NULL,
DESCRIPTION VARCHAR(60) NOT NULL,
MFGCOSTAMT DECIMAL(8,2) ,
MFGDEPTNO CHAR(3) ,
MARKUPPCT SMALLINT ,
SALESDEPTNO CHAR(3) ,
CURDATE DATE NOT NULL);
```

Creating declared temporary tables

If you need to store data for the life of an application process, but you don't need a permanent, sharable description of the table, you can define and use a declared temporary table.

To create a declared temporary table:

Issue the DECLARE GLOBAL TEMPORARY TABLE statement. Unlike other DB2 DECLARE statements, DECLARE GLOBAL TEMPORARY TABLE is an executable statement that you can embed in an application program or issue interactively. You can also dynamically prepare the statement.

When a program in an application process issues a DECLARE GLOBAL TEMPORARY TABLE statement, DB2 creates an empty instance of the table. You can populate the declared temporary table by using INSERT statements, modify the table by using searched or positioned UPDATE or DELETE statements, and query the table by using SELECT statements. You can also create indexes on the declared temporary table. The definition of the declared temporary table exists as long as the application process runs.

At the end of an application process that uses a declared temporary table, DB2 deletes the rows of the table and implicitly drops the description of the table.

Example: The following statement defines a declared temporary table, TEMP_EMP. (This example assumes that you have already created the WORKFILE database and corresponding table space for the temporary table.)

```
DECLARE GLOBAL TEMPORARY TABLE SESSION.TEMP_EMP
(EMPNO CHAR(6) NOT NULL,
SALARY DECIMAL(9, 2) ,
COMM DECIMAL(9, 2));
```

If specified explicitly, the qualifier for the name of a declared temporary table, must be SESSION. If the qualifier is not specified, it is implicitly defined to be SESSION.

Distinctions between DB2 base tables and temporary tables

DB2 base tables and the two types of temporary tables have several distinctions.

The following table summarizes important distinctions between base tables, created temporary tables, and declared temporary tables.

Table 12. Important distinctions between DB2 base tables and DB2 temporary tables

Area of distinction	Base tables	Created temporary tables	Declared temporary tables
Creation, persistence, and ability to share table descriptions	<p>CREATE TABLE statement puts a description of the table in catalog table SYSTABLES. The table description is persistent and is shareable across application processes.</p> <p>The name of the table in the CREATE statement can be a two-part or three-part name. If the table name is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements.</p>	<p>CREATE GLOBAL TEMPORARY TABLE statement puts a description of the table in catalog table SYSTABLES. The table description is persistent and is shareable across application processes.</p> <p>The name of the table in the CREATE statement can be a two-part- or three-part name. If the table name is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules applied to the SQL statements.</p> <p>The table space that is used by created temporary tables is reset by the following commands: START DB2, START DATABASE, and START DATABASE(<i>dbname</i>) SPACNAM(<i>tsname</i>), where <i>dbname</i> is the name of the database and <i>tsname</i> is the name of the table space.</p>	<p>DECLARE GLOBAL TEMPORARY TABLE statement does not put a description of the table in catalog table SYSTABLES. The table description is not persistent beyond the life of the application process that issued the DECLARE statement and the description is known only to that application process. Thus, each application process could have its own possibly unique description of the same table.</p> <p>The name of the table in the DECLARE statement can be a two-part or three-part name. If the table name is qualified, SESSION must be used as the qualifier for the owner (the second part in a three-part name). If the table name is not qualified, DB2 implicitly uses SESSION as the qualifier.</p> <p>The table space used by declared temporary tables is reset by the following commands: START DB2, START DATABASE, and START DATABASE(<i>dbname</i>) SPACNAM(<i>tsname</i>), where <i>dbname</i> is the name of the database and <i>tsname</i> is the name of the table space.</p>
Table instantiation and ability to share data	<p>CREATE TABLE statement creates one empty instance of the table, and all application processes use that one instance of the table. The table and data are persistent.</p>	<p>CREATE GLOBAL TEMPORARY TABLE statement does not create an instance of the table. The first implicit or explicit reference to the table in an OPEN, SELECT, INSERT, or DELETE operation that is executed by any program in the application process creates an empty instance of the given table. Each application process has its own unique instance of the table, and the instance is not persistent beyond the life of the application process.</p>	<p>DECLARE GLOBAL TEMPORARY TABLE statement creates an empty instance of the table for the application process. Each application process has its own unique instance of the table, and the instance is not persistent beyond the life of the application process.</p>

Table 12. Important distinctions between DB2 base tables and DB2 temporary tables (continued)

Area of distinction	Base tables	Created temporary tables	Declared temporary tables
References to the table in application processes	References to the table name in multiple application processes refer to the same single persistent table description and to the same instance at the current server. If the table name that is being referenced is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules that apply to the SQL statements. The name can be a two-part- or three-part name.	References to the table name in multiple application processes refer to the same single persistent table description but to a distinct instance of the table for each application process at the current server. If the table name that is being referenced is not qualified, DB2 implicitly qualifies the name using the standard DB2 qualification rules that apply to the SQL statements. The name can be a two-part or three-part name.	References to that table name in multiple application processes refer to a distinct description and instance of the table for each application process at the current server. References to the table name in an SQL statement (other than the DECLARE GLOBAL TEMPORARY TABLE statement) must include SESSION as the qualifier (the first part in a two-part table name or the second part in a three-part name). If the table name is not qualified with SESSION, DB2 assumes the reference is to a base table.
Table privileges and authorization	The owner implicitly has all table privileges on the table and the authority to drop the table. The owner's table privileges can be granted and revoked, either individually or with the ALL clause. Another authorization ID can access the table only if it has been granted appropriate privileges for the table.	The owner implicitly has all table privileges on the table and the authority to drop the table. The owner's table privileges can be granted and revoked, but only with the ALL clause; individual table privileges cannot be granted or revoked. Another authorization ID can access the table only if it has been granted ALL privileges for the table.	PUBLIC implicitly has all table privileges on the table without GRANT authority and has the authority to drop the table. These table privileges cannot be granted or revoked. Any authorization ID can access the table without a grant of any privileges for the table.
Indexes and other SQL statement support	Indexes and SQL statements that modify data (INSERT, UPDATE, DELETE, and so on) are supported.	Indexes, UPDATE (searched or positioned), and DELETE (positioned only) are not supported.	Indexes and SQL statements that modify data (INSERT, UPDATE, DELETE, and so on) are supported.
Locking, logging, and recovery	Locking, logging, and recovery do apply.	Locking, logging, and recovery do not apply. Work files are used as the space for the table.	Some locking, logging, and limited recovery do apply. No row or table locks are acquired. Share-level locks on the table space and DBD are acquired. A segmented table lock is acquired when all the rows are deleted from the table or the table is dropped. Undo recovery (rolling back changes to a savepoint or the most recent commit point) is supported, but redo recovery (forward log recovery) is not supported.
Table space and database operations	Table space and database operations do apply.	Table space and database operations do not apply.	Table space and database operations do apply.

Table 12. Important distinctions between DB2 base tables and DB2 temporary tables (continued)

Area of distinction	Base tables	Created temporary tables	Declared temporary tables
Table space requirements and table size limitations	<p>The table can be stored in implicitly created table spaces and databases.</p> <p>The table cannot span table spaces. Therefore, the size of the table is limited by the table space size (as determined by the primary and secondary space allocation values that are specified for the table space's data sets) and the shared usage of the table space among multiple users. When the table space is full, an error occurs for the SQL operation.</p>	<p>The table is stored in table spaces in the work file database.</p> <p>The table can span work file table spaces. Therefore, the size of the table is limited by the number of available work file table spaces, the size of each table space, and the number of data set extents that are allowed for the table spaces. Unlike the other types of tables, created temporary tables do not reach size limitations as easily.</p>	<p>The table is stored in a table space in the work file database.</p> <p>The table cannot span table spaces. Therefore, the size of the table is limited by the table space size (as determined by the primary and secondary space allocation values that are specified for the table space's data sets) and the shared usage of the table space among multiple users. When the table space is full, an error occurs for the SQL operation.</p>

You can find additional examples of implementing temporary tables and information about restrictions and extensions of temporary tables in the *DB2 Application Programming and SQL Guide* and in the *DB2 SQL Reference*. For information about temporary tables and their impact on DB2 resources, see the *DB2 Performance Guide*.

Creating materialized query tables

Materialized query tables improve the performance of complex queries that operate on very large amounts of data.

DB2 uses a materialized query table to precompute the results of data that is derived from one or more tables. When you submit a query, DB2 can use the results that are stored in a materialized query table rather than compute the results from the underlying source tables on which the materialized query table is defined.

To create a new materialized query table:

Issue the CREATE TABLE statement.

Example: The following CREATE TABLE statement defines a materialized query table named TRANSCNT. TRANSCNT summarizes the number of transactions in table TRANS by account, location, and year.

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

The fullselect, together with the DATA INITIALLY DEFERRED clause and the REFRESH DEFERRED clause, defines the table as a materialized query table.

Creating tables that use table-controlled partitioning

Table-controlled partitioning does not require an index for partitioning and is defined by PARTITION clauses on the CREATE TABLE statement.

To create a table that uses table-controlled partitioning:

Specify the partitioning key and the limit key values for a table in a partitioned table space by using the PARTITION BY clause and the PARTITION ENDING AT clause of the CREATE TABLE statement.

Example: Assume that you need to create a large transaction table that includes the date of the transaction in a column named POSTED. You want the transactions for each month in a separate partition. To create the table, issue the following statement:

```
CREATE TABLE TRANS
  (ACCTID ...,
   STATE ...,
   POSTED ...,
   ... , ...)
PARTITION BY (POSTED)
(PARTITION 1 ENDING AT ('01/31/2003'),
 PARTITION 2 ENDING AT ('02/28/2003'),
 ...
 PARTITION 13 ENDING AT ('01/31/2004'));
```

Differences between partitioning methods

Use table-controlled partitioning instead of index-controlled partitioning. Table-controlled partitioning will eventually replace index-controlled partitioning.

GUIP When you define a partitioning index on a table in a partitioned table space, you specify the partitioning key and the limit key values in the PARTITION clause of the CREATE INDEX statement. This type of partitioning is called *index-controlled partitioning*.

DB2 also supports a method called *table-controlled partitioning* for defining table partitions. You can use table-controlled partitioning instead of index-controlled partitioning. With table-controlled partitioning, you can specify the partitioning key and the limit key values for a table in a partitioned table space by using the PARTITION BY clause and the PARTITION ENDING AT clause of the CREATE TABLE statement. When you use this type of partitioning, an index is not required for partitioning.

Restriction: If you use table-controlled partitioning, you cannot specify the partitioning key and the limit key values by using the PARTITION clause of the CREATE INDEX statement. (Note that the preferred syntax changed from PARTITION to PARTITION ENDING AT.) **GUIP**

The following table lists the differences between the two partitioning methods.

Table 13. Differences between table-controlled and index-controlled partitioning

Table-controlled partitioning	Index-controlled partitioning
A partitioning index is not required; clustering index is not required.	A partitioning index is required; clustering index is required.
Multiple partitioned indexes can be created in a table space.	Only one partitioned index can be created in a table space.

Table 13. Differences between table-controlled and index-controlled partitioning (continued)

Table-controlled partitioning	Index-controlled partitioning
A table space partition is identified by both a physical partition number and a logical partition number.	A table space partition is identified by a physical partition number.
The high-limit key is always enforced.	The high-limit key is not enforced if the table space is non-large.

Automatic conversion to table-controlled partitioning

Some index operations cause DB2 to automatically convert an index-controlled partitioned table space to a table-controlled partitioned table space.

- **GUPI**
 - Use CREATE INDEX with the PARTITIONED clause to create a partitioned index on an index-controlled partitioned table space.
 - Use CREATE INDEX with a PART VALUES clause and without a CLUSTER clause to create a partitioning index.
DB2 stores the specified high limit key value instead of the default high limit key value.
 - Use ALTER INDEX with the NOT CLUSTER clause on a partitioning index that is on an index-controlled partitioned table space.
 - Use DROP INDEX to drop a partitioning index on an index-controlled partitioned table space.
 - Use ALTER TABLE to add a new partition, change a partition boundary, or rotate a partition from first to last on an index-controlled partitioned table space.
In these cases, DB2 automatically converts to table-controlled partitioning but does not automatically drop any indexes. DB2 assumes that any existing indexes are useful.

GUPI

After the conversion to table-controlled partitioning, DB2 changes the existing high-limit key value for non-large table spaces to the highest value for the key. Beginning in Version 8, DB2 enforces the high-limit key value. By default, DB2 does not put the last partition of the table space into a REORG-pending (REORP) state. Exceptions to this rule are:

- When adding a new partition, DB2 stores the original high-limit key value instead of the default high-limit key value. If this value was not previously enforced, DB2 puts the last partition into a REORP state.
- When rotating a new partition, DB2 stores the original high-limit key value instead of the default high-limit key value. DB2 puts the last partition into a REORP state.

After the conversion to table-controlled partitioning, the SQL statement that created the partitioning index is no longer valid. For example, after dropping a partitioning index on an index-controlled partitioned table space, an attempt to re-create the index by issuing the same CREATE INDEX statement that you originally used would fail because the boundary partitions are now under the control of the table.

Nullable partitioning columns

DB2 lets you use nullable columns as partitioning columns. The use of nullable columns has different implications for table-controlled partitioning than for index-controlled partitioning.

With table-controlled partitioning, DB2 can restrict the insertion of null values into a table with nullable partitioning columns, depending on the order of the partitioning key:

- If the partitioning key is ascending, DB2 prevents the INSERT of a row with a null value for the key column.
- If the partitioning key is descending, DB2 allows the INSERT of a row with a null value for the key column. The row is inserted into the first partition.

Example 1:  Assume that a partitioned table space is created with the following SQL statements:

```
CREATE TABLESPACE TS IN DB
  USING STOGROUP SG
  Numparts 4 BUFFERPOOL BP0;

CREATE TABLE TB (C01 CHAR(5),
  C02 CHAR(5) NOT NULL,
  C03 CHAR(5) NOT NULL)
  IN DB.TS
  PARTITION BY (C01)
  PARTITION 1 ENDING AT ('10000'),
  PARTITION 2 ENDING AT ('20000'),
  PARTITION 3 ENDING AT ('30000'),
  PARTITION 4 ENDING AT ('40000');
```

Because the CREATE TABLE statement does not specify the order in which to put entries, DB2 puts them in ascending order by default. DB2 subsequently prevents any INSERT into the TB table of a row with a null value for partitioning column C01. If the CREATE TABLE statement had specified the key as descending, DB2 would subsequently have allowed an INSERT into the TB table of a row with a null value for partitioning column C01. DB2 would have inserted the row into partition 1.

With index-controlled partitioning, DB2 does not restrict the insertion of null values into a value with nullable partitioning columns.

Example 2: Assume that a partitioned table space is created with the following SQL statements:

```
CREATE TABLESPACE TS IN DB
  USING STOGROUP SG
  Numparts 4 BUFFERPOOL BP0;

CREATE TABLE TB (C01 CHAR(5),
  C02 CHAR(5) NOT NULL,
  C03 CHAR(5) NOT NULL)
  IN DB.TS;

CREATE INDEX PI ON TB(C01) CLUSTER
  (PARTITION 1 ENDING AT ('10000'),
  PARTITION 2 ENDING AT ('20000'),
  PARTITION 3 ENDING AT ('30000'),
  PARTITION 4 ENDING AT ('40000'));
```

Regardless of the entry order, DB2 allows an INSERT into the TB table of a row with a null value for partitioning column C01. If the entry order is ascending, DB2

inserts the row into partition 4; if the entry order is descending, DB2 inserts the row into partition 1. Only if the table space is created with the LARGE keyword does DB2 prevent the insertion of a null value into the C01 column. 

Creating tables that use index-controlled partitioning

Index-controlled partitioning requires an index and is defined by the PARTITION clause of the CREATE INDEX statement.

Because the index is created separately from the associated table, you cannot insert data into the table until the partitioning index is created.

To define a partitioning index on a table in a partitioned table space:

Specify the partitioning key and the limit key values in the PARTITION clause of the CREATE INDEX statement.

Creating a clone table

You can create a clone table on an existing base table at the current server by using the ALTER TABLE statement.

Although ALTER TABLE syntax is used to create a clone table, the authorization granted as part of the clone creation process is the same as you would get during regular CREATE TABLE processing. The Schema for the clone table will be the same as for the base table.

Restriction: You can create a clone table only if the base table is in a universal table space.

 To create a clone table, complete the following steps:

Issue an ALTER TABLE statement with the ADD CLONE option.

For example,

```
ALTER TABLE base-table-name ADD CLONE clone-table-name
```

The creation or drop of a clone table does not impact applications accessing base table data. No base object quiesce is necessary and this process does not invalidate plans, packages, or the dynamic statement cache.

Restrictions:

- A clone table uses the statistics from the base table. RUNSTATS does not collect statistics on a clone table, and Access Path Selection (APS) does not use RUNSTATS statistics when accessing a clone table. This is in contrast to Real-time Statistics which keeps statistics for both the base and clone objects.
- Catalog and directory tables cannot be cloned.
- Indexes cannot be created on the clone table. Indexes can be created on the base table but not on the clone table. Indexes created on the base table apply to both the base and clone tables.
- Before triggers cannot be created on the clone table. Before triggers can be created on the base table but not on the clone table. Before triggers created on the base table apply to both the base and clone tables.
- You cannot rename a base table that has a clone relationship.

- You cannot clone an RTS table.
- You cannot drop an AUX table or AUX index on an object involved in cloning.
- You cannot alter any table or column attributes of a base table or clone table when the objects are involved with cloning.
- The maximum number of partitions cannot be altered when a clone table resides in a partition-by-growth table space. **GUIP**

Exchanging base table and clone table data

You can exchange the base and clone data by using the EXCHANGE statement.

GUIP To exchange table and index data between the base table and clone table:

Issue an EXCHANGE statement with the DATA BETWEEN TABLE *table-name1* AND *table-name2* syntax.

For example,

```
EXCHANGE DATA BETWEEN TABLE table-name1 AND table-name2
```

GUIP

After a data exchange, the base and clone table names remain the same as they were prior to the data exchange. No data movement actually takes place. The instance numbers in the underlying VSAM data sets for the objects (tables and indexes) do change, and this has the effect of changing the data that appears in the base and clone tables and their indexes. For example, a base table exists with the data set name *I0001.*. The table is cloned and the clone's data set is initially named *.I0002.*. After an exchange, the base objects are named *.I0002.* and the clones are named *I0001.*. Each time that an exchange happens, the instance numbers that represent the base and the clone objects change, which immediately changes the data contained in the base and clone tables and indexes.

Implementing DB2 views

When you design your database, you might need to give users access to only certain pieces of data. You can give users access by designing and using views.

You can use views to perform the following tasks:

- Control access to a table
- Make data easier to use
- Simplify authorization by granting access to a view without granting access to the table
- Show only portions of data in the table
- Show summary data for a given table
- Combine two or more tables in meaningful ways

Creating DB2 views

You can create a view on tables or other views at the current server.

Prerequisite: Before you create different column names for your view, remember the naming conventions that you established when designing the relational database.

GUIP To define a view, complete the following step:

Issue the CREATE VIEW SQL statement.

Unless you specifically list different column names after the view name, the column names of the view are the same as those of the underlying table. **GUIP**

Example 1: Defining a view on a single table Assume that you want to create a view on the DEPT table. Of the four columns in the table, the view needs only three: DEPTNO, DEPTNAME, and MGRNO. The order of the columns that you specify in the SELECT clause is the order in which they appear in the view:

GUIP

```
CREATE VIEW MYVIEW AS
  SELECT DEPTNO,DEPTNAME,MGRNO
  FROM DEPT;
```

GUIP

In this example, no column list follows the view name, MYVIEW. Therefore, the columns of the view have the same names as those of the DEPT table on which it is based. You can execute the following SELECT statement to see the view contents:

GUIP

```
SELECT * FROM MYVIEW;
```

GUIP

The result table looks like this:

DEPTNO	DEPTNAME	MGRNO
A00	CHAIRMANS OFFICE	000010
B01	PLANNING	000020
C01	INFORMATION CENTER	000030
D11	MANUFACTURING SYSTEMS	000060
E21	SOFTWARE SUPPORT	-----

Example 2: Defining a view that combines information from several tables You can create a view that contains a union of more than one table. DB2 provides two types of joins—an outer join and an inner join. An outer join includes rows in which the values in the join columns don't match, and rows in which the values match. An inner join includes only rows in which matching values in the join columns are returned.

The following example is an inner join of columns from the DEPT and EMP tables. The WHERE clause limits the view to just those columns in which the MGRNO in the DEPT table matches the EMPNO in the EMP table:

GUIP

```
CREATE VIEW MYVIEW AS
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT
  FROM DEPT, EMP
  WHERE EMP.EMPNO = DEPT.MGRNO;
```

GUIP

The result of executing this CREATE VIEW statement is an inner join view of two tables, which is shown below:

```
DEPTNO MGRNO  LASTNAME ADMRDEPT
=====
A00     000010 HAAS      A00
B01     000020 THOMPSON A00
C01     000030 KWAN      A00
D11     000060 STERN     D11
```

Guidelines for view names

The name for a view is an identifier of up to 128 characters.

The following example shows a view name:

```
Object Name
View      MYVIEW
```

Use the CREATE VIEW statement to define and name a view. Unless you specifically list different column names after the view name, the column names of the view are the same as those of the underlying table. When you create different column names for your view, remember the naming conventions that you established when designing the relational database.

How DB2 inserts and updates data through views

If you define a view, you can refer to the name of a view in an INSERT, UPDATE, or DELETE statement.

GUIP To ensure that the insert or update conforms to the view definition, specify the WITH CHECK OPTION clause. The following example illustrates some undesirable results of omitting that check.

Example 1: Suppose that you define a view, V1, as follows:

```
CREATE VIEW V1 AS
  SELECT * FROM EMP
  WHERE DEPT LIKE 'D%';
```

A user with the SELECT privilege on view V1 can see the information from the EMP table for employees in departments whose IDs begin with D. The EMP table has only one department (D11) with an ID that satisfies the condition.

Assume that a user has the INSERT privilege on view V1. A user with both SELECT and INSERT privileges can insert a row for department E01, perhaps erroneously, but cannot select the row that was just inserted.

The following example shows an alternative way to define view V1.

Example 2: You can avoid the situation in which a value that does not match the view definition is inserted into the base table. To do this, instead define view V1 to include the WITH CHECK OPTION clause:

```
CREATE VIEW V1 AS SELECT * FROM EMP
  WHERE DEPT LIKE 'D%' WITH CHECK OPTION;
```

With the new definition, any insert or update to view V1 must satisfy the predicate that is contained in the WHERE clause: DEPT LIKE 'D%'. The check can be valuable, but it also carries a processing cost; each potential insert or update must be checked against the view definition. Therefore, you must weigh the advantage of protecting data integrity against the disadvantage of performance degradation.

GUIP

Dropping DB2 views

You can drop a DB2 view by removing the object at the current server.

GUIP Issue the DROP VIEW SQL statement to remove the view.

GUIP

Implementing DB2 indexes

DB2 uses indexes not only to enforce uniqueness on column values, as for parent keys, but also to cluster data, to partition tables, to provide access paths to data for queries, and to order retrieved data without a sort.

Compressed indexes can hold more entries in individual leaf pages in the index tree. They can use 8 KB, 16 KB, or 32KB buffer pools.

Related concepts

"Padded and not padded indexes" (Introduction to DB2 for z/OS)

Creating DB2 indexes

When you define an index, DB2 builds and maintains an ordered set of pointers to rows of a base table or an auxiliary table.

Define the index after you have defined the table.

GUIP To create a partitioning index, or a secondary index, and an index space at the current server:

Issue the CREATE INDEX statement, and specify an index key.

The following example creates a unique index on the EMPPROJECT table. A composite key is defined on two columns, PROJNO and STDATE.

```
CREATE UNIQUE INDEX XPROJAC1
  ON EMPPROJECT
  (PROJNO ASC,
  STDATE ASC)
```

GUIP

Related reference

"CREATE INDEX" (DB2 SQL Reference)

Guidelines for defining indexes

By following certain guidelines, you can successfully work with indexes.

Index names

The name for an index is an identifier of up to 128 characters. You can qualify this name with an identifier, or schema, of up to 128 characters.

Example: The following example shows an index name:

```
Object Name  
Index MYINDEX
```

The index space name is an eight-character name, which must be unique among names of all index spaces and table spaces in the database.

Sequence of index entries

The sequence of the index entries can be in ascending order or descending order. The ASC and DESC keywords of the CREATE INDEX statement indicate ascending and descending order. ASC is the default.

Indexes on tables with large objects

You can use indexes on tables with LOBs the same way that you use them on other tables, but consider the following facts:

- A LOB column cannot be a column in an index.
- An auxiliary table can have only one index. (An auxiliary table, which you create by using the SQL CREATE AUXILIARY TABLE statement, holds the data for a column that a base table defines.)
- Indexes on auxiliary tables are different than indexes on base tables.

Creation of an index

If the table that you are indexing is empty, DB2 creates the index. However, DB2 does not actually create index entries until the table is loaded or rows are inserted. If the table is not empty, you can choose to have DB2 build the index when the CREATE INDEX statement is executed. Alternatively, you can defer the index build until later. Optimally, you should create the indexes on a table before loading the table. However, if your table already has data, choosing the DEFER option is preferred; you can build the index later by using the REBUILD INDEX utility.

Copies of an index

If your index is fairly large and needs the benefit of high availability, consider copying it for faster recovery. Specify the COPY YES clause on a CREATE INDEX or ALTER INDEX statement to allow the indexes to be copied. DB2 can then track the ranges of log records to apply during recovery, after the image copy of the index is restored. (The alternative to copying the index is to use the REBUILD INDEX utility, which might increase the amount of time that the index is unavailable to applications.)

Deferred allocation of index space data sets

When you execute a CREATE INDEX statement with the USING STOGROUP clause, DB2 generally defines the necessary VSAM data sets for the index space. In some cases, however, you might want to define an index without immediately allocating the data sets for the index space.

Example: You might be installing a software program that requires creation of many indexes, but your company might not need some of those indexes. You might prefer not to allocate data sets for indexes that you do not plan to use.

To defer the physical allocation of DB2-managed data sets, use the `DEFINE NO` clause of the `CREATE INDEX` statement. When you specify the `DEFINE NO` clause, DB2 defines the index but defers the allocation of data sets. The DB2 catalog table contains a record of the created index and an indication that the data sets are not yet allocated. DB2 allocates the data sets for the index space as needed when rows are inserted into the table on which the index is defined.

How DB2 implicitly creates an index

When the `PRIMARY KEY` or `UNIQUE` clause is specified in the `CREATE TABLE` statement and the `CREATE TABLE` statement is processed by the schema processor, or the table space that contains the table is implicitly created, DB2 implicitly creates the unique indexes that are used to enforce the uniqueness of the primary keys or unique keys.

When a `ROWID` column is defined as `GENERATED BY DEFAULT` in the `CREATE TABLE` statement, and the `CREATE TABLE` statement is processed by `SET CURRENT RULES = 'STD'`, or the table space that contains the table is implicitly created, DB2 implicitly creates the unique indexes used to enforce the uniqueness of the `ROWID` column. The privilege set must include the `USE` privilege of the buffer pool. Each index is created as if the following `CREATE INDEX` statement were issued:

```
CREATE UNIQUE INDEX xxx ON table-name (column1,...)
```

Where:

- *xxx* is the name of the index that DB2 generates.
- *table-name* is the name of the table that is specified in the `CREATE TABLE` statement.
- (*column1,...*) is the list of column names that were specified in the `UNIQUE` or `PRIMARY KEY` clause of the `CREATE TABLE` statement, or the column is a `ROWID` column that is defined as `GENERATED BY DEFAULT`.

In addition, if the table space that contains the table is implicitly created, DB2 will check the `DEFINE DATA SET` subsystem parameter to determine whether to define the underlying data set for the index space of the implicitly created index on the base table.

If `DEFINE DATA SET` is `NO`, the index is created as if the following `CREATE INDEX` statement is issued:

```
CREATE UNIQUE INDEX xxx ON table-name (column1,...) DEFINE NO
```

Recommendations for index page size

You can specify buffer pool sizes of 4-KB, 8-KB, 16-KB, and 32-KB for indexes on the `CREATE INDEX` statement.

In choosing an appropriate page size for an index, consider the following:

- A larger index page size can improve performance for indexes with sequential insert and fetch patterns but can cause degradation in performance for random accesses to an index. A larger page size can also yield a larger fanout in index non-leaf pages, which can reduce the number of levels in an index and improve performance.

- A smaller index page size can yield better performance for indexes with random fetch patterns.

Creating an index with a larger page size could reduce the number of page splits in the index. This is especially beneficial if the latch contention from the index splits is frequent. For example:

- Latch class 6 in data sharing
- Latch class X'46' in IFCID 57 performance trace record in a data sharing environment
- Latch class X'FE' in IFCID 57 record in a non-data-sharing environment

It can also lead to better performance for sequential access to the index.

GUIP The following example specifies a 16KB buffer pool for the index being created:

```
CREATE INDEX INDEX1 ON TABLE1 ( I1 ASC, I2 ASC) BUFFERPOOL BP16K1
```

You can specify a 4-KB, 8-KB, 16-KB, or 32-KB default buffer pool for indexes in a particular database using the CREATE DATABASE or ALTER DATABASE by using the INDEXBP option as in the following examples:

```
CREATE DATABASE MYDB INDEXBP BP16K1
ALTER DATABASE MYDB INDEXBP BP16K1
```

GUIP

Index versions

DB2 uses index versions to maximize data availability. Index versions enable DB2 to keep track of schema changes and simultaneously provide users with access to data in altered columns that are contained in one or more indexes.

When users retrieve rows from a table with an altered column, the data is displayed in the format that is described by the most recent schema definition, even though the data is not currently stored in this format. The most recent schema definition is associated with the current index version.

DB2 creates an index version each time you commit one of the following schema changes:

Table 14. Situations when DB2 creates an index version

When you commit this change to a schema	DB2 creates this type of corresponding index version
Use the ALTER TABLE statement to change the data type of a non-numeric column that is contained in one or more indexes.	A new index version for each index that is affected by this operation.
Use the ALTER TABLE statement to change the length of a VARCHAR column that is contained in one or more PADDED indexes.	A new index version for each index that is affected by this operation.
Use the ALTER TABLE statement to extend the length of a CHAR column in a table.	A new index version for each index that is affected by this operation.

Table 14. Situations when DB2 creates an index version (continued)

When you commit this change to a schema	DB2 creates this type of corresponding index version
Use the ALTER INDEX statement to add a column to an index.	One new index version; only one index is affected by this operation. The index is set to REBUILD-pending status if the column was not added to the table in the same commit operation.
Add a new column to both a table and an index in the same commit operation.	A new index version for each index that is affected by this operation.

Exceptions: DB2 does not create an index version under the following circumstances:

- When the index was created with DEFINE NO
- When you extend the length of a varying-length character (VARCHAR data type) or varying-length graphic (VARGRAPHIC data type) column that is contained in one or more indexes that are defined with the NOT PADDED option
- When you specify the same data type and length that a column (which is contained in one or more indexes) currently has, such that its definition does not actually change

DB2 creates only one index version if, in the same unit of work, you make multiple schema changes to columns that are contained in the same index. If you make these same schema changes in separate units of work, each change results in a new index version.

Related tasks

“Reorganizing indexes” on page 128

“Recycling index version numbers” on page 128

Compressing indexes

You can compress your indexes to significantly reduce the physical space requirements for most indexes.

Recommendation: Use index compression where a reduction in index storage consumption is more important than a possible decrease in index performance.

 To specify index compression:

Specify the compression option with a CREATE INDEX or ALTER INDEX statement.

- **YES:** Activates index compression. The buffer pool used to create the index must be either 8 KB or 16 KB in size. The physical page size on disk will be 4 KB. If you create the index with the clause COMPRESS YES, index compression begins as soon as the first index entries are added.

Restrictions:

- For user-managed index data sets, a compressed index requires a defined control interval size (CISZ) of 4 KB.

- For DB2-managed index data sets that are altered to enable compression (ALTER COMPRESS YES), the next utility operation to remove the REBUILD-pending state will not apply the utility REUSE option.
- **NO:** Specifies that no index compression will be in effect. This is the default option for the CREATE INDEX statement.

GUPI

If you activate or deactivate compression with an ALTER INDEX statement, the index will be placed into a REBUILD-pending (RBDP) state for partitioned indexes and a pageset REBUILD-pending (PSRBD) state for non-partitioned indexes. Then you need to use the REBUILD INDEX utility to rebuild the index, or use the REORG utility to reorganize the table space that corresponds to the index.

Implementing DB2 schemas

Use schemas to provide a logical classification of objects in the database.

Creating a schema by using the schema processor

Use the schema processor to create a schema.

Creating a schema by using the CREATE SCHEMA statement is also supported for compliance testing.

GUPI

CREATE SCHEMA statements cannot be embedded in a host program or executed interactively. To process the CREATE SCHEMA statement, you must use the schema processor. The ability to process schema definitions is provided for conformance to ISO/ANSI standards. The result of processing a schema definition is identical to the result of executing the SQL statements without a schema definition.

Outside of the schema processor, the order of statements is important. They must be arranged so that all referenced objects have been previously created. This restriction is relaxed when the statements are processed by the schema processor if the object table is created within the same CREATE SCHEMA. The requirement that all referenced objects have been previously created is not checked until all of the statements have been processed. For example, within the context of the schema processor, you can define a constraint that references a table that does not exist yet or GRANT an authorization on a table that does not exist yet.

To create a schema:

1. Write a CREATE SCHEMA statement.
2. Use the schema processor to execute the statement.

The following example shows schema processor input that includes the definition of a schema.

```
CREATE SCHEMA AUTHORIZATION SMITH
```

```
CREATE TABLE TESTSTUFF
  (TESTNO  CHAR(4),
   RESULT  CHAR(4),
   TESTTYPE CHAR(3))
```

```
CREATE TABLE STAFF
  (EMPNUM  CHAR(3) NOT NULL,
   EMPNAME CHAR(20),
   GRADE   DECIMAL(4),
```

```

CITY      CHAR(15))

CREATE VIEW STAFFV1
AS SELECT * FROM STAFF
WHERE GRADE >= 12

GRANT INSERT ON TESTSTUFF TO PUBLIC

GRANT ALL PRIVILEGES ON STAFF
TO PUBLIC

```



Processing schema definitions

You must use the schema processor to process CREATE SCHEMA statements.

Prerequisite: The schema processor sets the current SQLID to the value of the schema authorization ID before executing any of the statements in the schema definition. Therefore, that ID must have SYSADM or SYSCTRL authority, or it must be the primary or one of the secondary authorization IDs of the process that executes the schema processor. The same ID must have all the privileges that are needed to execute all the statements in the schema definition.

To process schema definitions, complete the following steps:

1. Run the schema processor (DSNHSP) as a batch job. Use the sample JCL provided in member DSNTEJ1S of the SDSNSAMP library.

The schema processor accepts only one schema definition in a single job. No statements that are outside the schema definition are accepted. Only SQL comments can precede the CREATE SCHEMA statement; the end of input ends the schema definition. SQL comments can also be used within and between SQL statements.

The processor takes the SQL from CREATE SCHEMA (the SYSIN data set), dynamically executes it, and prints the results in the SYSPRINT data set.

2. Optional: If a statement in the schema definition has an error, the schema processor processes the remaining statements but rolls back all the work at the end. In this case, you need to fix the statement in error and resubmit the entire schema definition.

Loading data into DB2 tables

You can use several methods to load data into DB2 tables.

The most common method for loading data into most of your tables is to use the LOAD utility. This utility loads data into DB2 persistent tables from sequential data sets by using BSAM. You can also use a cursor that is declared with an EXEC SQL utility control statement to load data from another SQL table with the DB2 UDB family cross-loader function. The LOAD utility cannot be used to load data into DB2 temporary tables or system-maintained materialized query tables.

When loading tables with indexes, referential constraints, or table check constraints, LOAD can perform several checks on the validity of data. If errors are found, the table space that is being loaded, its index spaces, and even other table spaces might be left in a restricted status. LOAD does not check the validity of informational referential constraints. Plan to make necessary corrections and remove restrictions after any LOAD job.

You can also use an SQL INSERT statement to copy all or selected rows of another table, in any of the following methods:

- Using the INSERT statement in an application program
- Interactively through SPUFI
- With the command line processor (for DB2 Version 9.1 or later databases)

To reformat data from IMS DL/I databases and VSAM and SAM loading for the LOAD utility, use DB2 DataPropagator™.

Loading data with the LOAD utility

Use the LOAD utility to load one or more tables of a table space. If you are loading a large number of rows, you will want to use the LOAD utility rather than inserting the rows by using the INSERT statement.

Before using the LOAD utility, make sure that you complete all of the prerequisite activities for your situation.

Run the LOAD utility control statement with the options that you need to load data.

Related reference

"Before running LOAD" (DB2 Utility Guide and Reference)

"LOAD" (DB2 Utility Guide and Reference)

How the LOAD utility loads DB2 tables

Use the LOAD utility to load one or more persistent tables of a table space, or one or more partitions of a table space. The LOAD utility operates on a table space, so you must have authority for all tables in the table space when you run LOAD.

The LOAD utility loads records into the tables and builds or extends any indexes defined on them. If the table space already contains data, you can choose whether you want to add the new data to the existing data or replace the existing data.

Additionally, you can use the LOAD utility to do the following:

- Compress data and build a compression dictionary
- Convert data between compatible data types and between encoding schemes
- Load multiple tables in a single table space

Delimited input and output files

The LOAD and UNLOAD utilities can accept or produce a delimited file, which is a sequential BSAM file with row delimiters and column delimiters. You can unload data from other systems into one or more files that use a delimited file format and then use these delimited files as input for the LOAD utility. You can also unload DB2 data into delimited files by using the UNLOAD utility and then use these files as input into another DB2 database.

INCURSOR option

The INCURSOR option of the LOAD utility specifies a cursor for the input data set. Use the EXEC SQL utility control statement to declare the cursor before running the LOAD utility. You define the cursor so that it selects data from another DB2 table. The column names in the SELECT statement must be identical to the column names of the table that is being loaded. The INCURSOR option uses the DB2 cross-loader function.

CCSID option

You can load input data into ASCII, EBCDIC, or Unicode tables. The ASCII, EBCDIC, and UNICODE options on the LOAD utility statement let you specify whether the format of the data in the input file is ASCII, EBCDIC, or Unicode. The CCSID option of the LOAD utility statement lets you specify the CCSIDs of the data in the input file. If the CCSID of the input data does not match the CCSID of the table space, the input fields are converted to the CCSID of the table space before they are loaded.

Availability during load

For partition-by-growth table spaces and nonpartitioned table spaces, data in the table space that is being loaded is unavailable to other application programs during the load operation with the exception of LOAD SHRLEVEL CHANGE. In addition, some SQL statements, such as CREATE, DROP, and ALTER, might experience contention when they run against another table space in the same DB2 database while the table is being loaded.

Default values for columns

When you load a table and do not supply a value for one or more of the columns, the action DB2 takes depends on the circumstances.

- If the column is not a ROWID or identity column, DB2 loads the default value of the column, which is specified by the DEFAULT clause of the CREATE or ALTER TABLE statement.
- If the column is a ROWID column that uses the GENERATED BY DEFAULT option, DB2 generates a unique value.
- If the column is an identity column that uses the GENERATED BY DEFAULT option, DB2 provides a specified value.
- With XML columns, if there is an implicitly created DOCID column in the table, it is created with the GENERATED ALWAYS attribute.

For ROWID or identity columns that use the GENERATED ALWAYS option, you cannot supply a value because this option means that DB2 always provides a value.

XML columns

You can load XML documents from input records if the total input record length is less than 32 KB. For input record length greater than 32 KB, you must load the data from a separate file. (You can also use a separate file if the input record length is less than 32 KB.)

When the XML data is to be loaded from the input record, specify XML as the input field type. The target column must be an XML column. The LOAD utility treats XML columns as varying-length data when loading XML directly from input records and expects a two-byte length field preceding the actual XML value.

The XML tables are loaded when the base table is loaded. You cannot specify the name of the auxiliary XML table to load.

XML documents must be well formed in order to be loaded.

LOB columns

The LOAD utility treats LOB columns as varying-length data. The length value for a LOB column must be 4 bytes. The LOAD utility can be used to load LOB data if the length of the row, including the length of the LOB data, does not exceed 32 KB. The auxiliary tables are loaded when the base table is loaded. You cannot specify the name of the auxiliary table to load.

Replacement or addition of data

You can use LOAD REPLACE to replace data in a single-table table space or in a multiple-table table space. You can replace all the data in a table space (using the REPLACE option), or you can load new records into a table space without destroying the rows that are already there (using the RESUME option).

Restricted status after LOAD

LOAD can place a table space or index space into a restricted status, of which there are several types.

Your use of a table space in restricted status is severely limited. In general, you cannot access its data through SQL; you can only drop the table space or one of its tables, or perform some operation that resets the status.

To discover what spaces are in restricted status, use the command:

```
-DISPLAY DATABASE (*) SPACENAM (*) RESTRICT
```

COPY-pending status

LOAD places a table space in the COPY-pending state if you load with LOG NO, which you might do to save space in the log. Immediately after that operation, DB2 cannot recover the table space. However, you can recover the table space by loading it again. Prepare for recovery, and remove the restriction, by making a full image copy using SHRLEVEL REFERENCE. (If you end the COPY job before it is finished, the table space is still in COPY-pending status.)

When you use REORG or LOAD REPLACE with the COPYDDN keyword, a full image copy data set (SHRLEVEL REF) is created during the execution of the REORG or LOAD utility. This full image copy is known as an *inline copy*. The table space is not left in COPY-pending state regardless of which LOG option is specified for the utility.

The inline copy is valid only if you replace the entire table space or partition. If you request an inline copy by specifying COPYDDN in a LOAD utility statement, an error message is issued, and the LOAD terminates if you specify LOAD RESUME YES or LOAD RESUME NO without REPLACE.

REBUILD-pending status

LOAD places all the index spaces for a table space in the REBUILD-pending status if you end the job (using -TERM UTILITY) before it completes the INDEXVAL phase. It places the table space itself in RECOVER-pending status if you end the job before it completes the RELOAD phase.

CHECK-pending status

LOAD places a table space in the CHECK-pending status if its referential or check integrity is in doubt. Because of this restriction, use of the CHECK DATA utility is recommended. That utility locates and, optionally, removes invalid data. If the CHECK DATA utility removes invalid data, the remaining data satisfies all referential and table check constraints, and the CHECK-pending restriction is lifted. LOAD does not set the CHECK-pending status for informational referential constraints.

Loading data by using the INSERT SQL statement

Another way to load data into tables is by using the INSERT SQL statement.

To load data by using the INSERT statement:

1. Issue an INSERT statement.
2. Insert single or multiple rows.

You can issue the statement interactively or embed it in an application program.

Related tasks

“Inserting a single row”

“Inserting multiple rows”

Inserting a single row

The simplest form of INSERT inserts a single row of data. In this form of the statement, you specify the table name, the columns into which the data is to be inserted, and the data itself.

GUPI To insert a single row, complete the following steps:

1. Issue an INSERT INTO statement.
2. Specify the table name, the columns into which the data is to be inserted, and the data itself.

For example, suppose that you create a test table, TEMPDEPT, with the same characteristics as the department table:

```
CREATE TABLE SMITH.TEMPDEPT
  (DEPTNO CHAR(3) NOT NULL,
  DEPTNAME VARCHAR(36) NOT NULL,
  MGRNO CHAR(6) NOT NULL,
  ADMRDEPT CHAR(3) NOT NULL)
IN DSN8D91A.DSN8S91D;
```

To now add a row to table TEMPDEPT, you can enter:

```
INSERT INTO SMITH.TEMPDEPT
  VALUES ('X05','EDUCATION','000631','A01');
```

If you write an application program to load data into tables, you use that form of INSERT, probably with host variables instead of the actual values shown in this

example. **GUPI**

Inserting multiple rows

You can use a form of INSERT that copies rows from another table.

GUPI To add multiple rows to a table, complete the following steps:

1. Issue an INSERT INTO statement. For example, the following statement loads TEMPDEPT with data from the department table about all departments that report to department D01.

```
INSERT INTO SMITH.TEMPDEPT
  SELECT DEPTNO,DEPTNAME,MGRNO,ADMRDEPT
  FROM DSN8910.DEPT
  WHERE ADMRDEPT='D01';
```

2. Optional: Embed the INSERT statement in an application program to insert multiple rows into a table from the values that are provided in host variable arrays.
 - a. Specify the table name, the columns into which the data is to be inserted, and the arrays that contain the data. Each array corresponds to a column.

For example, you can load TEMPDEPT with the number of rows in the host variable *num-rows* by using the following embedded INSERT statement:

```
EXEC SQL
  INSERT INTO SMITH.TEMPDEPT
  FOR :num-rows ROWS
  VALUES (:hva1, :hva2, :hva3, :hva4);
```

Assume that the host variable arrays *hva1*, *hva2*, *hva3*, and *hva4* are populated with the values that are to be inserted. The number of rows to insert must be

less than or equal to the dimension of each host variable array. 

Implications of using an INSERT statement to load tables

If you plan to use the INSERT statement to load tables, you should consider some of the implications.

- If you are inserting a large number of rows, you can use the LOAD utility. Alternatively, use multiple INSERT statements with predicates that isolate the data that is to be loaded, and then commit after each insert operation.
- When a table, whose indexes are already defined, is populated by using the INSERT statement, both the FREEPAGE and the PCTFREE parameters are ignored. FREEPAGE and PCTFREE are in effect only during a LOAD or REORG operation.
- Set the NOT LOGGED attribute for table spaces when large volumes of data are being inserted with parallel INSERT processes. If the data in the table space is lost or damaged, it can be reinserted from its original source.
- You can load a value for a ROWID column with an INSERT and fullselect only if the ROWID column is defined as GENERATED BY DEFAULT. If you have a table with a column that is defined as ROWID GENERATED ALWAYS, you can propagate non-ROWID columns from a table with the same definition.
- You cannot use an INSERT statement on system-maintained materialized query tables.
- REBUILD-pending (RBDP) status is set on a data-partitioned secondary index if you create the index after you insert a row into a table. In addition, the last partition of the table space is set to REORG-pending (REORP) restrictive status.
- When you insert a row into a table that resides in a partitioned table space and the value of the first column of the limit key is null, the result of the INSERT depends on whether DB2 enforces the limit key of the last partition:
 - When DB2 enforces the limit key of the last partition, the INSERT fails (if the first column is ascending).
 - When DB2 enforces the limit key of the last partition, the rows are inserted into the first partition (if the first column is descending).

- When DB2 does **not** enforce the limit key of the last partition, the rows are inserted into the last partition (if the first column is ascending) or the first partition (if the first column is descending).

DB2 enforces the limit key of the last partition for the following table spaces:

- Table spaces using table-controlled or index-controlled partitioning that are large (DSSIZE greater than, or equal to, 4 GB)
- Tables spaces using table-controlled partitioning that are large or non-large (any DSSIZE)

Loading data from DL/I

To convert data in IMS DL/I databases from a hierarchical structure to a relational structure so that it can be loaded into DB2 tables, you can use the DataRefresher™ licensed program.

Implementing DB2 stored procedures

Create and call stored procedures to perform a number of utility and application programming functions.

Creating stored procedures

Use the CREATE PROCEDURE statement to register a stored procedure with a database server.

GUIP To create a stored procedure, complete the following steps:

1. Issue the CREATE PROCEDURE statement.
2. Specify whether you want an external or SQL stored procedure.
 - **External:** defines an external stored procedure at the current server.
 - **SQL:** defines an SQL procedure at the current server and specifies the source statements for the procedure.

GUIP

Deleting stored procedures

Use the DROP statement to delete a stored procedure at the current server.

GUIP To delete a stored procedure, complete the following steps:

1. Issue the DROP PROCEDURE statement.
2. Specify the name of the stored procedure.

For example, drop the stored procedure MYPROC in schema SYSPROC.

```
DROP PROCEDURE SYSPROC.MYPROC;
```

GUIP

Implementing DB2 user-defined functions

In contrast to a built-in DB2 functions, you can create and implement your own external, sourced, or SQL functions.

Creating user-defined functions

The CREATE FUNCTION statement registers a user-defined function with a database server.

GUIP To create a user-defined function, complete the following step:

1. Issue the CREATE FUNCTION statement.
2. Specify the type of function you want to create.
 - External scalar
 - External table
 - Sourced
 - SQL scalar

GUIP

Deleting user-defined functions

Use the DROP statement to delete a user-defined function at the current server.

GUIP To delete a user-defined function, complete the following steps:

1. Issue the DROP statement.
2. Specify FUNCTION or SPECIFIC FUNCTION.

For example, drop the user-defined function ATOMIC_WEIGHT from the schema CHEM.

```
DROP FUNCTION CHEM.ATOMIC_WEIGHT;
```

GUIP

Estimating disk storage for user data

In order to properly estimate the amount of disk storage necessary to store your data, you need to consider several factors and figures.

Estimating the space requirements for DB2 objects is easier if you collect and maintain a statistical history of those objects. The accuracy of your estimates depends on the currentness of the statistical data.

To estimate disk storage for user data:

1. First, ensure that the statistics history is current by using the MODIFY STATISTICS utility to delete outdated statistical data from the catalog history tables.
2. Then, use the DB2 Estimator to calculate space estimates for tables, indexes, and other factors.

General approach to estimating storage

Estimating the space requirements for DB2 objects is easier if you collect and maintain a statistical history of those objects.

The accuracy of your estimates depends on the currentness of the statistical data. To ensure that the statistics history is current, use the MODIFY STATISTICS utility to delete outdated statistical data from the catalog history tables.

The amount of disk space you need for your data is not just the number of bytes of data; the true number is some multiple of that. That is,

$$\text{space required} = M \times (\text{number of bytes of data})$$

The multiplier M depends on your circumstances. It includes factors that are common to all data sets on disk, as well as others that are peculiar to DB2. It can vary significantly, from a low of about 1.25, to 4.0 or more. For a first approximation, set $M=2$. For more information, see “Calculating the space required for a table” on page 75.

For more accuracy, you can calculate M as the product of the following factors:

Record overhead

Allows for eight bytes of record header and control data, plus space wasted for records that do not fit exactly into a DB2 page. For the second consideration, see “Recommendations for LOB page size” on page 42. The factor can range from about 1.01 (for a careful space-saving design) to as great as 4.0. A typical value is about 1.10.

Free space

Allows for space intentionally left empty to allow for inserts and updates. You can specify this factor on the CREATE TABLESPACE statement. The factor can range from 1.0 (for no free space) to 200 (99% of each page used left free, and a free page following each used page). With default values, the factor is about 1.05.

Unusable space

Track lengths in excess of the nearest multiple of page lengths. The following table shows the track size, number of pages per track, and the value of the unusable-space factor for several different device types.

Table 15. Unusable space factor by device type

Device type	Track size	Pages per track	Factor value
3380	47476	10	1.16
3390	56664	12	1.15

Data set excess

Allows for unused space within allocated data sets, occurring as unused tracks or part of a track at the end of any data set. The amount of unused space depends upon the volatility of the data, the amount of space management done, and the size of the data set. Generally, large data sets can be managed more closely, and those that do not change in size are easier to manage. The factor can range without limit above 1.02. A typical value is 1.10.

Indexes

Allows for storage for indexes to data. For data with no indexes, the factor is 1.0. For a single index on a short column, the factor is 1.01. If every column is indexed, the factor can be greater than 2.0. A typical value is 1.20. For further discussion of the factor, see “Calculating the space required for an index” on page 78.

The following table shows calculations of the multiplier M for three different database designs:

- The *tight* design is carefully chosen to save space and allows only one index on a single, short field.

- The *loose* design allows a large value for every factor, but still well short of the maximum. Free space adds 30% to the estimate, and indexes add 40%.
- The *medium* design has values between the other two. You might want to use these values in an early stage of database design.

In each design, the device type is assumed to be a 3390. Therefore, the unusable-space factor is 1.15. M is always the product of the five factors.

Table 16. Calculations for different database designs

Factor	Tight design	Medium design	Loose design
Record overhead ×	1.02	1.10	1.30
Free space ×	1.00	1.05	1.30
Unusable space ×	1.15	1.15	1.15
Data set excess ×	1.02	1.10	1.30
Indexes =	1.02	1.20	1.40
Multiplier M	1.22	1.75	3.54

In addition to the space for your data, external storage devices are required for:

- Image copies of data sets, which can be on tape
- System libraries, system databases, and the system log
- Temporary work files for utility and sort jobs

A rough estimate of the additional external storage needed is three times the amount calculated for disk storage.

Calculating the space required for a table

The following topics provide information about how to calculate the space required for a table.

- “Calculations for record lengths and pages”
- “Saving space with data compression” on page 83
- “Estimating storage for LOBs” on page 76
- “Estimating storage when using the LOAD utility” on page 77

Space allocation parameters are specified in kilobytes (KB).

Calculations for record lengths and pages

An important consideration in the design of a table is the record size.

In DB2, a record is the storage representation of a row. Records are stored within pages that are 4 KB, 8 KB, 16 KB, or 32 KB. Generally, you cannot create a table in which the maximum record size is greater than the page size.

Also consider:

- Normalizing your entities
- Using larger page sizes
- Using LOB data types if a single column in a table is greater than 32 K

In addition to the bytes of actual data in the row (not including LOB data, which is not stored in the base row or included in the total length of the row), each record has:

- A six-byte prefix
- One additional byte for each column that can contain null values

- Two additional bytes for each varying-length column or ROWID column
- Six bytes of descriptive information in the base table for each LOB column

GUIP The sum of each column's length is the *record length*, which is the length of data that is physically stored in the table. You can retrieve the value of the AVGROWLEN column in the SYSIBM.SYSTABLES catalog table to determine the average length of rows within a table. The *logical record length* can be longer, for example, if the table contains LOBs. **GUIP**

Every data page has:

- A 22-byte header
- A 2-byte directory entry for each record stored in the page

To simplify the calculation of record and page length, consider the directory entry as part of the record. Then, every record has a fixed overhead of 8 bytes, and the space available to store records in a 4 KB page is 4074 bytes. Achieving that maximum in practice is not always simple. For example, if you are using the default values, the LOAD utility leaves approximately 5 percent of a page as free space when loading more than one record per page. Therefore, if two records are to fit in a page, each record cannot be longer than 1934 bytes (approximately $0.95 \times 4074 \times 0.5$).

Furthermore, the page size of the table space in which the table is defined limits the record length. If the table space is 4 KB, the record length of each record cannot be greater than 4056 bytes. Because of the 8-byte overhead for each record, the sum of column lengths cannot be greater than 4048 bytes (4056 minus the 8-byte overhead for a record).

DB2 provides three larger page sizes to allow for longer records. You can improve performance by using pages for record lengths that best suit your needs.

As shown in the following table, the maximum record size for each page size depends on the size of the table space and on whether you specified the EDITPROC clause.

Table 17. Maximum record size (in bytes)

EDITPROC	4-KB page	8-KB page	16-KB page	32-KB page
NO	4056	8138	16330	32714
YES	4046	8128	16320	32704

GUIP Creating a table using CREATE TABLE LIKE in a table space of a larger page size changes the specification of LONG VARCHAR to VARCHAR and LONG VARGRAPHIC to VARGRAPHIC. You can also use CREATE TABLE LIKE to create a table with a smaller page size in a table space if the maximum record size is within the allowable record size of the new table space. **GUIP**

Estimating storage for LOBs

Before calculating the storage required for LOB table spaces, you must understand the size restrictions for large object data types (LOBs).

Tables with LOBs can store byte strings up to 2 GB. A base table can be defined with one or more LOB columns. The LOB columns are logically part of the base table but are physically stored in an auxiliary table. In place of each LOB column,

there is an *indicator column*, which is a column with descriptive information about the LOB. When a base table has LOB columns, then each row of the table has a *row identifier*, which is a varying-length 17-byte field. You must consider the overhead of the indicator column and row identifiers when estimating table size. If the LOB column is NULL or has a value of zero, no space is allocated in the auxiliary table.

To estimate the storage required for LOB table spaces, complete the following steps:

1. Begin with your estimates from other table spaces
2. Round the figure up to the next page size
3. Multiply the figure by 1.1

One page never contains more than one LOB. When a LOB value is deleted, the space occupied by that value remains allocated as long as any application might access that value.

An auxiliary table resides in a LOB table space. There can be only one auxiliary table in a LOB table space. An auxiliary table can store only one LOB column of a base table and there must be one and only one index on this column.

Estimating storage when using the LOAD utility

You must complete several calculations to estimate the storage required for a table to be loaded by the LOAD utility.

For a table to be loaded by the LOAD utility, assume the following values:

- Let FLOOR be the operation of discarding the decimal portion of a real number.
- Let CEILING be the operation of rounding a real number up to the next highest integer.
- Let *number of records* be the total number of records to be loaded.
- Let *average record size* be the sum of the lengths of the fields in each record, using an average value for varying-length fields, and including the following amounts for overhead:
 - 8 bytes for the total record
 - 1 byte for each field that allows nulls
 - 2 bytes for each varying-length field
- Let *percsave* be the percentage of kilobytes saved by compression (as reported by the DSN1COMP utility in message DSN1940I)
- Let *compression ratio* be $\text{percsave}/100$

To calculate the storage required when using the LOAD utility, complete the following steps:

1. Calculate the usable page size.

Usable page size is the page size minus a number of bytes of overhead (that is, 4 KB - 40 for 4 KB pages, 8 KB - 54 for 8 KB pages, 16 KB - 54 for 16 KB pages, or 32 KB - 54 for 32 KB pages) multiplied by $(100-p) / 100$, where p is the value of PCTFREE.

If your average record size is less than 16, then usable page size is 255 (maximum records per page) multiplied by average record size multiplied by $(100-p) / 100$.

2. Calculate the records per page.

Records per page is $\text{MIN}(\text{MAXROWS}, \text{FLOOR}(\text{usable page size} / \text{average record size}))$, but cannot exceed 255 and cannot exceed the value you specify for MAXROWS.

3. Calculate the pages used.

Pages used is $2 + \text{CEILING}(\text{number of records} / \text{records per page})$.

4. Calculate the total pages used.

Total pages is $\text{FLOOR}(\text{pages used} \times (1 + fp) / fp)$, where *fp* is the (nonzero) value of `FREEPAGE`. If `FREEPAGE` is 0, then *total pages* is equal to *pages used*.

If you are using data compression, you need additional pages to store the dictionary.

5. Estimate the number of kilobytes required for a table.

- **If you do not use data compression**, the estimated number of kilobytes is *total pages* × page size (4 KB, 8 KB, 16 KB, or 32 KB).
- **If you use data compression**, the estimated number of kilobytes is *total pages* × page size (4 KB, 8 KB, 16 KB, or 32 KB) × (1 - *compression ratio*).

For example, consider a table space containing a single table with the following characteristics:

- *Number of records* = 100000
- *Average record size* = 80 bytes
- *Page size* = 4 KB
- `PCTFREE` = 5 (5% of space is left free on each page)
- `FREEPAGE` = 20 (one page is left free for each 20 pages used)
- `MAXROWS` = 255

If the data is not compressed, you get the following results:

- *Usable page size* = $4056 \times 0.95 = 3853$ bytes
- *Records per page* = $\text{MIN}(\text{MAXROWS}, \text{FLOOR}(3853 / 80)) = 48$
- *Pages used* = $2 + \text{CEILING}(100000 / 48) = 2085$
- *Total pages* = $\text{FLOOR}(2085 \times 21 / 20) = 2189$
- *Estimated number of kilobytes* = $2189 \times 4 = 8756$

If the data is compressed, multiply the estimated number of kilobytes for an uncompressed table by (1 - *compression ratio*) for the estimated number of kilobytes required for the compressed table.

Related reference

"CREATE TABLE" (DB2 SQL Reference)

Calculating the space required for an index

The following topics provide information about how to calculate the space required for an index.

- "Levels of index pages"
- "Estimating storage from the number of index pages" on page 79

Space allocation parameters are specified in kilobytes (KB).

Levels of index pages

The storage required for an index, newly built by the `LOAD` utility, depends on the number of index pages at all levels. That, in turn, depends on whether the index is unique or not. Indexes can have more than one level of pages.

Index pages that point directly to the data in tables are called *leaf pages* and are said to be on *level 0*. In addition to data pointers, leaf pages contain the key and record-ID (RID).

If an index has more than one leaf page, it must have at least one nonleaf page that contains entries that point to the leaf pages. If the index has more than one nonleaf page, then the nonleaf pages whose entries point to leaf pages are said to be on *level 1*. If an index has a second level of nonleaf pages whose entries point to nonleaf pages on level 1, then those nonleaf pages are said to be on *level 2*, and so on. The highest level of an index contains a single page, which DB2 creates when it first builds the index. This page is called the *root page*. The root page is a 4-KB index page. The following figure shows, in schematic form, a typical index.

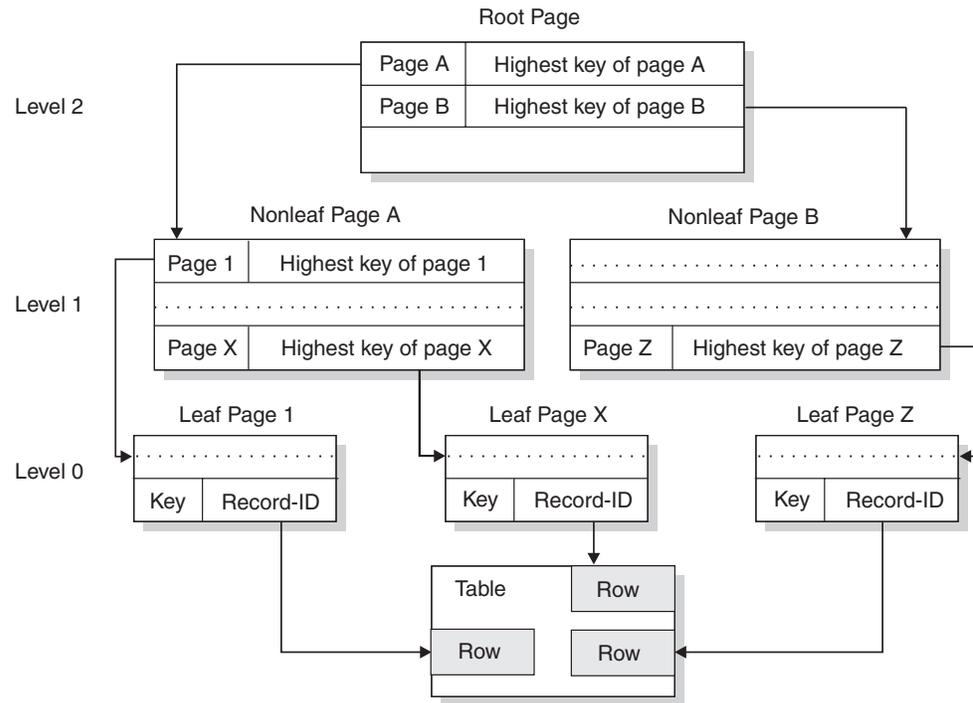


Figure 13. Sample index structure and pointers (three-level index)

If you insert data with a constantly increasing key, DB2 adds the new highest key to the top of a new page. Be aware, however, that DB2 treats nulls as the highest value. When the existing high key contains a null value in the first column that differentiates it from the new key that is inserted, the inserted non-null index entries cannot take advantage of the *highest-value* split.

Estimating storage from the number of index pages

For an index to be loaded by the LOAD utility, you should estimate the future storage requirements of the index.

An index key on an auxiliary table used for LOBs is 19 bytes and uses the same formula as other indexes. The RID value stored within the index is 5 bytes, the same as for large table spaces (defined with DSSIZE greater than or equal to 4 GB).

In general, the length of the index key is the sum of the lengths of all the columns of the key, plus the number of columns that allow nulls. The length of a varying-length column is the maximum length if the index is padded. Otherwise, if an index is not padded, estimate the length of a varying-length column to be the average length of the column data, and add a two-byte length field to the estimate. You can retrieve the value of the AVGKEYLEN column in the SYSIBM.SYSINDEXES catalog table to determine the average length of keys within an index.

The following index calculations are intended only to help you estimate the storage required for an index. Because there is no way to predict the exact number of duplicate keys that can occur in an index, the results of these calculations are not absolute. It is possible, for example, that for a nonunique index, more index entries than the calculations indicate might be able to fit on an index page.

Important: Space allocation parameters are specified in kilobytes.

In the following calculations, assume the following:

- k length of the index key
- n average number of data records per distinct key value of a nonunique index. For example:
 - a = number of data records per index
 - b = number of distinct key values per index
 - $n = a / b$
- f value of PCTFREE
- p value of FREEPAGE
- r record identifier (RID) length. Let $r = 4$ for indexes on nonlarge table spaces and $r = 5$ for indexes on large spaces (defined with DSSIZE greater than or equal to 4 GB) and on auxiliary tables.

FLOOR

the operation of discarding the decimal portion of a real number

CEILING

the operation of rounding a real number up to the next highest integer

MAX the operation of selecting the highest integer value

To estimate index storage size, complete the following calculations:

1. Calculate the pages for a unique index.
 - a. Calculate the *total leaf pages*
 - 1) Calculate the *space per key*
space per key is approximately $k + r + 3$
 - 2) Calculate the *usable space per page*
usable space per page is approximately $\text{FLOOR}((100 - f) \times 4038 / 100)$
 - 3) Calculate the *entries per page*
entries per page is approximately $\text{FLOOR}(\text{usable space per page} / \text{space per key})$
 - 4) Calculate the *total leaf pages*
total leaf pages is approximately $\text{CEILING}(\text{number of table rows} / \text{entries per page})$
 - b. Calculate the *total nonleaf pages*
 - 1) Calculate the *space per key*
space per key is approximately $k + 7$
 - 2) Calculate the *usable space per page*
usable space per page is approximately $\text{FLOOR}(\text{MAX}(90, (100 - f)) \times 4046 / 100)$
 - 3) Calculate the *entries per page*
entries per page is approximately $\text{FLOOR}(\text{usable space per page} / \text{space per key})$
 - 4) Calculate the *minimum child pages*

- minimum child pages* is approximately $\text{MAX}(2, (\text{entries per page} + 1))$
- 5) Calculate the *level 2 pages*
level 2 pages is approximately $\text{CEILING}(\text{total leaf pages} / \text{minimum child pages})$
 - 6) Calculate the *level 3 pages*
level 3 pages is approximately $\text{CEILING}(\text{level 2 pages} / \text{minimum child pages})$
 - 7) Calculate the *level x pages*
level x pages is approximately $\text{CEILING}(\text{previous level pages} / \text{minimum child pages})$
 - 8) Calculate the ***total nonleaf pages***
total nonleaf pages is approximately $(\text{level 2 pages} + \text{level 3 pages} + \dots + \text{level } x \text{ pages until the number of level } x \text{ pages} = 1)$
2. Calculate pages for a nonunique index:
- a. Calculate the *total leaf pages*
 - 1) Calculate the *space per key*
space per key is approximately $4 + k + (n \times (r+1))$
 - 2) Calculate the *usable space per page*
usable space per page is approximately $\text{FLOOR}((100 - f) \times 4038 / 100)$
 - 3) Calculate the *key entries per page*
key entries per page is approximately $n \times (\text{usable space per page} / \text{space per key})$
 - 4) Calculate the *remaining space per page*
remaining space per page is approximately $\text{usable space per page} - (\text{key entries per page} / n) \times \text{space per key}$
 - 5) Calculate the *data records per partial entry*
data records per partial entry is approximately $\text{FLOOR}((\text{remaining space per page} - (k + 4)) / 5)$
 - 6) Calculate the *partial entries per page*
partial entries per page is approximately $(n / \text{CEILING}(n / \text{data records per partial entry}))$ if *data records per partial entry* ≥ 1 , or 0 if *data records per partial entry* < 1
 - 7) Calculate the *entries per page*
entries per page is approximately $\text{MAX}(1, (\text{key entries per page} + \text{partial entries per page}))$
 - 8) Calculate the ***total leaf pages***
total leaf pages is approximately $\text{CEILING}(\text{number of table rows} / \text{entries per page})$
 - b. Calculate the *total nonleaf pages*
 - 1) Calculate the *space per key*
space per key is approximately $k + r + 7$
 - 2) Calculate the *usable space per page*
usable space per page is approximately $\text{FLOOR}(\text{MAX}(90, (100 - f)) \times (4046 / 100))$
 - 3) Calculate the *entries per page*
entries per page is approximately $\text{FLOOR}(\text{usable space per page} / \text{space per key})$
 - 4) Calculate the *minimum child pages*
minimum child pages is approximately $\text{MAX}(2, (\text{entries per page} + 1))$
 - 5) Calculate the *level 2 pages*

- level 2 pages* is approximately $\text{CEILING}(\text{total leaf pages} / \text{minimum child pages})$
- 6) Calculate the *level 3 pages*
level 3 pages is approximately $\text{CEILING}(\text{level 2 pages} / \text{minimum child pages})$
- 7) Calculate the *level x pages*
level x pages is approximately $\text{CEILING}(\text{previous level pages} / \text{minimum child pages})$
- 8) Calculate the ***total nonleaf pages***
total nonleaf pages is approximately $(\text{level 2 pages} + \text{level 3 pages} + \dots + \text{level } x \text{ pages until } x = 1)$
3. Calculate the total space requirement by estimating the number of kilobytes required for an index built by LOAD
- Calculate the *free pages*
free pages is approximately $\text{FLOOR}(\text{total leaf pages} / p)$, or 0 if $p = 0$
 - Calculate the *space map pages*
space map pages is approximately $\text{CEILING}((\text{tree pages} + \text{free pages}) / 8131)$
 - Calculate the *tree pages*
tree pages is approximately $\text{MAX}(2, (\text{total leaf pages} + \text{total nonleaf pages}))$
 - Calculate the *total index pages*
total index pages is approximately $\text{MAX}(4, (1 + \text{tree pages} + \text{free pages} + \text{space map pages}))$
 - Calculate the ***total space requirement***
total space requirement is approximately $4 \times (\text{total index pages} + 2)$

In the following example of the entire calculation, assume that an index is defined with these characteristics:

- It is unique.
- The table it indexes has 100000 rows.
- The key is a single column defined as CHAR(10) NOT NULL.
- The value of PCTFREE is 5.
- The value of FREEPAGE is 4.

Table 18. Sample of the total space requirement for an index

Quantity	Calculation	Result
Length of key	k	10
Average number of duplicate keys	n	1
PCTFREE	f	5
FREEPAGE	p	4
Calculate total leaf pages		
Space per key	$k + 7$	17
Usable space per page	$\text{FLOOR}((100 - f) \times 4038/100)$	3844
Entries per page	$\text{FLOOR}(\text{usable space per page} / \text{space per key})$	225
Total leaf pages	$\text{CEILING}(\text{number of table rows} / \text{entries per page})$	445

Table 18. Sample of the total space requirement for an index (continued)

Quantity	Calculation	Result
Calculate total nonleaf pages		
Space per key	$k + 7$	17
Usable space per page	$\text{FLOOR}(\text{MAX}(90, (100 - f)) \times (4046/100))$	3836
Entries per page	$\text{FLOOR}(\text{usable space per page} / \text{space per key})$	226
Minimum child pages	$\text{MAX}(2, (\text{entries per page} + 1))$	227
Level 2 pages	$\text{CEILING}(\text{total leaf pages} / \text{minimum child pages})$	2
Level 3 pages	$\text{CEILING}(\text{level 2 pages} / \text{minimum child pages})$	1
Total nonleaf pages	$(\text{level 2 pages} + \text{level 3 pages} + \dots + \text{level } x \text{ pages until } x = 1)$	3
Calculate total space required		
Free pages	$\text{FLOOR}(\text{total leaf pages} / p)$, or 0 if $p = 0$	111
Tree pages	$\text{MAX}(2, (\text{total leaf pages} + \text{total nonleaf pages}))$	448
Space map pages	$\text{CEILING}((\text{tree pages} + \text{free pages})/8131)$	1
Total index pages	$\text{MAX}(4, (1 + \text{tree pages} + \text{free pages} + \text{space map pages}))$	561
TOTAL SPACE REQUIRED, in KB	$4 \times (\text{total index pages} + 2)$	2252

Saving space with data compression

You can reduce the space required for a table by using data compression.

The disk space that is saved by data compression is countered by the disk space that is required for a dictionary. Every compressed table space or partition requires a dictionary.

Compressing data

Use data compression to reduce the space required for a table if your system meets the requirements.

Restriction: You cannot compress data in a LOB table space.

To find out how much space you can save by compressing your data:

Run the DSN1COMP utility on your data sets.

Message DSN1940I of DSN1COMP reports an estimate of the percentage of kilobytes that would be saved by using data compression.

Related reference

"DSN1COMP" (DB2 Utility Guide and Reference)

Calculating the space that is required for a dictionary

A *dictionary* contains the information that is used for compressing and decompressing the data in a table space or partition, and resides in that table space or partition.

You can skip this task if you are not going to compress data. Space allocation parameters are specified in pages (either 4 KB, 8 KB, 16 KB, or 32 KB).

To calculate the disk space required by a dictionary, and the virtual storage required in the xxxxDBM1 address space when a dictionary is read into storage from a buffer pool, you must complete the following steps.

Calculating disk requirements for a dictionary

The following task explains how to calculate the disk requirements for a dictionary associated with a compressed nonsegmented table space and for a dictionary associated with a compressed segmented table space.

Determine your table space type:

- **Nonsegmented table space**

The dictionary contains 4096 entries in most cases.

This means you need to allocate an additional sixteen 4-KB pages, eight 8-KB pages, four 16-KB pages, or two 32-KB pages. Although it is possible that your dictionary can contain fewer entries, allocate enough space to accommodate a dictionary with 4096 entries.

For 32-KB pages, one segment (minimum of four pages) is sufficient to contain the dictionary. Use the following table to determine how many 4-KB pages, 8-KB pages, 16-KB pages, or 32-KB pages to allocate for the dictionary of a compressed nonsegmented table space.

Table 19. Pages required for the dictionary of a compressed nonsegmented table space

Table space page size (KB)	Dictionary size (512 entries)	Dictionary size (1024 entries)	Dictionary size (2048 entries)	Dictionary size (4096 entries)	Dictionary size (8192 entries)
4	2	4	8	16	32
8	1	2	4	8	16
16	1	1	2	4	8
32	1	1	1	2	4

- **Segmented table space**

The size of the dictionary depends on the size of your segments. Assuming 4096 entries is recommended.

Use the following table to determine how many 4-KB pages to allocate for the dictionary of a compressed segmented table space.

Table 20. Pages required for the dictionary of a compressed segmented table space

Segment size (4-KB pages)	Dictionary size (512 entries)	Dictionary size (1024 entries)	Dictionary size (2048 entries)	Dictionary size (4096 entries)	Dictionary size (8192 entries)
4	4	4	8	16	32
8	8	8	8	16	32
12	12	12	12	24	36
≥16	Segment size	Segment size	Segment size	Segment size	Segment size

Now, determine the virtual storage size required for a dictionary.

Calculating virtual storage requirements for a dictionary

Estimate the virtual storage as part of your calculations for the space required for a dictionary.

To calculate how much storage is needed in the xxxxDBM1 address space for each dictionary, complete the following step:

Calculate the necessary dictionary size by using the following formula:

dictionary size (number of entries) × 16 bytes

When a dictionary is read into storage from a buffer pool, the *whole* dictionary is read, and it remains there as long as the compressed table space is being accessed.

Chapter 3. Altering your database design

After using a relational database for a while, you might want to change some aspects of its design.

To alter the database design you need to change the definitions of DB2 objects.

Recommendation: If possible, use the SQL ALTER statement to change the definitions of DB2 objects. When you cannot make changes with the ALTER statement, you typically must:

1. Use the DROP statement to remove the object.
2. Use the COMMIT statement to commit the changes to the object.
3. Use the CREATE statement to re-create the object.

Attention: The DROP statement has a cascading effect. Objects that are dependent on the dropped object are also dropped. For example, all authorities for those objects disappear, and plans or packages that reference deleted objects are marked invalid by DB2.

Altering DB2 databases

You can alter a DB2 database by changing the description of a database at the current server.

GUI Issue the ALTER DATABASE SQL statement to change clauses that are used to create a database. **GUI**

Related concepts

"DB2 databases" (Introduction to DB2 for z/OS)

Related reference

"ALTER DATABASE options"

"ALTER DATABASE" (DB2 SQL Reference)

ALTER DATABASE options

Issue the ALTER DATABASE SQL statement to change clauses that are used to create a database.

The ALTER DATABASE statement allows you to change the following clauses:

STOGROUP

Use this option to change the name of the default storage group to support disk space requirements for table spaces and indexes within the database. The new default DB2 storage group is used only for new table spaces and indexes; existing definitions do not change.

BUFFERPOOL

Use this option to change the name of the default buffer pool for table spaces and indexes within the database. Again, it applies only to new table spaces and indexes; existing definitions do not change.

INDEXBP

Use this option to change the name of the default buffer pool for the

indexes within the database. The new default buffer pool is used only for new indexes; existing definitions do not change.

Related reference

"ALTER DATABASE" (DB2 SQL Reference)

Altering DB2 storage groups

To change the description of a storage group at the current server use the ALTER STOGROUP statement.

GUIP To alter a storage group, complete the following steps:

1. Issue an ALTER STOGROUP statement.
2. Specify whether you want SMS to manage your DB2 storage groups, or to add or remove volumes from a storage group. **GUIP**

If you want to migrate to another device type or change the catalog name of the integrated catalog facility, you need to move the data.

Related concepts

"Moving DB2 data" on page 140

Letting SMS manage your DB2 storage groups

Using the SMS product Data Facility Storage Management Subsystem (DFSMS) to manage your data sets can result in a reduced workload for DB2 database and storage administrators.

GUIP To let SMS manage the storage needed for the objects that the storage group supports, complete the following steps:

1. Issue an ALTER STOGROUP statement. You can specify SMS classes when you alter a storage group.
2. Specify ADD VOLUMES ('*') and REMOVE VOLUMES (*current-vols*) where *current-vols* is the list of the volumes that are currently assigned to the storage group. For example,

```
ALTER STOGROUP DSN8G910
  REMOVE VOLUMES (VOL1)
  ADD VOLUMES ('*');
```

The following example shows how to alter a storage group to SMS-managed using the DATACLAS, MGMTCLAS, or STORCLAS keywords.

```
ALTER STOGROUP SGO5001
  MGMTCLAS REGSMMC2
  DATACLAS REGSMDC2
  STORCLAS REGSMSC2;
```

SMS manages every new data set that is created after the ALTER STOGROUP statement is executed; SMS does not manage data sets that are created before the execution of the statement. **GUIP**

Adding or removing volumes from a DB2 storage group

When you add or remove volumes from a storage group, all the volumes in a storage group must be of the same type; and, when a storage group is used to

extend a data set, the volumes must have the same device type as the volumes that were used when the data set was defined.

The changes you make to the volume list by ALTER STOGROUP have no effect on existing storage. Changes take effect when new objects are defined or when the REORG, RECOVER, or LOAD REPLACE utilities are used on those objects. For example, if you use ALTER STOGROUP to remove volume 22222 from storage group DSN8G910, the DB2 data on that volume remains intact. However, when a new table space is defined using DSN8G910, volume 22222 is not available for space allocation.

GUIP To force a volume off and add a new volume, complete the following steps:

1. Use the SYSIBM.SYSTABLEPART catalog table to determine which table spaces are associated with the storage group. For example, the following query indicates which table spaces use storage group DSN8G910:

```
SELECT TSNAME, DBNAME
FROM SYSIBM.SYSTABLEPART
WHERE STORNAME = 'DSN8G910' AND STORTYPE = 'I';
```

2. Make an image copy of each table space; for example, COPY TABLESPACE *dbname.tsname* DEVT SYSDA.
3. Ensure that the table space is not being updated in such a way that the data set might need to be extended. For example, you can stop the table space with the DB2 command STOP DATABASE (*dbname*) SPACENAM (*tsname*).
4. Use the ALTER STOGROUP statement to remove the volume that is associated with the old storage group and to add the new volume:

```
ALTER STOGROUP DSN8G910
REMOVE VOLUMES (VOL1)
ADD VOLUMES (VOL2);
```

Restriction: When a new volume is added, or when a storage group is used to extend a data set, the volumes must have the same device type as the volumes that were used when the data set was defined.

5. Start the table space with utility-only processing by using the DB2 command START DATABASE (*dbname*) SPACENAM (*tsname*) ACCESS(UT).
6. Use the RECOVER or REORG utility to move the data in each table space; for example, RECOVER *dbname.tsname*.
7. Start the table space with START DATABASE (*dbname*) SPACENAM (*tsname*).

GUIP

Altering table spaces

Use the ALTER TABLESPACE statement to change the description of a table space at the current server.

GUIP The ALTER TABLESPACE statement can be embedded in an application program or issued interactively.

You can use the ALTER TABLESPACE statement to change the maximum partition size for partition-by-growth table spaces.

The ALTER TABLESPACE statement cannot be used to change certain attributes of your table space. For example, you must use other methods to:

- Change the SEGSIZE attribute
- Change the number of partitions for regular-partitioned or range-partitioned universal table space
- Convert to a large table space 

Related reference

"ALTER TABLESPACE" (DB2 SQL Reference)

Changing the logging attribute

You can use the ALTER TABLESPACE statement to set the logging attribute of a table space.

Important: Limit the use of the NOT LOGGED attribute. Logging is not generally a performance bottleneck, given that in an average environment logging accounts for less than 5% of the central processing unit (CPU) utilization. Therefore, you should use the NOT LOGGED attribute only when data is being used by a single task, where the table space can be recovered if errors occur.

To change the logging attribute of a table space, complete the following steps:

1. Issue an ALTER TABLESPACE statement.
2. Specify the LOGGED or NOT LOGGED attribute.
 - **LOGGED:** Specifies that changes made to data in this table space are to be recorded on the log.
 - **NOT LOGGED:** Specifies that changes made to data in this table space are not to be recorded on the log. The NOT LOGGED attribute suppresses the logging of undo and redo information.

The change in logging applies to all tables in this table space and also applies to all indexes on those tables, as well as associated LOB and XML table spaces.

The NOT LOGGED attribute

The NOT LOGGED attribute for a table space indicates that changes to tables in the table space are not recorded on the log.

You should use the NOT LOGGED attribute only for situations where the data is in effect being duplicated. If the data is corrupted, you can recreate it from its original source, rather than from an image copy and the log. For example, you could use NOT LOGGED when you are inserting large volumes of data with the INSERT statement.

Restrictions: If you use the NOT LOGGED logging attribute, you can use images copies for recovery with certain restrictions.

- The logging attribute applies to all partitions of a table space. NOT LOGGED suppresses only the logging of undo and redo information; control records of the table space continue to be logged.
- You can take full and incremental SHRLEVEL REFERENCE image copies even though the table space has the NOT LOGGED attribute. You cannot take SHRLEVEL CHANGE copies because the NOT LOGGED attribute suppresses the logging of changes necessary for recovery.
- System-level backups taken with the BACKUP SYSTEM utility will contain NOT LOGGED objects, but they cannot be used for object level recovery of NOT LOGGED objects.

You can set the NOT LOGGED attribute when creating or altering table spaces.

When to use the NOT LOGGED attribute

Consider using the NOT LOGGED attribute in the following specific situations:

- For tables that summarize information in other tables, including materialized query tables, where the data can be easily re-created.
- When you are inserting large volumes of data with the INSERT statement.
- When you are using LOAD RESUME.

To utilize table spaces that are not logged, when using LOAD RESUME, complete the following steps:

1. Alter the table space to not logged before the load. Altering the logging attribute requires exclusive use of the table space.
2. Run the LOAD utility with the RESUME option.
3. Before normal update processing, alter the table space back to logged, and make an image copy of the table space.

Restriction: Online LOAD RESUME against a table space that is not logged is not recoverable if the load fails. If an online load attempt fails and rollback is necessary, the not logged table space is placed in LPL RECOVER-pending status. If this happens, you must terminate the LOAD job, recover the data from a prior image copy, and restart the online LOAD RESUME.

What happens when you change the logging attribute

Altering the logging attribute of a table space from LOGGED to NOT LOGGED establishes a recoverable point for the table space. Indexes automatically inherit the logging attribute of their table spaces. For the index, the change establishes a recoverable point that can be used by the RECOVER utility. Each subsequent image copy establishes another recoverable point for the table space and its associated indexes if the image copy is taken as a set.

Altering the logging attribute of a table space from NOT LOGGED to LOGGED marks the table space as COPY-pending (a recoverable point must be established before logging resumes). The indexes on the tables in the table space that have the COPY YES attribute are unchanged.

Changing the space allocation for user-managed data sets

If the table space is supported by user-managed data sets, you must complete several steps to change the space allocation.

GUIP To change the space allocation for user-managed data sets, complete the following steps:

1. Run the REORG TABLESPACE utility, and specify the UNLOAD PAUSE option.
2. Make the table space unavailable with the STOP DATABASE command and the SPACENAM option after the utility completes the unload and stops.
3. Delete and redefine the data sets.
4. Resubmit the utility job with the RESTART(PHASE) parameter specified on the EXEC statement.

The job now uses the new data sets when reloading.

Use of the REORG utility to extend data sets causes the newly acquired free space to be distributed throughout the table space rather than to be clustered at the end.

GUPI

Dropping, recreating, or converting a table space

To make changes to a table space, such as changing SEGSIZE, changing the number of partitions, or converting it to a large table space, you must drop the table space and then recreate it.

GUPI

To change or convert a table space by dropping the table space and then recreating it:

1. Locate the original CREATE TABLE statement and all authorization statements for all tables in the table space (for example, TA1, TA2, TA3, ... in TS1). If you cannot find these statements, query the DB2 catalog to determine the table's description, the description of all indexes and views on it, and all users with privileges on the table.

2. In another table space (for example, TS2), create tables TB1, TB2, TB3, ... identical to TA1, TA2, TA3, For example, use a statements such as:

```
CREATE TABLE TB1 LIKE TA1 IN TS2;
```

3. Optional: If necessary, unload the data. For example, use a statement such as:

```
REORG TABLESPACE DSN8D91A.TS1 LOG NO SORTDATA UNLOAD EXTERNAL;
```

Another way of unloading data from your old tables and loading the data into new tables is by using the INCURSOR option of the LOAD utility. This option uses the DB2 cross-loader function.

4. Optional: Alternatively, instead of unloading the data, you can insert the data from your old tables into the new tables by executing an INSERT statement for each table. For example:

```
INSERT INTO TB1  
  SELECT * FROM TA1;
```

If a table contains a ROWID column or an identity column and you want to keep the existing column values, you must define that column as GENERATED BY DEFAULT. If the ROWID column or identity column is defined with GENERATED ALWAYS, and you want DB2 to generate new values for that column, specify OVERRIDING USER VALUE on the INSERT statement with the subselect.

5. Drop the table space. For example, use a statement such as:

```
DROP TABLESPACE TS1;
```

The compression dictionary for the table space is dropped, if one exists. All tables in TS1 are dropped automatically.

6. Commit the DROP statement. You must commit the DROP TABLESPACE statement before creating a table space or index with the same name. When you drop a table space, all entries for that table space are dropped from SYSIBM.SYSCOPY. This makes recovery for that table space impossible from previous image copies.

7. Create the new table space, TS1, and grant the appropriate user privileges. You can also create a partitioned table space. For example, use a statement such as:

```
CREATE TABLESPACE TS1  
  IN DSN8D91A  
  USING STOGROUP DSN8G910
```

```

        PRIQTY 4000
        SECQTY 130
        ERASE NO
NUMPARTS 95
(PARTITION 45 USING STOGROUP DSN8G910
        PRIQTY 4000
        SECQTY 130
        COMPRESS YES,
PARTITION 62 USING STOGROUP DSN8G910
        PRIQTY 4000
        SECQTY 130
        COMPRESS NO)
LOCKSIZE PAGE
BUFFERPOOL BP1
CLOSE NO;

```

8. Create new tables TA1, TA2, TA3,
9. Recreate indexes on the tables, and grant user privileges on those tables.
10. Execute an INSERT statement for each table. For example:

```

INSERT INTO TA1
  SELECT * FROM TB1;

```

If a table contains a ROWID column or an identity column and you want to keep the existing column values, you must define that column as GENERATED BY DEFAULT. If the ROWID column or identity column is defined with GENERATED ALWAYS, and you want DB2 to generate new values for that column, specify OVERRIDING USER VALUE on the INSERT statement with the subselect.

11. Drop table space TS2. If a table in the table space has been created with RESTRICT ON DROP, you must alter that table to remove the restriction before you can drop the table space.
12. Notify users to recreate any synonyms they had on TA1, TA2, TA3,
13. REBIND plans and packages that were invalidated as a result of dropping the table space. 

Rebalancing data in partitioned table spaces

When data becomes out of balance in partitioned table spaces, performance can decrease. It is possible to improve performance by rebalancing the partitions to redistribute the data.

If an index established the partitions for the table space, you can use the ALTER INDEX statement for that index and a REORG job to shift data among the affected partitions. The result is that data is balanced according to your specifications. You can rebalance data by changing the limit key values of all or most of the partitions. The limit key is the highest value of the index key for a partition. You roll the changes through the partitions one or more at a time, making relatively small parts of the data unavailable at any given time. For more information about rebalancing data for index-controlled partitioned table spaces by using the ALTER INDEX statement, see *The Official Introduction to DB2 UDB for z/OS*.

In addition, for index-controlled and table-controlled partitioned table spaces, you can use the REBALANCE option of the REORG TABLESPACE utility to shift data among the partitions. When you use REORG TABLESPACE with REBALANCE, DB2 automatically changes the limit key values for the partitioned table space.

 To redistribute skewed data, complete the following steps:

Determine your DB2 version level.

- Version 7 and earlier: Issue the ALTER TABLE VALUES or ALTER INDEX VALUES statements, followed by a REORG utility job, to shift data among the affected partitions.
- Version 8 and after: Use the REBALANCE keyword of the REORG utility to reorganize selected partitions without affecting data availability.



Altering a page set to contain DB2-defined extents

The RECOVER utility can run the DFSMSdss RESTORE command, which extends a data set differently than DB2, so after this command runs, you must alter the page set to contain extents that are defined by DB2.

To alter a page set to contain extents that are defined by DB2:

1. Issue the ALTER TABLESPACE SQL statement.
After you use the ALTER TABLESPACE statement, the new values take effect only when you use REORG or LOAD REPLACE.
2. Enlarge the primary and secondary space allocation values for DB2-managed data sets.

Using the RECOVER utility again does not resolve the extent definition.

For user-defined data sets, define the data sets with larger primary and secondary space allocation values.

Altering DB2 tables

When you alter a table, you do not change the data in the table; you merely change the specifications that you used in creating the table.



To alter a table, issue the ALTER TABLE statement. With the ALTER TABLE statement, you can:

- Add a new column to a table.
- Rename a column.
- Change the data type of a column, with certain restrictions.
- Add or drop a parent key or a foreign key.
- Add or drop a table check constraint.
- Add a new partition to a table space.
- Change the boundary between partitions, extend the boundary of the last partition, rotate partitions, or instruct DB2 to insert rows at the end of a table or appropriate partition.
- Register an existing table as a materialized query table, change the attributes of a materialized query table, or change a materialized query table to a base table.
- Change the VALIDPROC clause.
- Change the DATA CAPTURE clause.
- Change the AUDIT clause by using the options ALL, CHANGES, or NONE.
- Add or drop the restriction on dropping the table and the database and table space that contain the table.

- Alter the length of a VARCHAR column using the SET DATA TYPE VARCHAR clause.
- Add or drop a clone table.
- Alter APPEND attributes.
- Drop the default value for a column.

GUIP

Related reference

"ALTER TABLE" (DB2 SQL Reference)

Adding a new column to a table

When you use ALTER TABLE to add a new column to a table, the new column becomes the rightmost column of the table and the table space is placed in an advisory REORG-pending (AREO*) state.

The physical records are not actually changed until values are inserted in the new column. Plans and packages are not invalidated unless the new column is a TIME, TIMESTAMP, or DATE. However, to use the new column in a program, you need to modify and recompile the program and bind the plan or package again. You might also need to modify any program containing a static SQL statement SELECT *, which returns the new column after the plan or package is rebound. You must also modify any INSERT statement not containing a column list.

Access time to the table is not affected immediately, unless the record was previously fixed length. If the record was fixed length, the addition of a new column causes DB2 to treat the record as variable length and then access time is affected immediately.

GUIP

To change the records to fixed length, complete the following steps:

1. Run REORG with COPY on the table space, using the inline copy.
2. Run the MODIFY utility with the DELETE option to delete records of all image copies that were made before the REORG you ran in step 1.
3. Create a unique index if you add a column that specifies PRIMARY KEY.

GUIP

Tip: Inserting values in the new column might also degrade performance by forcing rows onto another physical page. You can avoid this situation by creating the table space with enough free space to accommodate normal expansion. If you already have this problem, run REORG on the table space to fix it.

You can define the new column as NOT NULL by using the DEFAULT clause unless the column has a ROWID data type or is an identity column. If the column has a ROWID data type or is an identity column, you must specify NOT NULL without the DEFAULT clause. You can let DB2 choose the default value, or you can specify a constant or the value of the CURRENT SQLID or USER special register as the value to be used as the default. When you retrieve an existing row from the table, a default value is provided for the new column. Except in the following cases, the value for retrieval is the same as the value for insert:

- For columns of data type DATE, TIME, and TIMESTAMP, the retrieval defaults are:

Data type

Default for retrieval

DATE

0001-01-01

TIME

00.00.00

TIMESTAMP

0001-01-01-00.00.00.000000

- For DEFAULT USER and DEFAULT CURRENT SQLID, the retrieved value for rows that existed before the column was added is the value of the special register when the column was added.

If the new column is a ROWID column, DB2 returns the same, unique row ID value for a row each time you access that row. Reorganizing a table space does not affect the values on a ROWID column. You cannot use the DEFAULT clause for ROWID columns.

If the new column is an identity column (a column that is defined with the AS IDENTITY clause), DB2 places the table space in REORG-pending (REORP) status, and access to the table space is restricted until the table space is reorganized. When the REORG utility is run, DB2

- Generates a unique value for the identity column of each existing row
- Physically stores these values in the database
- Removes the REORP status

You cannot use the DEFAULT clause for identity columns.

If the new column is a short string column, you can specify a field procedure for it. If you do specify a field procedure, you cannot also specify NOT NULL.

GUPI The following example adds a new column to the table DSN8910.DEPT, which contains a location code for the department. The column name is LOCATION_CODE, and its data type is CHAR (4).

```
ALTER TABLE DSN8910.DEPT
  ADD LOCATION_CODE CHAR (4);
```

GUPI

Specifying a default value when altering a column

You can use the ALTER TABLE statement to add, change, or remove the default value for a column.

Restrictions: **GUPI**

- You cannot alter a column to specify a default value if the table is referenced by a view.
- If the column is part of a unique constraint or unique index, the new default to a value should not be the same as a value that already exists in the column.
- The new default value applies only to new rows.

To alter the default value for a column:

1. To set the default value, issue the following statement:

```
ALTER TABLE table-name ALTER COLUMN column-name
  SET default-clause
```

You can use this statement to add a default value for a column that does not already have one, or to change the existing default value.

2. To remove the default value without specifying a new one, issue the following statement:

```
ALTER TABLE table-name ALTER COLUMN column-name  
DROP DEFAULT
```

GUIP

Altering the data type of a column

You can use the ALTER TABLE statement to change the data types of columns in existing tables in several ways.

In general, DB2 can alter a data type if the data can be converted from the old type to the new type without truncation or without losing arithmetic precision.

Restriction: The column that you alter cannot be a part of a referential constraint, have a field procedure, be defined as an identity column, or be defined as a column of a materialized query table.

When you alter the data type of a column in a table, DB2 creates a new version for the table space that contains the data rows of the table.

GUIP To alter the data type of a column, complete the following steps:

1. Issue an ALTER TABLE statement.
2. Specify the data type change that you would like to make. Potential changes include:
 - Altering the length of fixed-length or varying-length character data types, and the length of fixed-length or varying-length graphic data types.
 - Switching between fixed-length and varying-length types for character and graphic data.
 - Switching between compatible numeric data types.

GUIP

What happens to the column

When you change the data type of a column by using the ALTER TABLE statement, the new definition of the column is stored in the catalog.

When you retrieve table rows, the columns are materialized in the format that is indicated by the catalog, but the data is not saved in that format. When you change or insert a row, the entire row is saved in the format that is indicated by the catalog. When you reorganize the table space (or perform a load replace), DB2 reloads the data in the table space according to the format of the current definitions in the catalog.

Example: **GUIP** Assume that a table contains basic account information for a small bank. The initial account table was created many years ago in the following manner:

```
CREATE TABLE ACCOUNTS (
  ACCTID      DECIMAL(4,0)  NOT NULL,
  NAME        CHAR(20)     NOT NULL,
  ADDRESS     CHAR(30)     NOT NULL,
  BALANCE     DECIMAL(10,2) NOT NULL)
IN dbname.tname;
```

The columns, as currently defined, have the following problems:

- The ACCTID column allows for only 9999 customers.
- The NAME and ADDRESS columns were defined as fixed-length columns, which means that some of the longer values are truncated and some of the shorter values are padded with blanks.
- The BALANCE column allows for amounts up to 99 999 999.99, but inflation rates demand that this column hold larger numbers.

By altering the column data types in the following ways, you can make the columns more appropriate for the data that they contain. The INSERT statement that follows shows the kinds of values that you can now store in the ACCOUNTS table.

```
ALTER TABLE ACCOUNTS ALTER COLUMN NAME    SET DATA TYPE VARCHAR(40);
ALTER TABLE ACCOUNTS ALTER COLUMN ADDRESS SET DATA TYPE VARCHAR(60);
ALTER TABLE ACCOUNTS ALTER COLUMN BALANCE SET DATA TYPE DECIMAL(15,2);
ALTER TABLE ACCOUNTS ALTER COLUMN ACCTID  SET DATA TYPE INTEGER;
COMMIT;
```

```
INSERT INTO ACCOUNTS (ACCTID, NAME, ADDRESS, BALANCE)
VALUES (123456, 'LAGOMARSINO, MAGDALENA',
        '1275 WINTERGREEN ST, SAN FRANCISCO, CA, 95060', 0);
COMMIT;
```

The NAME and ADDRESS columns can now handle longer values without truncation, and the shorter values are no longer padded. The BALANCE column is extended to allow for larger dollar amounts. DB2 saves these new formats in the catalog and stores the inserted row in the new formats.

Recommendation: If you change both the length and the type of a column from fixed-length to varying-length by using one or more ALTER statements, issue the ALTER statements within the same unit of work. Reorganize immediately so that the format is consistent for all of the data rows in the table.



What happens to an index on the column

Altering the data type of a column that is contained in an index has implications for the index.

Example:  Assume that the following indexes are defined on the ACCOUNTS table:

```
CREATE INDEX IX1 ON ACCOUNTS(ACCTID);
CREATE INDEX IX2 ON ACCOUNTS(NAME);
```

When the data type of the ACCTID column is altered from DECIMAL(4,0) to INTEGER, the IX1 index is placed in a REBUILD-pending (RBDP) state. Similarly, when the data type of the NAME column is altered from CHAR(20) to VARCHAR(40), the IX2 index is placed in an RBDP state. These indexes cannot be

accessed until they are rebuilt from the data. 

Index inaccessibility and data availability

Whenever possible, DB2 tries to avoid using inaccessible indexes in an effort to increase data availability. DB2 allows you to insert into, and delete from, tables that have non-unique indexes that are in an RBDP state. DB2 also allows you to delete from tables that have unique indexes that are in an RBDP state.

REBUILD INDEX with the SHRLEVEL CHANGE option allows read and write access to the data for most of the rebuild operation.

In certain situations, when an index is inaccessible, DB2 can bypass the index to allow applications access to the underlying data. In these situations, DB2 offers accessibility at the expense of performance. In making its determination of the best access path, DB2 can bypass an index under the following circumstances:

- Dynamic PREPAREs

DB2 avoids choosing an index that is in an RBDP state. Bypassing the index typically degrades performance, but provides availability that would not be possible otherwise.

- Cached PREPAREs

DB2 avoids choosing an index that is both in an RBDP state and within a cached PREPARE statement, because the dynamic cache for the table is invalidated whenever an index is put into an RBDP state.

In the case of static BINDs, DB2 might choose an index that is in an RBDP state as the best access path. DB2 does so by making the optimistic assumption that the index will be available by the time it is actually used. (If the index is not available at that time, an application can receive a resource unavailable message.)

Padding

Whether an index is padded or not padded depends on when the index was created and whether the index contains any varying length columns. In pre-Version 8 releases, an index is padded by default. In new-function mode for Version 8 or later, an index is not padded by default for new installations of DB2, but will be padded for migrated systems which came from at least Version 7 and later. In Version 8 or later, this default behavior is specified by the value of the PAD INDEX BY DEFAULT parameter on the DSNTIPE installation panel, which can be set to YES or NO.

When an index is not padded, the value of the PADDED column of the SYSINDEXES table is set to N. An index is only considered not padded when it is created with at least one varying length column and either:

- The NOT PADDED keyword is specified.
- The default padding value is NO.

When an index is padded, the value of the PADDED column of the SYSINDEXES table is set to Y. An index is padded if it is created with at least one varying length column and either:

- The PADDED keyword is specified
- The default padding is YES.

In the example of the ACCOUNTS table, the IX2 index retains its padding attribute. The padding attribute of an index is altered only if the value is

inconsistent with the current state of the index. The value can be inconsistent, for example, if you change the value of the PADDED column in the SYSINDEXES table after creating the index.

Consider the following information when you migrate indexes from one version of DB2 to the next version, or when you install a new DB2 subsystem and create indexes:

- If the index was migrated from a pre-Version 8 release, the index is padded by default. In this case, the value of the PADDED column of the SYSINDEXES table is blank (PADDED = ' '). The PADDED column is also blank when there are no varying length columns.
- If a subsystem has been migrated from Version 7 to Version 8 compatibility mode or new-function mode, the default is to pad all indexes that have a key with at least one varying length column. In this case, the value of the PADDED column of the SYSINDEXES table is YES (PADDED = 'Y').
- If an installed subsystem is new to Version 8 or later, and the install is done directly into new-function mode for Version 8 or later, indexes created with at least one varying length column are not padded by default. In this case, the PADDED column of the SYSINDEXES table is set to NO (PADDED = 'N').

Table space versions

DB2 creates a table space version each time that you commit one or more of the following schema changes by using the ALTER TABLE statement:

- Extend the length of a character (char data type) or graphic (graphic data type) column
- Change the type of a column within character data types (char, varchar)
- Change the type of a column within graphic data types (graphic, vargraphic)
- Change the type of a column within numeric data types (small integer, integer, float, real, float8, double, decimal)
- Add a column to a table
- Extend the length of a varying character (varchar data type) or varying graphic (vargraphic data type) column, if the table already has a version number that is greater than 0

Exceptions: DB2 does not create a table space version under the following circumstances:

- When the table space was created with DEFINE NO
- When you extend the length of a varying character (varchar data type) or varying graphic (vargraphic data type) column, and the table does not have a version number yet.
- When you specify the same data type and length that a column currently has, such that its definition does not actually change.
- When you add a column to a table in a non-partitioned table space, and the table is already versioned with no data in the current committed version format.

GUPI DB2 creates only one table space version if, in the same unit of work, you make multiple schema changes. If you make these same schema changes in separate units of work, each change results in a new table space version. For example, the first three ALTER TABLE statements in the following example are all associated with the same table space version. The scope of the first COMMIT statement encompasses all three schema changes. The last ALTER TABLE statement

is associated with the next table space version. The scope of the second COMMIT statement encompasses a single schema change.

```
ALTER TABLE ACCOUNTS ALTER COLUMN NAME SET DATA TYPE VARCHAR(40);  
ALTER TABLE ACCOUNTS ALTER COLUMN ADDRESS SET DATA TYPE VARCHAR(60);  
ALTER TABLE ACCOUNTS ALTER COLUMN BALANCE SET DATA TYPE DECIMAL(15,2);  
COMMIT;
```

```
ALTER TABLE ACCOUNTS ALTER COLUMN ACCTID SET DATA TYPE INTEGER;  
COMMIT;
```

Reorganizing table spaces

After you commit a schema change, DB2 puts the affected table space into an advisory REORG-pending (AREO*) state. The table space stays in this state until you run the REORG TABLESPACE utility, which reorganizes the table space and applies the schema changes.

DB2 uses table space versions to maximize data availability. Table space versions enable DB2 to keep track of schema changes and simultaneously, provide users with access to data in altered table spaces. When users retrieve rows from an altered table, the data is displayed in the format that is described by the most recent schema definition, even though the data is not currently stored in this format. The most recent schema definition is associated with the *current* table space version.

Although data availability is maximized by the use of table space versions, performance might suffer because DB2 does not automatically reformat the data in the table space to conform to the most recent schema definition. DB2 defers any reformatting of existing data until you reorganize the table space with the REORG TABLESPACE utility. The more ALTER statements you commit between reorganizations, the more table space versions DB2 must track, and the more performance can suffer.

Recommendation: Run the REORG TABLESPACE utility as soon as possible after a schema change to correct any performance degradation that might occur and to keep performance at its highest level.

Recycling table space version numbers

DB2 can store up to 256 table space versions, numbered sequentially from 0 to 255. (The next consecutive version number after 255 is 1. Version number 0 is never reused; it is reserved for the original version of the table space.) The versions that are associated with schema changes that have not been applied yet are considered to be “in use,” and the range of used versions is stored in the catalog. In-use versions can be recovered from image copies of the table space, if necessary. To determine the range of version numbers currently in use for a table space, query the OLDEST_VERSION and CURRENT_VERSION columns of the SYSIBM.SYSTABLESPACE catalog table.

To prevent DB2 from running out of table space version numbers (and to prevent subsequent ALTER statements from failing), you must recycle unused table space version numbers regularly by running the MODIFY RECOVERY utility. Version numbers are considered to be “unused” if the schema changes that are associated with them have been applied and there are no image copies that contain data at those versions. If all reusable version numbers (1 to 255) are currently in use, you

must reorganize the table space by running REORG TABLESPACE before you can recycle the version numbers. 

Altering a table for referential integrity

You can alter a table to add, change, or remove referential constraints.

If you plan to let DB2 enforce referential integrity in a set of tables, see the *DB2 Application Programming and SQL Guide* for a description of the requirements for referential constraints. DB2 does not enforce informational referential constraints.

Adding referential constraints to existing tables

This task presents an example of how to add referential constraints to existing tables. For the example in this task assume that the tables in the sample application already exist, have the appropriate column definitions, and are already populated. This task refers to the DB2 sample activity, project, project activity, employee, and department tables.

Suppose that you want to define relationships among the sample tables by adding primary and foreign keys with the ALTER TABLE statement. The rules for these relationships are as follows:

- An existing table must have a unique index on its primary key columns before you can add the primary key. The index becomes the primary index.
- The parent key of the parent table must be added before the corresponding foreign key of the dependent table.

You can build the same referential structure in several different ways. The following example sequence does not have the fewest number of possible operations, but it is perhaps the simplest to understand.

1. Create a unique index on the primary key columns for any table that does not already have one.
2. For each table, issue an ALTER TABLE statement to add its primary key.

In the next steps, you issue an ALTER TABLE statement to add foreign keys for each table except the activity table. This leaves the table space in CHECK-pending status, which you reset by running the CHECK DATA utility with the DELETE(YES) option.

CHECK DATA deletes are not bound by delete rules; they cascade to all descendents of a deleted row, which can be disastrous. For example, if you delete the row for department A00 from the department table, the delete might propagate through most of the referential structure. The remaining steps prevent deletion from more than one table at a time.

3. Add the foreign keys for the department table and run CHECK DATA DELETE(YES) on its table space. Correct any rows in the exception table, and use INSERT to replace them in the department table. This table is now consistent with existing data.
4. Drop the foreign key on MGRNO in the department table. This drops the association of the department table with the employee table, without changing the data of either table.
5. Add the foreign key to the employee table, run CHECK DATA again, and correct any errors. If errors are reported, be particularly careful not to make any row inconsistent with the department table when you make corrections.

6. Again, add the foreign key on MGRNO to the department table, which again leaves the table space in CHECK-pending status. Run CHECK DATA. If you have not changed the data since the previous check, you can use DELETE(YES) with no fear of cascading deletions.
7. For each of the following tables, in the order shown, add its foreign keys, run CHECK DATA DELETE(YES), and correct any rows that are in error:
 - a. Project table
 - b. Project activity table
 - c. Employee to project activity table

Adding parent keys and foreign keys

You can add parent keys, both primary and unique, and foreign keys to an existing table.

GUIP To add a key to a table:

1. Choose the type of key that you want to add.
2. Add the key by using the ALTER TABLE statement.

Option	Description
Adding a primary key	To add a primary key to an existing table, use the PRIMARY KEY clause in an ALTER TABLE statement. For example, if the department table and its index XDEPT1 already exist, create its primary key by issuing the following statement: <pre>ALTER TABLE DSN8910.DEPT ADD PRIMARY KEY (DEPTNO);</pre>
Adding a unique key	To add a unique key to an existing table, use the UNIQUE clause of the ALTER TABLE statement. For example, if the department table has a unique index defined on column DEPTNAME, you can add a unique key constraint, KEY_DEPTNAME, consisting of column DEPTNAME by issuing the following statement: <pre>ALTER TABLE DSN8910.DEPT ADD CONSTRAINT KEY_DEPTNAME UNIQUE (DEPTNAME);</pre>
Adding a foreign key	To add a foreign key to an existing table, use the FOREIGN KEY clause of the ALTER TABLE statement. The parent key must exist in the parent table before you add the foreign key. For example, if the department table has a primary key defined on the DEPTNO column, you can add a referential constraint, REFKEY_DEPTNO, on the DEPTNO column of the project table by issuing the following statement: <pre>ALTER TABLE DSN8910.PROJ ADD CONSTRAINT REFKEY_DEPTNO FOREIGN KEY (DEPTNO) REFERENCES DSN8910.DEPT ON DELETE RESTRICT;</pre>

GUIP

Implications of adding parent or foreign keys:

When you add parent keys and foreign keys to an existing table, consider the following restrictions and implications.

- If you add a primary key, the table must already have a unique index on the key columns. If multiple unique indexes include the primary key columns, the index that was most recently created on the key columns becomes the primary index. Because of the unique index, no duplicate values of the key exist in the table; therefore you do not need to check the validity of the data.
- If you add a unique key, the table must already have a unique index with a key that is identical to the unique key. If multiple unique indexes include the primary key columns, DB2 arbitrarily chooses a unique index on the key columns to enforce the unique key. Because of the unique index, no duplicate values of the key exist in the table; therefore you do not need to check the validity of the data.
- You can use only one FOREIGN KEY clause in each ALTER TABLE statement; if you want to add two foreign keys to a table, you must execute two ALTER TABLE statements.
- If you add a foreign key, the parent key and unique index of the parent table must already exist. Adding the foreign key requires the ALTER privilege on the dependent table and either the ALTER or REFERENCES privilege on the parent table.
- Adding a foreign key establishes a referential constraint relationship. DB2 does not validate the data when you add the foreign key. Instead, if the table is populated (or, in the case of a nonsegmented table space, if the table space has ever been populated), the table space that contains the table is placed in CHECK-pending status, just as if it had been loaded with ENFORCE NO. In this case, you need to execute the CHECK DATA utility to clear the CHECK-pending status.
- You can add a foreign key with the NOT ENFORCED option to create an informational referential constraint. This action does not leave the table space in CHECK-pending status, and you do not need to execute CHECK DATA.

Dropping parent keys and foreign keys

You can drop parent keys, both primary and unique, and foreign keys from an existing table.

Before you drop a foreign key or a parent key, consider carefully the effects on your application programs. The primary key of a table serves as a permanent, unique identifier of the occurrences of the entities it describes. Application programs often depend on that identifier. The foreign key defines a referential relationship and a delete rule. Without the key, your application programs must enforce the constraints.

GUPI To drop a key, complete the following steps:

1. Choose the type of key that you want to drop.

2. Drop the key by using the ALTER TABLE statement.

Option	Description
Dropping a foreign key	When you drop a foreign key using the DROP FOREIGN KEY clause of the ALTER TABLE statement, DB2 drops the corresponding referential relationships. (You must have the ALTER privilege on the dependent table and either the ALTER or REFERENCES privilege on the parent table.) If the referential constraint references a unique key that was created implicitly, and no other relationships are dependent on that unique key, the implicit unique key is also dropped.
Dropping a unique key	When you drop a unique key using the DROP UNIQUE clause of the ALTER TABLE statement, DB2 drops all the referential relationships in which the unique key is a parent key. The dependent tables no longer have foreign keys. (You must have the ALTER privilege on any dependent tables.) The table's unique index that enforced the unique key no longer indicates that it enforces a unique key, although it is still a unique index.
Dropping a primary key	When you drop a primary key using the DROP PRIMARY KEY clause of the ALTER TABLE statement, DB2 drops all the referential relationships in which the primary key is a parent key. The dependent tables no longer have foreign keys. (You must have the ALTER privilege on any dependent tables.) The table's primary index is no longer primary, although it is still a unique index.

GUPI

Adding or dropping table check constraints

You can add or drop constraint by using the ALTER TABLE statement.

GUPI To add or drop check constraints, complete the following steps:

1. Decide to add or drop a constraint.

2. Use the ALTER TABLE statement to add or drop the constraint.

Option	Description
<p>Add constraint</p>	<p>You can define a check constraint on a table by using the ADD CHECK clause of the ALTER TABLE statement. If the table is empty, the check constraint is added to the description of the table.</p> <p>If the table is not empty, what happens when you define the check constraint depends on the value of the CURRENT RULES special register, which can be either STD or DB2.</p> <ul style="list-style-type: none"> • If the value is STD, the check constraint is enforced immediately when it is defined. If a row does not conform, the table check constraint is not added to the table and an error occurs. • If the value is DB2, the check constraint is added to the table description but its enforcement is deferred. Because some rows in the table might violate the check constraint, the table is placed in check-pending status. <p>The ALTER TABLE statement that is used to define a check constraint always fails if the table space or partition that contains the table is in a CHECK-pending status, the CURRENT RULES special register value is STD, and the table is not empty.</p>
<p>Drop constraint</p>	<p>To remove a check constraint from a table, use the DROP CONSTRAINT or DROP CHECK clauses of the ALTER TABLE statement. You must not use DROP CONSTRAINT on the same ALTER TABLE statement as DROP FOREIGN KEY, DROP CHECK, DROP PRIMARY KEY, or DROP UNIQUE.</p>



Adding a partition

You can use the ALTER TABLE statement to add a partition to an existing regular-partitioned or range-partitioned universal table space and to each partitioned index in the table space.

Restrictions:

- You cannot add a new partition to an existing partitioned table space if the table has LOB columns.
- You cannot add or alter a partition for a materialized query table.
- You cannot explicitly add a new partition to a table created in the partition-by-growth table space.

When you add a partition, DB2 uses the next physical partition that is not already in use until you reach the maximum number of partitions for the table space.

When DB2 manages your data sets, the next available data set is allocated for the table space and for each partitioned index. When you manage your own data sets, you must first define the data sets for the table space and the partitioned indexes before adding a new partition.

GUPI To add a partition:

1. Issue an ALTER TABLE statement.
2. Specify the ADD PARTITION option. **GUPI**

Example: Assume that a table space that contains a transaction table named TRANS is divided into 10 partitions, and each partition contains one year of data. Partitioning is defined on the transaction date, and the limit key value is the end of the year. The following table shows a representation of the table space.

Table 21. Initial table space with 10 partitions

Partition	Limit value	Data set name that backs the partition
P001	12/31/1994	catname.DSNDBx.dbname.psname.I0001.A001
P002	12/31/1995	catname.DSNDBx.dbname.psname.I0001.A002
P003	12/31/1996	catname.DSNDBx.dbname.psname.I0001.A003
P004	12/31/1997	catname.DSNDBx.dbname.psname.I0001.A004
P005	12/31/1998	catname.DSNDBx.dbname.psname.I0001.A005
P006	12/31/1999	catname.DSNDBx.dbname.psname.I0001.A006
P007	12/31/2000	catname.DSNDBx.dbname.psname.I0001.A007
P008	12/31/2001	catname.DSNDBx.dbname.psname.I0001.A008
P009	12/31/2002	catname.DSNDBx.dbname.psname.I0001.A009
P010	12/31/2003	catname.DSNDBx.dbname.psname.I0001.A010

GUPI Assume that you want to add a new partition to handle the transactions for the next year. To add a partition, issue the following statement:

```
ALTER TABLE TRANS ADD PARTITION ENDING AT ('12/31/2004');
```

GUPI

DB2 adds a new partition to the table space and to each partitioned index on the TRANS table. For the table space, DB2 uses the existing table space PRIQTY and SECQTY attributes of the previous partition for the space attributes of the new partition. For each partitioned index, DB2 uses the existing PRIQTY and SECQTY attributes of the previous index partition.

When the ALTER statement completes, you can use the new partition immediately if the table space is a large table space. In this case, the partition is not placed in a REORG-pending (REORP) state, because it extends the high-range values that were not previously used. For non-large table spaces, the partition is placed in a REORG-pending (REORP) state, because the last partition boundary was not previously enforced.

The following table shows a representative excerpt of the table space after the partition for the year 2004 was added.

Table 22. An excerpt of the table space after adding a new partition (P011)

Partition	Limit value	Data set name that backs the partition
P008	12/31/2001	catname.DSNDBx.dbname.psname.I0001.A008
P009	12/31/2002	catname.DSNDBx.dbname.psname.I0001.A009
P010	12/31/2003	catname.DSNDBx.dbname.psname.I0001.A010
P011	12/31/2004	catname.DSNDBx.dbname.psname.I0001.A011

GUPI If you want to specify the space attributes for a new partition, use the ALTER TABLESPACE and ALTER INDEX statements. For example, suppose the new partition is PARTITION 11 for the table space and the index. Issue the following statements to specify quantities for the PRIQTY, SECQTY, FREEPAGE, and PCTFREE attributes:

```
ALTER TABLESPACE tsname ALTER PARTITION 11
  USING STOGROUP stogroup-name
  PRIQTY 200 SECQTY 200
  FREEPAGE 20 PCTFREE 10;
```

```
ALTER INDEX index-name ALTER PARTITION 11
  USING STOGROUP stogroup-name
  PRIQTY 100 SECQTY 100
  FREEPAGE 25 PCTFREE 5;
```

GUPI

Recommendation: When you create a partitioned table space, you do not need to allocate extra partitions for expected growth. Instead, use ALTER TABLE ADD PARTITION statement to add partitions as needed.

Altering partitions

You can use the ALTER TABLE statement to change the boundary between partitions, to rotate the first partition to be the last partition, to extend the boundary of the last partition, or to instruct DB2 to insert rows at the end of a table or appropriate partition.

To alter a partition, complete the following step:

1. Issue an ALTER TABLE statement.
2. Specify the options that you want to change.

Example: Assume that a table space that contains a transaction table named TRANS is divided into 10 partitions, and each partition contains one year of data. Partitioning is defined on the transaction date, and the limit key value is the end of the year. The following table shows a representation of the table space.

Table 23. Table space with multiple partitions

Partition	Limit value	Data set name that backs the partition
P001	12/31/1994	catname.DSNDBx.dbname.psname.I0001.A001
P002	12/31/1995	catname.DSNDBx.dbname.psname.I0001.A002
P003	12/31/1996	catname.DSNDBx.dbname.psname.I0001.A003
P004	12/31/1997	catname.DSNDBx.dbname.psname.I0001.A004
P005	12/31/1998	catname.DSNDBx.dbname.psname.I0001.A005

Table 23. Table space with multiple partitions (continued)

Partition	Limit value	Data set name that backs the partition
P006	12/31/1999	catname.DSNDBX.dbname.psname.I0001.A006
P007	12/31/2000	catname.DSNDBX.dbname.psname.I0001.A007
P008	12/31/2001	catname.DSNDBX.dbname.psname.I0001.A008
P009	12/31/2002	catname.DSNDBX.dbname.psname.I0001.A009
P010	12/31/2003	catname.DSNDBX.dbname.psname.I0001.A010

Changing the boundary between partitions

You can change the boundary between partitions in two ways.

GUPI To change partition boundaries, complete the following step:

Decide on the partitions that you want change the boundary for.

Option	Description
ALTER TABLE	<p>Issue an ALTER TABLE statement with the ALTER PARTITION option.</p> <p>For example, assume that the year 2006 resulted in more data than was projected so that the allocation for partition 10 reached its maximum of 4 GB. In addition, the year 2005 resulted in less data than was projected. You want to change the boundary between partition 9 and partition 10 so that some of the data in partition 10 becomes part of the data in partition 9.</p> <p>To change the boundary, issue the following statement:</p> <pre>ALTER TABLE TRANS ALTER PARTITION 9 ENDING AT ('03/31/2006');</pre> <p>Now the data in the first quarter of the year 2006 is part of partition 9. The partitions on either side of the new boundary (partitions 9 and 10) are placed in REORG-pending (REORP) status and are not available until the partitions are reorganized.</p>
REORG TABLESPACE	<p>Use the REBALANCE option of the REORG utility.</p> <p>For example, you can rebalance the data in partitions 9 and 10 as follows:</p> <pre>REORG TABLESPACE dbname.tsname PART(9:10) REBALANCE</pre> <p>This method avoids leaving the partitions in a REORP state. When you use the REBALANCE option on partitions, DB2 automatically changes the limit key values.</p> <p>Restriction: You cannot use the REBALANCE option for partitioned-by-growth table spaces.</p>

|
|
|



Rotating partitions

You can use the ALTER TABLE statement to rotate the first partition to become the last partition.

Recommendation: When you create a partitioned table space, you do not need to allocate extra partitions for expected growth. Instead, use either ALTER TABLE ADD PARTITION to add partitions as needed or, if rotating partitions is appropriate for your application, use ALTER TABLE ROTATE PARTITION to avoid adding another partition.

Nullable partitioning columns: DB2 lets you use nullable columns as partitioning columns. But with table-controlled partitioning, DB2 can restrict the insertion of null values into a table with nullable partitioning columns, depending on the order of the partitioning key. After a rotate operation, if the partitioning key is ascending, DB2 prevents an INSERT of a row with a null value for the key column. If the partitioning key is descending, DB2 allows an INSERT of a row with a null value for the key column. The row is inserted into the first partition.

To rotate the first partition to be the last partition:

1. Issue an ALTER TABLE *table-name* statement with the ROTATE PARTITION option.
2. Optional: Run the RUNSTATS utility.

Example

For example, assume that the partition structure of the table space is sufficient through the year 2006. The following table shows a representation of the table space through the year 2006. When another partition is needed for the year 2007, you determine that the data for 1996 is no longer needed. You want to recycle the partition for the year 1996 to hold the transactions for the year 2007.

Table 24. An excerpt of a partitioned table space

Partition	Limit value	Data set name that backs the partition
P008	12/31/2004	catname.DSNDBx.dbname.psname.I0001.A008
P009	12/31/2005	catname.DSNDBx.dbname.psname.I0001.A009
P010	12/31/2006	catname.DSNDBx.dbname.psname.I0001.A010
P011	12/31/2007	catname.DSNDBx.dbname.psname.I0001.A011

To rotate the first partition for table TRANS to be the last partition, issue the following statement:

```
ALTER TABLE TRANS ROTATE PARTITION FIRST TO LAST
  ENDING AT ('12/31/2007') RESET;
```



For a table with limit values in ascending order, the data in the ENDING AT clause must be higher than the limit value for previous partitions. DB2 chooses the first partition to be the partition with the lowest limit value.

For a table with limit values in descending order, the data must be lower than the limit value for previous partitions. DB2 chooses the first partition to be the partition with the highest limit value.

GUPI The RESET keyword specifies that the existing data in the oldest partition is deleted, and no delete triggers are activated. **GUPI**

What happens:

- Because the oldest (or first) partition is P001, DB2 assigns the new limit value to P001. This partition holds all rows in the range between the new limit value of 12/31/2005 and the previous limit value of 12/31/2004.
- The RESET operation deletes all existing data. You can use the partition immediately after the ALTER completes. The partition is not placed in REORG-pending (REORP) status because it extends the high-range values that were not previously used.

The following table shows a representation of the table space after the first partition is rotated to become the last partition.

Table 25. Rotating the low partition to the end

Partition	Limit value	Data set name that backs the partition
P002	12/31/1997	catname.DSNDBX.dbname.psname.I0001.A002
P003	12/31/1998	catname.DSNDBX.dbname.psname.I0001.A003
P004	12/31/1999	catname.DSNDBX.dbname.psname.I0001.A004
P005	12/31/2000	catname.DSNDBX.dbname.psname.I0001.A005
P006	12/31/2001	catname.DSNDBX.dbname.psname.I0001.A006
P007	12/31/2002	catname.DSNDBX.dbname.psname.I0001.A007
P008	12/31/2003	catname.DSNDBX.dbname.psname.I0001.A008
P009	12/31/2004	catname.DSNDBX.dbname.psname.I0001.A009
P010	12/31/2005	catname.DSNDBX.dbname.psname.I0001.A010
P011	12/31/2006	catname.DSNDBX.dbname.psname.I0001.A011
P001	12/31/2007	catname.DSNDBX.dbname.psname.I0001.A001

Extending the boundary of the last partition

You can extend the last partition by using the ALTER TABLE statement.

- GUPI** To extend the boundary of the last partition, complete the following steps:
1. Issue an ALTER TABLE TRANS statement.
 2. Specify the ALTER PARTITION option. **GUPI**

Assume that you have decided to save the data for the year 2005 and the data for the year 2006 into one partition. You have rotated the first partition to be the last partition. The following table shows a representation of the table space through the year 2007.

Table 26. Rotating the low partition to the end

Partition	Limit value	Data set name that backs the partition
P002	12/31/1997	catname.DSNDBx.dbname.psname.I0001.A002
P003	12/31/1998	catname.DSNDBx.dbname.psname.I0001.A003
P004	12/31/1999	catname.DSNDBx.dbname.psname.I0001.A004
P005	12/31/2000	catname.DSNDBx.dbname.psname.I0001.A005
P006	12/31/2001	catname.DSNDBx.dbname.psname.I0001.A006
P007	12/31/2002	catname.DSNDBx.dbname.psname.I0001.A007
P008	12/31/2003	catname.DSNDBx.dbname.psname.I0001.A008
P009	12/31/2004	catname.DSNDBx.dbname.psname.I0001.A009
P010	12/31/2005	catname.DSNDBx.dbname.psname.I0001.A010
P011	12/31/2006	catname.DSNDBx.dbname.psname.I0001.A011
P001	12/31/2007	catname.DSNDBx.dbname.psname.I0001.A001

GUIP To extend the boundary of the last partition, issue the following statement:
ALTER TABLE TRANS ALTER PARTITION 1 ENDING AT ('12/31/2006');

GUIP

You can use the partition immediately after the ALTER completes. The partition is not placed in REORG-pending (REORP) status because it extends the high-range values that were not previously used.

Reverting to the previous boundary:

You can use the ALTER TABLE statement to change a partition boundary back to a previous setting.

GUIP To revert a partition boundary back to a previous boundary:

1. Issue an ALTER TABLE statement.
2. Specify the ALTER PARTITION option. This partition is placed in REORG-pending (REORP) status because some of the data could fall outside of the boundary that is defined by the limit key value of 12/31/2005. You can take either of the following corrective actions:
 - Run REORG with the DISCARD option to clear the REORG-pending status, set the new partition boundary, and discard the data rows that fall outside of the new boundary.
 - Add a new partition for the data rows that fall outside of the current partition boundaries. **GUIP**

For example, the following table shows a representation of the table space through the year 2007, where each year of data is saved into separate partitions. Assume that you changed the limit key for P001 to be 12/31/2006 so that the data for the year 2005 and the data for the year 2006 is saved into one partition.

Table 27. Rotating the low partition to the end

Partition	Limit value	Data set name that backs the partition
P002	12/31/1997	catname.DSNDBx.dbname.psname.I0001.A002
P003	12/31/1998	catname.DSNDBx.dbname.psname.I0001.A003
P004	12/31/1999	catname.DSNDBx.dbname.psname.I0001.A004
P005	12/31/2000	catname.DSNDBx.dbname.psname.I0001.A005
P006	12/31/2001	catname.DSNDBx.dbname.psname.I0001.A006
P007	12/31/2002	catname.DSNDBx.dbname.psname.I0001.A007
P008	12/31/2003	catname.DSNDBx.dbname.psname.I0001.A008
P009	12/31/2004	catname.DSNDBx.dbname.psname.I0001.A009
P010	12/31/2005	catname.DSNDBx.dbname.psname.I0001.A010
P011	12/31/2006	catname.DSNDBx.dbname.psname.I0001.A011
P001	12/31/2007	catname.DSNDBx.dbname.psname.I0001.A001

GUIP To change the limit key back to 12/31/2005, issue the following statement:
ALTER TABLE TRANS ALTER PARTITION 1 ENDING AT ('12/31/2005');

GUIP

Adding a partition when the last partition is in REORG-pending status:

You can add a partition to a table when the last partition is in REORG-pending (REORP) status.

GUIP To add a partition to a table when the last partition is in REORG-pending status:

1. Issue an ALTER TABLE statement.
2. Specify the ADD PARTITION option. **GUIP**

For example, assume that you extended the boundary of the last partition and then changed back to the previous boundary for that partition. The following table shows a representation of the table space through the year 2005. The last partition is in REORP.

Table 28. Rotating the low partition to the end

Partition	Limit value	Data set name that backs the partition
P002	12/31/1997	catname.DSNDBx.dbname.psname.I0001.A002
P003	12/31/1998	catname.DSNDBx.dbname.psname.I0001.A003
P004	12/31/1999	catname.DSNDBx.dbname.psname.I0001.A004
P005	12/31/2000	catname.DSNDBx.dbname.psname.I0001.A005
P006	12/31/2001	catname.DSNDBx.dbname.psname.I0001.A006
P007	12/31/2002	catname.DSNDBx.dbname.psname.I0001.A007
P008	12/31/2003	catname.DSNDBx.dbname.psname.I0001.A008
P009	12/31/2004	catname.DSNDBx.dbname.psname.I0001.A009
P010	12/31/2005	catname.DSNDBx.dbname.psname.I0001.A010

Table 28. Rotating the low partition to the end (continued)

Partition	Limit value	Data set name that backs the partition
P011	12/31/2006	catname.DSNDBX.dbname.psname.I0001.A011
P001	12/31/2007	catname.DSNDBX.dbname.psname.I0001.A001

You want to add a new partition with a limit key value of 12/31/2006. You can use ALTER TABLE ADD PARTITION because this limit key value is higher than the previous limit key value of 12/31/2005. Issue the following statement:

GUIP

```
ALTER TABLE TRANS ADD PARTITION ENDING AT ('12/31/2006');
```

GUIP

The new partition is placed in REORG-pending (REORP) status because it inherits the REORP state from the previous partition. You can now reorganize the table space or only the last two partitions without discarding any of the data rows.

Inserting rows at the end partition

Use the CREATE TABLE or ALTER TABLE statement to instruct DB2 on how to insert rows at the end of a partition.

To insert rows at the end of a partition, complete the following steps:

1. Issue a CREATE TABLE or ALTER TABLE statement:
2. Specify the APPEND option.
 - **YES:** Requests data rows to be placed into the table by disregarding the clustering during SQL INSERT and online LOAD operations. Rather than attempting to insert rows in cluster-preserving order, rows are appended at the end of the table or appropriate partition.
 - **NO:** Requests standard behavior of SQL INSERT and online LOAD operations, namely that they attempt to place data rows in a well clustered manner with respect to the value in the row's cluster key columns. NO is the default option.

After populating a table with the APPEND option in effect, you can achieve clustering by running the REORG utility.

Restriction: You cannot specify the APPEND option for tables created in LOB, XML or Work file table spaces.

Adding XML columns

XML columns can be added to regular relational tables using the ALTER TABLE statement. XML columns can also be added by using the CREATE TABLE statement.

When you add an XML column to a table, an XML table and XML table space are implicitly created to store the XML data. If the new XML column is the first XML column that you created for the table, DB2 also implicitly creates a BIGINT DOCID column to store a unique document identifier for the XML columns of a row.

DB2 also implicitly creates indexes. If this is the first XML column that you created for the table, DB2 implicitly creates an index on the DOCID column of the base table.

An XML column has several restrictions. The column cannot:

- Be specified as part of the primary key
- Be specified as part of a UNIQUE key
- Have a DEFAULT value specified
- Be specified as part of the FOREIGN KEY references clause
- Be specified as part of the REFERENCES table-name clause
- Be specified in the PARTITION BY RANGE clause
- Be used in a materialized query table even if the table is specified WITH NO DATA
- Be referenced in CHECK constraints
- Have GENERATED specified
- Have a FIELDPROC specified
- Have AS SECURITY LABEL specified
- Be added to a created temporary table
- The table that contains an XML column will not have its XML column value passed to a VALIDPROC
- Be part of a transition table

GUIP To add an XML column to an existing table:

Issue an ALTER TABLE statement, specifying the ADD *column-name* XML option.

For example,

```
ALTER TABLE orders ADD shipping_info XML;
```

GUIP

Altering materialized query tables

You can use the ALTER TABLE statement to register an existing table as a materialized query table, to change a materialized query table to a base table, or to change the attributes of a materialized query table.

Materialized query tables enable DB2 to use automatic query rewrite to optimize queries. Automatic query rewrite is a process that DB2 uses to examine a query and, if appropriate, to rewrite the query so that it executes against a materialized query table that has been derived from the base tables in the submitted query.

Registering an existing table as a materialized query table

You can take advantage of automatic query rewrite for an existing table by registering it as a materialized query table.

GUIP To register an existing table as a materialized query table, complete the following steps:

1. Issue an ALTER TABLE statement.
2. Specify the ADD MATERIALIZED QUERY AS option. **GUIP**

For example, assume that you have a very large transaction table named TRANS that contains one row for each transaction. The table has many columns, but you are interested in only the following columns:

- ACCTID, which is the customer account ID
- LOCID, which is the customer location ID
- YEAR, which holds the year of the transaction

GUIP You created another base table named TRANCOUNT that consists of these columns and a count of the number of rows in TRANS that are grouped by the account, location, and year of the transaction. Suppose that you repopulate TRANCOUNT periodically by deleting the rows and then by using the following INSERT statement:

```
INSERT INTO TRANCOUNT (ACCTID, LOCID, YEAR, CNT)
  SELECT ACCTID, LOCID, YEAR, COUNT(*)
  FROM TRANS
  GROUP BY ACCTID, LOCID, YEAR;
```

You want to take advantage of automatic query rewrite for TRANCOUNT by registering it as a materialized query table. You can do this by issuing the following ALTER TABLE statement:

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

This statement registers TRANCOUNT with its associated subselect as a materialized query table, and DB2 can now use it in automatic query rewrite. The data in TRANCOUNT remains the same, as specified by the DATA INITIALLY DEFERRED option.

You can still maintain the data, as specified by the MAINTAINED BY USER option, which means that you can continue to load, insert, update, or delete data. You can also use the REFRESH TABLE statement to populate the table. REFRESH DEFERRED indicates that the data in the table is the result of your most recent update or, if more recent, the result of a REFRESH TABLE statement.

The REFRESH TABLE statement deletes all the rows in a materialized query table, executes the fullselect in its definition, inserts the result into the table, and updates the catalog with the refresh timestamp and cardinality of the table. **GUIP**

Changing a materialized query table to a base table

You can use the ALTER TABLE statement to change a materialized query table into a base table.

GUIP To change a materialized query table to a base table, complete the following steps:

1. Issue an ALTER TABLE statement.
2. Specify the DROP MATERIALIZED QUERY option. For example,

```
ALTER TABLE TRANCOUNT DROP MATERIALIZED QUERY;
```

After you issue this statement, DB2 can no longer use the table for query optimization, and you cannot populate the table by using the REFRESH TABLE statement. **GUPI**

Changing the attributes of a materialized query table

You can use the ALTER TABLE statement to change the following attributes of an existing materialized query table:

GUPI To change the attributes of an existing materialized query table, complete the following steps:

1. Issue an ALTER TABLE statement.
2. Decide which attributes to alter.

Option	Description
Enable or disable automatic query rewrite.	By default, when you create or register a materialized query table, DB2 enables it for automatic query rewrite. To disable automatic query rewrite, issue the following statement: <code>ALTER TABLE TRANSCOUNT DISABLE QUERY OPTIMIZATION;</code>
Switch between system-maintained and user-maintained.	By default, a materialized query table is system-maintained; the only way you can change the data is by using the REFRESH TABLE statement. To change to a user-maintained materialized query table, issue the following statement: <code>ALTER TABLE TRANSCOUNT SET MAINTAINED BY USER;</code>
Change back to a system-maintained materialized query table.	Specify the MAINTAINED BY SYSTEM option.

GUPI

Changing the definition of a materialized query table

After you create a materialized query table, you can change the definition in one of two ways.

To change the definition of an existing materialized query table, complete either of the following steps:

- Optional: Drop and re-create the materialized query table with a different definition.
- Optional: Use ALTER TABLE to change the materialized query table into a base table. Then, change it back to a materialized query table with a different but equivalent definition (that is, with a different but equivalent SELECT for the query).

Altering the assignment of a validation routine

You can use the ALTER TABLE statement to make certain changes to a validation exit routine associated with a table, if one exists.

GUPI If you have a validation exit routine associated with a table, you can use the ALTER TABLE statement to make the following changes:

- Disassociate the validation routine from the table using the VALIDPROC NULL clause. The routine is no longer given control when DB2 accesses the table. For example:

```
ALTER TABLE DSN8910.EMP
  VALIDPROC NULL;
```

- Assign a new validation routine to the table using the VALIDPROC clause. (Only one validation routine can be connected to a table at a time; so if a validation routine already exists, DB2 disconnects the old one and connects the new routine.) Rows that existed before the connection of a new validation routine are not validated. In this example, the previous validation routine is disconnected and a new routine is connected with the program name EMPLNEWE:

```
ALTER TABLE DSN8910.EMP
  VALIDPROC EMPLNEWE;
```

◀ GUPI

To ensure that the rows of a table conform to a new validation routine, you must run the validation routine against the old rows. One way to accomplish this is to use the REORG and LOAD utilities as shown in the following steps:

1. Use REORG to reorganize the table space that contains the table with the new validation routine. Specify UNLOAD ONLY, as in this example:

```
REORG TABLESPACE DSN8D91A.DSN8S91E
  UNLOAD ONLY
```

This step creates a data set that is used as input to the LOAD utility.

2. Run LOAD with the REPLACE option, and specify a discard data set to hold any invalid records. For example,

```
LOAD INTO TABLE DSN8910.EMP
  REPLACE
  FORMAT UNLOAD
  DISCARDN SYSDISC
```

The EMPLNEWE validation routine validates all rows after the LOAD step has completed. DB2 copies any invalid rows into the SYSDISC data set.

Altering a table for capture of changed data

You can use the ALTER TABLE statement to have data changes to that table written to the log in an expanded format.

▶ GUPI To alter a table to capture changed data, complete the following steps:

1. Issue an ALTER TABLE statement.
2. Specify the DATA CAPTURE CHANGES option.

You can then retrieve the log by using a program such as the log apply feature of the Remote Recovery Data Facility (RRDF) program offering, or DB2 DataPropagator.

LOB values are not available for DATA CAPTURE CHANGES. To return a table back to normal logging, use DATA CAPTURE NONE. **◀ GUPI**

Changing an edit procedure or a field procedure

You cannot use ALTER TABLE to change the assignment of an edit procedure or a field procedure. However, with the assistance of DB2 utilities, you can change an existing edit procedure or field procedure.

To change an edit procedure or a field procedure for a table space in which the maximum record length is less than 32 KB, use the following procedure:

1. Run the UNLOAD utility or run the REORG TABLESPACE utility with the UNLOAD EXTERNAL option to unload the data and decode it using the existing edit procedure or field procedure.

These utilities generate a LOAD statement in the data set (specified by the PUNCHDDN option of the REORG TABLESPACE utility) that you can use to reload the data into the original table space.

If you are using the same edit procedure or field procedure for many tables, unload the data from all the table spaces that have tables that use the procedure.

2. Modify the code of the edit procedure or the field procedure.
3. After the unload operation is completed, stop DB2.
4. Link-edit the modified procedure, using its original name.
5. Start DB2.
6. Use the LOAD utility to reload the data. LOAD then uses the modified procedure or field procedure to encode the data.

To change an edit procedure or a field procedure for a table space in which the maximum record length is greater than 32 KB, use the DSNTIAUL sample program to unload the data.

Altering the subtype of a string column

If you add a column with a string data type, you can specify its subtype in the ALTER TABLE statement. Subtypes are valid for string columns of data types CHAR and VARCHAR. SBCS and MIXED are valid subtypes for CLOB data.

GUPI The interpretation of the FOREIGNKEY column depends on whether the MIXED DATA install option is YES or NO.

Entering an M in the column when the MIXED DATA install option is NO specifies SBCS data, not MIXED data.

To alter the subtype of an existing string column, enter the following SQL statement:

```
ALTER TABLE table-name ALTER COLUMN column-name
      SET DATA TYPE altered-data-type
```

GUPI

Related reference

"ALTER TABLE" (DB2 SQL Reference)

Altering the attributes of an identity column

You can change the attributes of an identity column by using the ALTER TABLE statement.

GUIP To change the attributes of an identity column, complete the following steps:

1. Issue an ALTER TABLE statement.
2. Specify the ALTER COLUMN option.

This clause changes all of the attributes of an identity column except the data type. However, if the ALTER TABLE statement is rolled back, a gap in the sequence of identity column values can occur because of unassigned cache values.

Changing the data type of an identity column, like changing some other data types, requires that you drop and then re-create the table. **GUIP**

Changing data types by dropping and re-creating the table

Some changes to a table cannot be made with the ALTER TABLE statement.

For example, you must make the following changes by redefining the column (that is, dropping the table and then re-creating the table with the new definitions):

- An original specification of CHAR (25) to CHAR (20)
- A column defined as INTEGER to SMALLINT
- A column defined as NOT NULL to allow null values
- The data type of an identity column

To make such changes, complete the following steps:

1. Unload the table.
2. Drop the table.

Attention: Be very careful about dropping a table. In most cases, recovering a dropped table is nearly impossible. If you decide to drop a table, remember that such changes might invalidate a plan or a package.

You must alter tables that have been created with RESTRICT ON DROP to remove the restriction before you can drop them.

3. Commit the changes.
4. Re-create the table.

GUIP If the table has an identity column:

- Choose carefully the new value for the START WITH attribute of the identity column in the CREATE TABLE statement if you want the first generated value for the identity column of the new table to resume the sequence after the last generated value for the table that was saved by the unload in step 1.
- Define the identity column as GENERATED BY DEFAULT so that the previously generated identity values can be reloaded into the new table.

GUIP

5. Reload the table.

Implications of dropping a table

Dropping a table has several implications that you should be aware of.

GUIP The DROP TABLE statement deletes a table. For example, to drop the project table, run the following statement:

```
DROP TABLE DSN8910.PROJ;
```

The statement deletes the row in the SYSIBM.SYSTABLES catalog table that contains information about DSN8910.PROJ. This statement also drops any other objects that depend on the project table. This action results in the following implications:

- The column names of the table are dropped from SYSIBM.SYSCOLUMNS.
- If the dropped table has an identity column, the sequence attributes of the identity column are removed from SYSIBM.SYSSEQUENCES.
- If triggers are defined on the table, they are dropped, and the corresponding rows are removed from SYSIBM.SYSTRIGGERS and SYSIBM.SYSPACKAGES.
- Any views based on the table are dropped.
- Application plans or packages that involve the use of the table are invalidated.
- Cached dynamic statements that involve the use of the table are removed from the cache.
- Synonyms for the table are dropped from SYSIBM.SYSSYNONYMS.
- Indexes created on any columns of the table are dropped.
- Referential constraints that involve the table are dropped. In this case, the project table is no longer a dependent of the department and employee tables, nor is it a parent of the project activity table.
- Authorization information that is kept in the DB2 catalog authorization tables is updated to reflect the dropping of the table. Users who were previously authorized to use the table, or views on it, no longer have those privileges, because catalog rows are deleted.
- Access path statistics and space statistics for the table are deleted from the catalog.
- The storage space of the dropped table might be reclaimed.
 - If the table space containing the table is implicitly created (using the CREATE TABLE statement without the TABLESPACE clause), the table space is also dropped. If the data sets are in a storage group, dropping the table space reclaims the space. For user-managed data sets, you must reclaim the space yourself.
 - If the table space containing the table is partitioned, or contains only the one table, you can drop the table space.
 - If the table space containing the table is segmented, DB2 reclaims the space.
 - If the table space containing the table is simple, and contains other tables, you must run the REORG utility to reclaim the space.
- If the table contains a LOB column, the auxiliary table and the index on the auxiliary table are dropped. The LOB table space is dropped if it was created with SQLRULES(STD).

If a table has a partitioning index, you must drop the table space or use LOAD REPLACE when loading the redefined table. If the CREATE TABLE that is used to redefine the table creates a table space implicitly, commit the DROP statement before re-creating a table by the same name. You must also commit the DROP statement before you create any new indexes with the same name as the original indexes.

Objects that depend on the dropped table

Before dropping a table, check to see what objects are dependent on it. For example, the DB2 catalog tables SYSIBM.SYSVIEWDEP, SYSIBM.SYSPLANDEP, and SYSIBM.SYSPACKDEP indicate what views, application plans, and packages are dependent on different DB2 objects.

Finding dependent views

GUIP The following example query lists the views, with their creators, that are affected if you drop the project table:

```
SELECT DNAME, DCREATOR
  FROM SYSIBM.SYSVIEWDEP
 WHERE BNAME = 'PROJ'
 AND BCREATOR = 'DSN8910'
 AND BTYPE = 'T';
```

GUIP

Finding dependent packages

GUIP The next example lists the packages, identified by the package name, collection ID, and consistency token (in hexadecimal representation), that are affected if you drop the project table:

```
SELECT DNAME, DCOLLID, HEX(DCONTOKEN)
  FROM SYSIBM.SYSPACKDEP
 WHERE BNAME = 'PROJ'
 AND BQUALIFIER = 'DSN8910'
 AND BTYPE = 'T';
```

GUIP

Finding dependent plans

GUIP The next example lists the plans, identified by plan name, that are affected if you drop the project table:

```
SELECT DNAME
  FROM SYSIBM.SYSPLANDEP
 WHERE BNAME = 'PROJ'
 AND BCREATOR = 'DSN8910'
 AND BTYPE = 'T';
```

GUIP

Finding other dependencies

In addition, the SYSIBM.SYSINDEXES table tells you what indexes currently exist on a table. From the SYSIBM.SYSTABAUTH table, you can determine which users are authorized to use the table.

Re-creating a table

You can re-create a DB2 table in order to decrease the length attribute of a string column or the precision of a numeric column.

GUIP To re-create a DB2 table to decrease the length attribute of a string column or the precision of a numeric column, complete the following steps:

1. If you do not have the original CREATE TABLE statement and all authorization statements for the table (call it T1), query the catalog to determine its description, the description of all indexes and views on it, and all users with privileges on it.
2. Create a new table (call it T2) with the desired attributes.
3. Copy the data from the old table T1 into the new table T2 by using one of the following methods:
 - a. Execute the following INSERT statement:

```
INSERT INTO T2
  SELECT * FROM T1;
```
 - b. Load data from your old table into the new table by using the INCURSOR option of the LOAD utility. This option uses the DB2 UDB family cross-loader function.
4. Execute the statement DROP TABLE T1. If T1 is the only table in an explicitly created table space, and you do not mind losing the compression dictionary, if one exists, drop the table space instead, so that the space is reclaimed.
5. Commit the DROP statement.
6. Use the statement RENAME TABLE to rename table T2 to T1.
7. Run the REORG utility on the table space that contains table T1.
8. Notify users to re-create any synonyms, indexes, views, and authorizations they had on T1. **GUIP**

If you want to change a data type from string to numeric or from numeric to string (for example, INTEGER to CHAR or CHAR to INTEGER), use the CHAR and DECIMAL scalar functions in the SELECT statement to do the conversion. Another alternative is to use the following method:

1. Use UNLOAD or REORG UNLOAD EXTERNAL (if the data to unload is less than 32 KB) to save the data in a sequential file, and then
2. Use the LOAD utility to repopulate the table after re-creating it. When you reload the table, make sure you edit the LOAD statement to match the new column definition.

This method is particularly appealing when you are trying to re-create a large table.

Moving a table to a table space of a different page size

You cannot alter a table to use a different page size. However, you can move a table to a table space of a different page size.

GUIP To move a table to a table space of a different page size:

1. Unload the table using UNLOAD FROM TABLE or REORG UNLOAD EXTERNAL FROM TABLE.
2. Use CREATE TABLE LIKE on the table to re-create it in the table space of the new page size.
3. Use DB2 Control Center, DB2 Administration Tool, or catalog queries to determine the dependent objects: views, authorization, plans, packages, synonyms, triggers, referential integrity, and indexes.
4. Drop the original table.

5. Rename the new table to the name of the old table using RENAME TABLE.
6. Re-create all dependent objects.
7. Rebind plans and packages.
8. Reload the table using data from the SYSREConn data set and the control statements from the SYSPUNCH data set, which was created when the table was unloaded. 

Altering DB2 views

In many cases, changing user requirements can be satisfied by modifying an existing view.

However, to change a view you must drop the view and create a new view with your modified specifications.

 Complete the following steps to drop and recreate a view:

1. Issue the DROP VIEW SQL statement.
2. Commit the drop. When you drop a view, DB2 also drops the dependent views.
3. Recreate the modified view using the CREATE VIEW SQL statement. 

Attention: When you drop a view, DB2 invalidates application plans and packages that are dependent on the view and revokes the privileges of users who are authorized to use it. DB2 attempts to rebind the package or plan the next time it is executed, and you receive an error if you do not recreate the view.

To tell how much rebinding and reauthorizing is needed if you drop a view, see the following table.

Table 29. Catalog tables to check after dropping a view

Catalog table	What to check
SYSIBM.SYSPLANDEP	Application plans dependent on the view
SYSIBM.SYSPACKDEP	Packages dependent on the view
SYSIBM.SYSVIEWDEP	Views dependent on the view
SYSIBM.SYSTABAUTH	Users authorized to use the view

Altering views by using the INSTEAD OF trigger

Only specific types of views can be updated, inserted, and deleted. The INSTEAD OF trigger provides an extension to the updatability of views.

 Unlike other forms of triggers that are defined only on tables, the INSTEAD OF trigger can only be defined on views.

If you use the INSTEAD OF trigger, the requested update operation against the view is replaced by the trigger logic, which then performs the operation on behalf of the view.

Use the CREATE TRIGGER statement to perform INSTEAD OF insert, update, and delete operations. 

Related reference

"CREATE TRIGGER" (DB2 SQL Reference)

Altering DB2 indexes

You can add a new column to an index or change most of the index clauses for an existing index with the ALTER INDEX statement, including BUFFERPOOL, CLOSE, COPY, PIECESIZE, PADDED or NOT PADDED, CLUSTER or NOT CLUSTER, and those clauses that are associated with storage space, free space, and group buffer pool caching.

With the ALTER INDEX statement, you can:

- Add a new column to an index.
- Alter the PADDED or NOT PADDED attribute to change how varying-length columns are stored in the index.
- Alter the CLUSTER or NOT CLUSTER attribute to change how data is stored.
- Alter the compression setting using ALTER COMPRESS YES or ALTER COMPRESS NO.
- Change the limit key for index-controlled partitioning to rebalance data among the partitions in a partitioned table space.

For other changes, you must drop and recreate the index.

When you add a new column to an index, change how varying-length columns are stored in the index, or change the data type of a column in the index, DB2 creates a new version of the index.

Restrictions:

- If the padding of an index is changed, the index is placed in RBDP (REBUILD-pending) and a new version of the index is not created.
- If an index is versioned, any alteration to use index compression places the index in REBUILD-pending status.
- If an index is compressed, altering a column that participates in the index key places all indexes that contain that column as part of the index key in REBUILD-pending status.
- You cannot add a column with the DESC attribute to an index if the column is a VARBINARY column or a column with a distinct type that is based on the VARBINARY type.

To change the description of an index at the current server, issue the ALTER INDEX statement.

The ALTER INDEX statement can be embedded in an application program or issued interactively.

Related concepts

"Padded and not padded indexes" (Introduction to DB2 for z/OS)

Related reference

"ALTER INDEX" (DB2 SQL Reference)

Adding a column to an index when you add the column to a table

When you use the ALTER INDEX statement to add a column to an existing index, the new column becomes the rightmost column of the index key.

Restriction: You cannot add a column to an index that enforces a primary key, unique key, or referential constraint. Also, you cannot add columns to IBM-defined indexes on the DB2 catalog.

GUPI To add a column to an existing index:

1. Issue the ALTER INDEX ADD COLUMN SQL statement when you add a column to a table.
2. Commit the alter procedure.

If the column that is being added to the index is already part of the table on which the index is defined, the index is left in a REBUILD-pending (RBDP) status. However, if you add a new column to a table and to an existing index on that table within the same unit of work, the index is left in advisory REORG-pending (AREO*) status and can be used immediately for data access.

If you add a column to an index and to a table within the same unit of work, this will cause table and index versioning.

For example, assume that you created a table with columns that include ACCTID, STATE, and POSTED:

```
CREATE TABLE TRANS
  (ACCTID ...,
   STATE ...,
   POSTED ...,
   ... , ...)
...;
```

You have an existing index on the STATE column:

```
CREATE INDEX STATE_IX ON TRANS(STATE);
```

To add a ZIPCODE column to the table and the index, issue the following statements:

```
ALTER TABLE TRANS ADD COLUMN ZIPCODE CHAR(5);
ALTER INDEX STATE_IX ADD COLUMN (ZIPCODE);
COMMIT;
```

Because the ALTER TABLE and ALTER INDEX statements are executed within the same unit of work, DB2 immediately can use the new index with the key STATE, ZIPCODE for data access.

GUPI

Related reference

"ALTER INDEX" (DB2 SQL Reference)

Altering how varying-length index columns are stored

You can use the ALTER INDEX statement to change how varying-length column values are stored in an index.

GUIP To alter how varying-length column values are stored in an index, complete the following steps:

1. Choose the padding attribute for the columns.
2. Issue the ALTER INDEX SQL statement.
 - Specify the NOT PADDED clause if you do not want column values to be padded to their maximum length. This clause specifies that VARCHAR and VARGRAPHIC columns of an existing index are stored as varying-length columns.
 - Specify the PADDED clause if you want column values to be padded to the maximum lengths of the columns. This clause specifies that VARCHAR and VARGRAPHIC columns of an existing index are stored as fixed-length columns.
3. Commit the alter procedure.

The ALTER INDEX statement is successful only if the index has at least one varying-length column.

When you alter the padding attribute of an index, the index is placed into a restricted REBUILD-pending (RBDP) state. When you alter the padding attribute of a nonpartitioned secondary index (NPSI), the index is placed into a page set REBUILD-pending (PSRBD) state. In both cases, the indexes cannot be accessed until they are rebuilt from the data. **GUIP**

Related concepts

"Padded and not padded indexes" (Introduction to DB2 for z/OS)

Altering the clustering of an index

You can use the ALTER INDEX SQL statement to change the clustering index for a table.

GUIP To change the clustering option of an index:

1. Issue the ALTER INDEX SQL statement.
2. Specify the desired clustering option.

Restriction: You can only specify CLUSTER if there is not already another clustering index.

- CLUSTER indicates that the index is to be used as the clustering index of the table. The change takes effect immediately. Any subsequently inserted rows use the new clustering index. Existing data remains clustered by the previous clustering index until the table space is reorganized.
 - NOT CLUSTER indicates that the index is not to be used as the clustering index of the table. However, if the index was previously defined as the clustering index, it continues to be used as the clustering index until you explicitly specify CLUSTER for a different index.
3. Commit the alter procedure.

Specifying NOT CLUSTER for an index that is not a clustering index is ignored.

GUIP

Related reference

"ALTER TABLE" (DB2 SQL Reference)

Dropping and redefining a DB2 index

Dropping an index does not cause DB2 to drop any other objects. The consequence of dropping indexes is that DB2 invalidates application plans and packages that use the index and automatically rebinds them when they are next used.

Prerequisite: Any primary key, unique key, or referential constraints associated with a unique index must be dropped before you drop the unique index. However, you can drop a unique index for a unique key without dropping the unique constraint if the unique key was created before Version 9.

Commit the drop before you create any new table spaces or indexes by the same name.

GUIP To drop and re-create an index, complete the following steps:

1. Issue a DROP INDEX SQL statement.
2. Commit the drop procedure. The index space associated with the index is also dropped.
3. Re-create the modified index by issuing a CREATE INDEX SQL statement.
4. Rebind any application programs that use the dropped index. **GUIP**

If you drop an index and then run an application program using that index (and thereby automatically rebound), that application program does not use the old index. If, at a later time, you re-create the index and the application program is not rebound, the application program cannot take advantage of the new index.

Reorganizing indexes

You want to reorganize indexes to correct any performance degradation that might occur after a schema change that affects an index.

Although data availability is maximized by the use of index versions, performance might suffer because DB2 does not automatically reformat the data in the index to conform to the most recent schema definition. DB2 defers any reformatting of existing data until you reorganize the index and apply the schema changes. The more ALTER statements (which affect indexes) that you commit between reorganizations, the more index versions DB2 must track, and the more performance can suffer.

To reorganize an index:

Run the REORG INDEX utility as soon as possible after a schema change that affects an index. You can also run the REORG TABLESPACE utility.

Related concepts

“Index versions” on page 63

Recycling index version numbers

To prevent DB2 from running out of index version numbers (and to prevent subsequent ALTER statements from failing), you must recycle unused index version numbers regularly.

DB2 can store up to 16 index versions, numbered sequentially from 0 to 15. The next consecutive version number after 15 is 1. Version number 0 is never reused, because it is reserved for the original version of the index. The versions that are

associated with schema changes that have not been applied yet are considered to be “in use,” and the range of used versions is stored in the catalog. In use versions can be recovered from image copies of the table space, if necessary.

Version numbers are considered to be unused if the schema changes that are associated with them have been applied and no image copies contain data at those versions.

GUIP To recycle unused index version numbers:

1. Determine the range of version numbers that are currently in use for an index by querying the `OLDEST_VERSION` and `CURRENT_VERSION` columns of the `SYSIBM.SYSINDEXES` catalog table.
2. Next, run the appropriate utility to recycle unused index version numbers.
 - For indexes that are defined as `COPY YES`, run the `MODIFY RECOVERY` utility.
If all reusable version numbers (1 to 15) are currently in use, reorganize the index by running `REORG INDEX` or `REORG TABLESPACE` before you recycle the version numbers.
 - For indexes that are defined as `COPY NO`, run the `REORG TABLESPACE`, `REORG INDEX`, `LOAD REPLACE`, or `REBUILD INDEX` utility. These utilities recycle the version numbers as they perform their primary functions. **GUIP**

Related concepts

“Index versions” on page 63

Altering stored procedures

Use the `ALTER PROCEDURE` statement to update the description of a stored procedure.

GUIP To alter an existing stored procedure, complete the following step:

Issue the `ALTER PROCEDURE` statement.

Example 1: The following example changes the stored procedure `SYSPROC.MYPROC` to run only in the WLM environment `PARTSEC`:

```
ALTER PROCEDURE SYSPROC.MYPROC
  WLM ENVIRONMENT PARTSEC;
```

Example 2: If `SYSPROC.MYPROC` is defined with `SECURITY DEFINER`, the external security environment for the stored procedure uses the authorization ID of the owner of the stored procedure. The following example changes the procedure to use the authorization ID of the person running it:

```
ALTER PROCEDURE SYSPROC.MYPROC
  SECURITY USER;
```

GUIP

Altering user-defined functions

The ALTER FUNCTION statement updates the description of user-defined functions.

GUIP To alter a user-defined function, complete the following step:

Issue the ALTER FUNCTION SQL statement.

Changes to the user-defined function take effect immediately.

In the following example, two functions named CENTER exist in the SMITH schema. The first function has two input parameters with INTEGER and FLOAT data types, respectively. The specific name for the first function is FOCUS1. The second function has three parameters with CHAR(25), DEC(5,2), and INTEGER data types.

Using the specific name to identify the function, change the WLM environment in which the first function runs from WLMENVNAME1 to WLMENVNAME2:

```
ALTER SPECIFIC FUNCTION SMITH.FOCUS1  
  WLM ENVIRONMENT WLMENVNAME2;
```

The following example changes the second function when any arguments are null:

```
ALTER FUNCTION SMITH.CENTER (CHAR(25), DEC(5,2), INTEGER)  
  RETURNS ON NULL CALL;
```

GUIP

Altering implicitly created XML objects

You can alter implicitly created objects, however there are several restrictions.

GUIP To alter implicitly created XML objects:

Determine the restrictions on the XML object that you want to alter.

Option	Description
XML table space	<p>You can alter the following properties:</p> <ul style="list-style-type: none"> • BUFFERPOOL (16 KB buffer pools only) • COMPRESS • PRIQTY • SECQTY • MAXROWS • FREEPAGE • PCTFREE • GBPCACHE • USING STOGROUP • ERASE • LOCKSIZE (The only possible values are XML and TABLESPACE.) <p>XML table space attributes that are inherited from the base table space, such as LOG, are implicitly altered if the base table space is altered.</p>
XML table	The ALTER TABLE ALTER PARTITION statement is not supported if the table contains an XML column.
Index	<p>You cannot alter the following properties:</p> <ul style="list-style-type: none"> • CLUSTER • PADDED • ADD COLUMN.



Changing the high-level qualifier for DB2 data sets

The high-level qualifier for DB2 data sets is the catalog name of the integrated catalog facility, which is commonly called the *user catalog*.

You cannot change this qualifier for DB2 data sets using the DB2 installation or migration update process. You must use other methods to change this qualifier for both system data sets and user data sets.

The following procedures do not actually move or copy data.

Changing the high-level qualifier for DB2 data sets is a complex task; so, you should have experience both with DB2 and with managing user catalogs.

Prerequisite: To concentrate on DB2-related issues, this procedure assumes that the catalog alias resides in the *same* user catalog as the one that is currently used. If the new catalog alias resides in a *different* user catalog, see *DFSMS/MVS™: Access Method Services for the Integrated Catalog* for information about planning such a move.

If the data sets are managed by the Storage Management Subsystem (SMS), make sure that automatic class selection routines are in place for the new data set name.

Defining a new integrated catalog alias

You can complete this task at any time before changing the high-level qualifier for system or user data sets.

To define the new high-level qualifier as an alias to a current integrated catalog, complete the following step:

Issue the following access method services command:

```
DEFINE ALIAS (NAME (newcat) RELATE (usercat) CATALOG (master-cat))
```

See *DFSMS/MVS: Access Method Services for the Integrated Catalog* for more information.

Changing the qualifier for system data sets

In this task, you stop DB2, change the high-level qualifier in the system parameter load module (possibly DSNZPARM), and establish a new xxxxMSTR cataloged procedure before restarting DB2.

Important: The following steps must be done in sequence.

Changing the load module to reflect the new qualifier

To change the system parameter load module to specify the new qualifier for new archive data sets and the DB2 catalog and directory data sets, you must use the installation process.

To specify the new qualifier:

1. Run the installation CLIST, and specify INSTALL TYPE=INSTALL and DATA SHARING FUNCTION=NONE.
2. Enter new values for the fields shown in the following table.

Table 30. CLIST panels and fields to change to reflect new qualifier

Panel name	Field name	Comments
DSNTIPA1	INSTALL TYPE	Specify INSTALL. Do <i>not</i> specify a new default prefix for the input data sets listed on this panel.
DSNTIPA1	OUTPUT MEMBER NAME	
DSNTIPA2	CATALOG ALIAS	
DSNTIPH	COPY 1 NAME and COPY 2 NAME	These are the bootstrap data set names.
DSNTIPH	COPY 1 PREFIX and COPY 2 PREFIX	These fields appear for both active and archive log prefixes.
DSNTIPT	SAMPLE LIBRARY	This field allows you to specify a field name for edited output of the installation CLIST. Avoid overlaying existing data sets by changing the middle node, NEW, to something else. The only members you use in this procedure are xxxxMSTR and DSNTIJUZ in the sample library.
DSNTIPO	PARAMETER MODULE	Change this value only if you want to preserve the existing member through the CLIST.

The output from the CLIST is a new set of tailored JCL with new cataloged procedures and a DSNTIJUZ job, which produces a new member.

3. Run the first two job steps of DSNTIJUZ to update the subsystem parameter load module.

Unless you have specified a new name for the load module, make sure the output load module does not go to the SDSNEXIT or SDSNLOAD library used by the active DB2 subsystem.

If you are changing the subsystem ID in addition to the system data set name qualifier, you should also run job steps DSNITIZP and DSNITIZQ to update the DSNHDECP module (ZPARM parameter SSID). Make sure that the updated DSNHDECP module does not go to the SDSNEXIT or SDSNLOAD library used by the active DB2 subsystem. Use caution when changing the subsystem ID.

For more information, see the heading "MVS PARMLIB updates panel: DSNITIPM" for the discussion of panel DSNITIPM for PARMLIB members where the subsystem ID has to be changed.

Stopping DB2 when no activity is outstanding

In this step, make sure the subsystem does not have any outstanding activity, such as outstanding units of recovery or pending writes. This ensures that DB2 does not need to access the data sets on restart through the log, which contains the *old* data set qualifiers.

To stop DB2 when no activity is outstanding:

1. Stop DB2 by entering the following command:

```
-STOP DB2 MODE(QUIESCE)
```

This command allows DB2 to complete processing currently executing programs.

2. Start DB2 by entering the following command:

```
-START DB2 ACCESS(MAINT)
```

3. Use the following commands to make sure the subsystem is in a consistent state.

```
-DISPLAY THREAD(*) TYPE(*)  
-DISPLAY UTILITY (*)  
-TERM UTILITY(*)  
-DISPLAY DATABASE(*) RESTRICT  
-DISPLAY DATABASE(*) SPACENAM(*) RESTRICT  
-RECOVER INDOUBT
```

Correct any problems before continuing.

4. Stop DB2 by entering the following command:

```
-STOP DB2 MODE(QUIESCE)
```

5. Run the print log map utility (DSNJU004) to identify the current active log data set and the last checkpoint RBA.

6. Run DSN1LOGP with the SUMMARY (YES) option, using the last checkpoint RBA from the output of the print log map utility you ran in the previous step.

The report headed DSN1157I RESTART SUMMARY identifies active units of recovery or pending writes. If either situation exists, do not attempt to continue. Start DB2 with ACCESS(MAINT), use the necessary commands to correct the problem, and repeat steps 4 through 6 until all activity is complete.

Renaming system data sets with the new qualifier

All of the following steps assume that the new qualifier and the old qualifier reside in the same user catalog.

Prerequisite: Access method services does not allow ALTER where the new name does not match the existing catalog structure for an SMS-managed VSAM data set. If the data set is not managed by SMS, the rename succeeds, but DB2 cannot allocate it as described in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

DB2 table spaces are defined as linear data sets with DSNDBC as the second node of the name for the cluster and DSNDBD for the data component. The examples shown here assume the normal defaults for DB2 and VSAM data set names. Use access method services statements with a generic name (*) to simplify the process. Access method services allows only one generic name per data set name string.

To rename the system data sets:

1. Using IDCAMS, change the names of the catalog and directory table spaces. Be sure to specify the instance qualifier of your data set, *y*, which can be either I or J. For example,

```
ALTER oldcat.DSNDBC.DSNDB01.*.y0001.A001 -
    NEWNAME (newcat.DSNDBC.DSNDB01.*.y0001.A001)
ALTER oldcat.DSNDBD.DSNDB01.*.y0001.A001 -
    NEWNAME (newcat.DSNDBD.DSNDB01.*.y0001.A001)
ALTER oldcat.DSNDBC.DSNDB06.*.y0001.A001 -
    NEWNAME (newcat.DSNDBC.DSNDB06.*.y0001.A001)
ALTER oldcat.DSNDBD.DSNDB06.*.y0001.A001 -
    NEWNAME (newcat.DSNDBD.DSNDB06.*.y0001.A001)
```

2. Using IDCAMS, change the active log names. Active log data sets are named *oldcat.LOGCOPY1.COPY01* for the cluster component and *oldcat.LOGCOPY1.COPY01.DATA* for the data component. For example,

```
ALTER oldcat.LOGCOPY1.* -
    NEWNAME (newcat.LOGCOPY1.*)
ALTER oldcat.LOGCOPY1.*.DATA -
    NEWNAME (newcat.LOGCOPY1.*.DATA)
ALTER oldcat.LOGCOPY2.* -
    NEWNAME (newcat.LOGCOPY2.*)
ALTER oldcat.LOGCOPY2.*.DATA -
    NEWNAME (newcat.LOGCOPY2.*.DATA)
```

3. Using IDCAMS, change the BSDS names. For example,

```
ALTER oldcat.BSDS01 -
    NEWNAME (newcat.BSDS01)
ALTER oldcat.BSDS01.* -
    NEWNAME (newcat.BSDS01.*)
ALTER oldcat.BSDS02 -
    NEWNAME (newcat.BSDS02)
ALTER oldcat.BSDS02.* -
    NEWNAME (newcat.BSDS02.*)
```

Updating the BSDS with the new qualifier

Update the first BSDS with the new alias and correct data set names for the active logs. This procedure does not attempt to change the names of existing archive log data sets.

If these catalog entries or data sets will not be available in the future, copy all the table spaces in the DB2 subsystem to establish a new recovery point. You can optionally delete the entries from the BSDS. If you do not delete the entries, they will gradually be replaced by newer entries.

To update the BSDS:

1. Run the change log inventory utility (DSNJU003).

Use the new qualifier for the BSDS because it has now been renamed. The following example illustrates the control statements required for three logs and dual copy is specified for the logs. This is only an example; the number of logs can vary and dual copy is an option. The starting and ending log RBAs are from the print log map report.

```

NEWCAT VSAMCAT=newcat
DELETE DSNNAME=oldcat.LOGCOPY1.DS01
DELETE DSNNAME=oldcat.LOGCOPY1.DS02
DELETE DSNNAME=oldcat.LOGCOPY1.DS03
DELETE DSNNAME=oldcat.LOGCOPY2.DS01
DELETE DSNNAME=oldcat.LOGCOPY2.DS02
DELETE DSNNAME=oldcat.LOGCOPY2.DS03
NEWLOG DSNNAME=newcat.LOGCOPY1.DS01,COPY1,STARTRBA=strtrba,ENDRBA=endrba
NEWLOG DSNNAME=newcat.LOGCOPY1.DS02,COPY1,STARTRBA=strtrba,ENDRBA=endrba
NEWLOG DSNNAME=newcat.LOGCOPY1.DS03,COPY1,STARTRBA=strtrba,ENDRBA=endrba
NEWLOG DSNNAME=newcat.LOGCOPY2.DS01,COPY2,STARTRBA=strtrba,ENDRBA=endrba
NEWLOG DSNNAME=newcat.LOGCOPY2.DS02,COPY2,STARTRBA=strtrba,ENDRBA=endrba
NEWLOG DSNNAME=newcat.LOGCOPY2.DS03,COPY2,STARTRBA=strtrba,ENDRBA=endrba

```

During startup, DB2 compares the *newcat* value with the value in the system parameter load module, and they must be the same.

2. Using the IDCAMS REPRO command, replace the contents of BSDS2 with the contents of BSDS01.
3. Run the print log map utility (DSNJU004) to verify your changes to the BSDS.
4. At a convenient time, change the DD statements for the BSDS in any of your offline utilities to use the new qualifier.

Establishing a new xxxxMSTR cataloged procedure

Before you start DB2, establish a new xxxxMSTR cataloged procedure.

1. Update xxxxMSTR in SYS1.PROCLIB with the new BSDS data set names.
2. Copy the new system parameter load module to the active SDSNEXIT/SDSNLOAD library.

Starting DB2 with the new xxxxMSTR and load module

Now you can start DB2 with the new xxxxMSTR and load module.

1. Issue a START DB2 command with the new module name as shown in the following example.

```
-START DB2 PARM(new_name)
```
2. Optional: If you stopped DSNDB01 or DSNDB06 in “Stopping DB2 when no activity is outstanding” on page 133, you must explicitly start them in this step.

Changing qualifiers for other databases and user data sets

This task changes qualifiers for DB2 databases other than the catalog and directory.

DSNDB07 is a system database but contains no permanent data, and can be deleted and redefined with the new qualifier. If you are changing its qualifier, do that before the rest of the user databases.

You can change the databases in the following list that apply to your environment:

- DSNDB07 (work file database)
- DSNDB04 (system default database)
- DSNDDF (communications database)
- DSNRLST (resource limit facility database)
- DSNRGFDB (the database for data definition control)
- Any other application databases that use the old high-level qualifier

At this point, the DB2 catalog tables SYSSTOGROUP, SYSTABLEPART, and SYSINDEXPART contain information about the old integrated user catalog alias. To update those tables with the new alias, you must use the following procedures. Until you do so, the underlying resources are not available.

Important: Table spaces and indexes that span more than one data set require special procedures. Partitioned table spaces can have different partitions allocated to different DB2 storage groups. Nonpartitioned table spaces or indexes only have the additional data sets to rename (those with the lowest level name of A002, A003, and so on).

Changing your work database to use the new high-level qualifier

You can use one of two methods to change the high-level qualifier for your work database or possibly DSNDB07. Which method you use depends on if you have a new installation or a migrated installation.

New installation

To change your work database, complete the following steps:

1. Reallocate the database using the installation job DSNTIJTM from *prefix.SDSNSAMP*
2. Modify your existing job. Change the job to remove the BIND step for DSNTIAD and rename the data set names in the DSNTTMP step to your new names, making sure you include your current allocations.

Migrated installation

Migrated installations do not have a usable DSNTIJTM, because the IDCAMS allocation step is missing.

To change your work database, complete the following steps:

1. Stop the database, using the following command (for a database named DSNDB07). For example,
`-STOP DATABASE (DSNDB07)`
2. Drop the database, using the following SQL statement. For example,
`DROP DATABASE DSNDB07;`
3. Re-create the database, using the following SQL statement. For example,
`CREATE DATABASE DSNDB07;`
4. Define the clusters, using the following access method services commands. Be sure to specify the instance qualifier of your data set, *y*, which can be either I or J. For example,

```
ALTER oldcat.DSNDBC.DSNDB07.DSN4K01.y0001.A001
  NEWNAME newcat.DSNDBC.DSNDB07.DSN4K01.y0001.A001
ALTER oldcat.DSNDBC.DSNDB07.DSN32K01.y0001.A001
  NEWNAME newcat.DSNDBC.DSNDB07.DSN32K01.y0001.A001
```

Repeat the preceding statements (with the appropriate table space name) for as many table spaces as you use.

5. Create the table spaces in DSNDB07. For example,
`CREATE TABLESPACE DSN4K01
 IN DSNDB07
 BUFFERPOOL BP0
 CLOSE NO
 USING VCAT DSN910;`

```
CREATE TABLESPACE DSN32K01
  IN DSNDB07
  BUFFERPOOL BP32K
  CLOSE NO
  USING VCAT DSNC910;
```

6. Start the database, using the following command:
-START DATABASE (DSNDB07)

Changing user-managed objects to use the new qualifier

To change user-managed objects, complete the following tasks:

1. Stop the table spaces and index spaces, using the following command:
-STOP DATABASE(*dbname*) SPACENAM(*)
2. Use the following SQL ALTER TABLESPACE and ALTER INDEX statements with the USING clause to specify the new qualifier:

```
ALTER TABLESPACE dbname.tsname
  USING VCAT newcat;
```

```
ALTER INDEX creator.index-name
  USING VCAT newcat;
```

Repeat this step for all the objects in the database.

3. Using IDCAMS, rename the data sets to the new qualifier. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:

```
ALTER oldcat.DSNDBC.dbname.*.y0001.A001 -
  NEWNAME newcat.DSNDBC.dbname.*.y0001.A001
ALTER oldcat.DSNDBD.dbname.*.y0001.A001 -
  NEWNAME newcat.DSNDBD.dbname.*.y0001.A001
```

4. Start the table spaces and index spaces, using the following command:
-START DATABASE(*dbname*) SPACENAM(*)
5. Verify the success of the procedure by entering the following command:
-DISPLAY DATABASE(*dbname*)
6. Using SQL, verify that you can access the data.

Renaming the data sets can be done while DB2 is down. They are included here because the names must be generated for each database, table space, and index space that is to change.

Changing DB2-managed objects to use the new qualifier

Use the following procedure when you want to keep the existing DB2 storage group, changing only the high-level qualifier.

To change DB2- managed objects, complete the following steps:

1. Remove all table spaces and index spaces from the storage group by converting the data sets temporarily to user-managed data sets.
 - a. Stop each database that has data sets you are going to convert, using the following command:
-STOP DATABASE(*dbname*) SPACENAM(*)

Restriction: Some databases must be explicitly stopped to allow any alterations. For these databases, use the following command:

```
-STOP DATABASE(dbname)
```

- b. Convert to user-managed data sets with the USING VCAT clause of the SQL ALTER TABLESPACE and ALTER INDEX statements, as shown in the following statements. Use the new catalog name for VCAT.

```
ALTER TABLESPACE dbname.tsname
  USING VCAT newcat;
```

```
ALTER INDEX creator.index-name
  USING VCAT newcat;
```

2. Drop the storage group, using the following statement:

```
DROP STOGROUP stogroup-name;
```

The DROP succeeds only if all the objects that referenced this STOGROUP are dropped or converted to user-managed (USING VCAT clause).

3. Re-create the storage group using the correct volumes and the new alias, using the following statement:

```
CREATE STOGROUP stogroup-name
  VOLUMES (VOL1,VOL2)
  VCAT newcat;
```

4. Using IDCAMS, rename the data sets for the index spaces and table spaces to use the new high-level qualifier. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J:

```
ALTER oldcat.DSNDBC.dbname.*.y0001.A001 -
  NEWNAME newcat.DSNDBC.dbname.*.y0001.A001
ALTER oldcat.DSNDBD.dbname.*.y0001.A001 -
  NEWNAME newcat.DSNDBD.dbname.*.y0001.A001
```

If your table space or index space spans more than one data set, be sure to rename those data sets also.

5. Convert the data sets back to DB2-managed data sets by using the new DB2 storage group. Use the following SQL ALTER TABLESPACE and ALTER INDEX statements:

```
ALTER TABLESPACE dbname.tsname
  USING STOGROUP stogroup-name
  PRIQTY priqty
  SECQTY secqty;
```

```
ALTER INDEX creator.index-name
  USING STOGROUP stogroup-name
  PRIQTY priqty
  SECQTY secqty;
```

If you specify USING STOGROUP without specifying the PRIQTY and SECQTY clauses, DB2 uses the default values.

6. Start each database, using the following command:
-START DATABASE(*dbname*) SPACENAM(*)
7. Verify the success of the procedure by entering the following command:
-DISPLAY DATABASE(*dbname*)
8. Using SQL, verify that you can access the data.

Tools for moving DB2 data

Moving DB2 data can be complicated. Fortunately, several tools exist that can help to simplify the process.

Important: Before copying any DB2 data, resolve any data that is in an inconsistent state. Use the DISPLAY DATABASE command to determine whether any inconsistent state exists, and the RECOVER INDOUBT command or the RECOVER utility to resolve the inconsistency. The copying process generally loses all traces of an inconsistency except the problems that result.

Although DB2 data sets are created using VSAM access method services, they are specially formatted for DB2 and cannot be processed by services that use VSAM

record processing. They can be processed by VSAM utilities that use control-interval (CI) processing and, if they are linear data sets (LDSs), also by utilities that recognize the LDS type.

Furthermore, copying the data might not be enough. Some operations require copying DB2 object definitions. And when copying from one subsystem to another, you must consider internal values that appear in the DB2 catalog and the log, for example, the DB2 object identifiers (OBIDs) and log relative byte addresses (RBAs).

The following tools can help to simplify the operations:

- The REORG and LOAD utilities move data sets from one disk device type to another within the same DB2 subsystem.
The INCURSOR option of the LOAD utility allows you to specify a cursor to select data from another DB2 table or tables, which can be on a remote DB2 system. Use the EXEC SQL utility control statement to declare the cursor before executing the LOAD utility. This option uses the DB2 UDB family cross-loader function.
- The COPY and RECOVER utilities allow you to recover an image copy of a DB2 table space or index space onto another disk device within the same subsystem.
- The UNLOAD or REORG UNLOAD EXTERNAL utility unloads a DB2 table into a sequential file and generates statements to allow the LOAD utility to load it elsewhere.
- The DSN1COPY utility copies the data set for a table space or index space to another data set. It can also translate the object identifiers and reset the log RBAs in the target data set. When you use the OBIDXLAT option of DSN1COPY to move objects from one system to another, use REPAIR VERSIONS to update the version information in the catalog and directory for the target table space or index.

You might also want to use the following tools to move DB2 data:

- The DB2 DataPropagator is a licensed program that can extract data from DB2 tables, DL/I databases, VSAM files, and sequential files.
- DFSMS, which contains the following functional components:
 - Data Set Services (DFSMSdss)
Use DFSMSdss to copy data between disk devices. For instructions, see *Data Facility Data Set Services: User's Guide and Reference*. You can use online panels to control this, through the Interactive Storage Management Facility (ISMF) that is available with DFSMS; for instructions, refer to *z/OS DFSMSdftp Storage Administration Reference*.
 - Data Facility Product (DFSMSdftp™)
This is a prerequisite for DB2. You can use access method services EXPORT and IMPORT commands with DB2 data sets when control interval processing (CIMODE) is used. For instructions on EXPORT and IMPORT, see *DFSMS/MVS: Access Method Services for the Integrated Catalog*.
 - Hierarchical Storage Manager (DFSMSHsm)
With the MIGRATE, HMIGRATE, or HRECALL commands, which can specify specific data set names, you can move data sets from one disk device type to another within the same DB2 subsystem. Do not migrate the DB2 directory, DB2 catalog, and the work file database (DSNDB07). Do not migrate any data sets that are in use frequently, such as the bootstrap data set and the active log. With the MIGRATE VOLUME command, you can move an entire disk volume from one device type to another. The program can be controlled using

online panels, through the Interactive Storage Management Facility (ISMF). For instructions, see *z/OS DFSMSHsm Managing Your Own Data*.

The following table shows which tools are applicable to specific operations.

Table 31. Tools applicable to data-moving operations

Tool	Moving a data set	Copying a database	Copying an entire subsystem
REORG and LOAD	Yes	Yes	No
UNLOAD	Yes	No	No
COPY and RECOVER	Yes	No	No
DSNTIAUL	Yes	Yes	No
DSN1COPY	Yes	Yes	No
DataRefresher or DXT™	Yes	Yes	No
DFSMSdss	Yes	No	Yes
DFSMSdfp	Yes	No	Yes
DFSMSHsm	Yes	No	No

Some of the listed tools rebuild the table space and index space data sets, and they therefore generally require longer to execute than the tools that merely copy them. The tools that rebuild are REORG and LOAD, RECOVER and REBUILD, DSNTIAUL, and DataRefresher. The tools that merely copy data sets are DSN1COPY, DFSMSdss, DFSMSdfp EXPORT and IMPORT, and DFSMSHsm.

DSN1COPY is fairly efficient in use, but somewhat complex to set up. It requires a separate job step to allocate the target data sets, one job step for each data set to copy the data, and a step to delete or rename the source data sets. DFSMSdss, DFSMSdfp, and DFSMSHsm all simplify the job setup significantly.

Although less efficient in execution, RECOVER is easy to set up if image copies and recover jobs already exist. You might only need to redefine the data sets involved and recover the objects as usual.

Moving DB2 data

DB2 provides several tools and options to make moving data easier.

You can move data within DB2 in several ways: copying a database, copying a DB2 subsystem, or by moving data sets within a particular DB2 subsystem.

Copying a relational database

Copying your relational database involves not only copying data, but also finding or generating, and executing, SQL statements to create storage groups, databases, table spaces, tables, indexes, views, synonyms, and aliases.

You can copy a database by using the DSN1COPY utility. As with the other operations, DSN1COPY is likely to execute faster than the other applicable tools. It copies directly from one data set to another, while the other tools extract input for LOAD, which then loads table spaces and builds indexes. But again, DSN1COPY is more difficult to set up. In particular, you must know the internal DB2 object identifiers, which other tools translate automatically.

Copying an entire DB2 subsystem

Copying a DB2 subsystem from one z/OS system to another involves the following:

- All the user data and object definitions
- The DB2 system data sets:
 - The log
 - The bootstrap data set
 - Image copy data sets
 - The DB2 catalog
 - The integrated catalog that records all the DB2 data sets

Although you can have two DB2 subsystems on the same z/OS system, one cannot be a copy of the other.

Only two of the tools listed are applicable: DFSMSdss DUMP and RESTORE, and DFSMSdftp EXPORT and IMPORT.

The tasks and tools associated with moving data within your DB2 subsystem include:

- “Tools for moving DB2 data” on page 138
- “Moving a DB2 data set”

Related information

 DFSMSdss and DFSMSdftp programs

Moving a DB2 data set

Moving DB2 data is accomplished by RECOVER, REORG, or DSN1COPY, or by the use of non-DB2 facilities, such as DFSMSdss.

Both the DB2 utilities and the non-DB2 tools can be used while DB2 is running, but the space to be moved should be stopped to prevent users from accessing it.

If you use storage groups, then you can change the storage group definition to include the new volumes.

The following procedures differ mainly in that the first procedure assumes that you do not want to reorganize or recover the data. Generally, this means that the first procedure is faster. In all cases, make sure that there is enough space on the target volume to accommodate the data set.

Choose between the following methods for moving data sets:

- “Moving data without REORG or RECOVER”
- “Moving DB2-managed data with REORG, RECOVER, or REBUILD” on page 142

Moving data without REORG or RECOVER

You can move data that you do not want to reorganize or recover.

To move data without using the REORG or RECOVER utilities, complete the following steps:

1. Stop the database by issuing a STOP DATABASE command.
-STOP DATABASE(*dbname*) SPACENAM(*)
2. Move the data, using DSN1COPY or a non-DB2 facility.

3. Issue the ALTER INDEX or ALTER TABLESPACE statement to use the new integrated catalog facility catalog name or DB2 storage group name.
4. Start the database by issuing a START DATABASE command.
-START DATABASE(*dbname*) SPACENAM(*)

Moving DB2-managed data with REORG, RECOVER, or REBUILD

The following procedure shows you how to create a storage group (possibly using a new catalog alias) and move the data to that new storage group.

1. Create a new storage group using the correct volumes and the new alias.
CREATE STOGROUP *stogroup-name*
VOLUMES (VOL1,VOL2)
VCAT (*newcat*);
2. Prevent access to the data sets you are going to move.
-STOP DATABASE(*dbname*) SPACENAM(*)
3. Enter the ALTER TABLESPACE and ALTER INDEX SQL statements to use the new storage group name.
ALTER TABLESPACE *dbname.tsname*
USING STOGROUP *stogroup-name*;
ALTER INDEX *creator.index-name*
USING STOGROUP *stogroup-name*;
4. Using IDCAMS, rename the data sets for the index spaces and table spaces to use the new high-level qualifier. Also, be sure to specify the instance qualifier of your data set, *y*, which can be either I or J. If you have run REORG with SHRLEVEL CHANGE or SHRLEVEL REFERENCE on any table spaces or index spaces, the fifth-level qualifier might be J0001.
ALTER *oldcat*.DSNDBC.*dbname*.*.y0001.A001 -
NEWNAME *newcat*.DSNDBC.*dbname*.*.y0001.A001
ALTER *oldcat*.DSNDBD.*dbname*.*.y0001.A001 -
NEWNAME *newcat*.DSNDBD.*dbname*.*.y0001.A001
5. Start the database for utility processing only.
-START DATABASE(*dbname*) SPACENAM(*) ACCESS(UT)
6. Use the REORG or the RECOVER utility on the table space or index space, or use the REBUILD utility on the index space.
7. Start the database.
-START DATABASE(*dbname*) SPACENAM(*)

Scenario: Moving from index-controlled to table-controlled partitioning

The following scenario describes how you can change an existing index-controlled partitioned table space to a table-controlled partitioned table space and implement a DPSI.

- GUIP** Assume that you have a very large transaction table named TRANS that contains one row for each transaction. The table includes the following columns:
- ACCTID, which is the customer account ID
 - POSTED, which holds the date of the transaction

The table space that contains TRANS is divided into 13 partitions, each of which contains one month of data. Two existing indexes are defined as follows:

- A partitioning index is defined on the transaction date by the following CREATE INDEX statement with a PARTITION ENDING AT clause:

```
CREATE INDEX IX1 ON TRANS(POSTED)
  CLUSTER
  (PARTITION 1 ENDING AT ('01/31/2002'),
   PARTITION 2 ENDING AT ('02/28/2002'),
   ...
   PARTITION 13 ENDING AT ('01/31/2003'));
```

The partitioning index is the clustering index, and the data rows in the table are in order by the transaction date. The partitioning index controls the partitioning of the data in the table space.

- A nonpartitioning index is defined on the customer account ID:

```
CREATE INDEX IX2 ON TRANS(ACCTID);
```

DB2 usually accesses the transaction table through the customer account ID by using the nonpartitioning index IX2. The partitioning index IX1 is not used for data access and is wasting space. In addition, you have a critical requirement for availability on the table, and you want to be able to run an online REORG job at the partition level with minimal disruption to data availability.

To save space and to facilitate reorganization of the table space, you can drop the partitioning index IX1, and you can replace the access index IX2 with a partitioned clustering index that matches the 13 data partitions in the table.

Issue the following statements:

```
DROP INDEX IX1;
CREATE INDEX IX3 ON TRANS(ACCTID)
  PARTITIONED CLUSTER;
COMMIT;

DROP INDEX IX2;
COMMIT;
```



What happens:

- When you drop the partitioning index IX1, DB2 converts the table space from index-controlled partitioning to table-controlled partitioning. DB2 changes the high limit key value that was originally specified to the highest value for the key column.
- When you create the index IX3, DB2 creates a partitioned index with 13 partitions that match the 13 data partitions in the table. Each index partition contains the account numbers for the transactions during that month, and those account numbers are ordered within each partition. For example, partition 11 of the index matches the table partition that contains the transactions for November, 2002, and it contains the ordered account numbers of those transactions.
- You drop the nonpartitioning index IX2 because it has been replaced by IX3.

You can now run an online REORG at the partition level with minimal impact on availability. For example:

```
REORG TABLESPACE dbname.tsname PART 11
  SHRLEVEL CHANGE
```

Running this utility reorganizes the data for partition 11 of *dbname.tsname*. The data rows are ordered within each partition to match the ordering of the clustering index.

Recommendations:

- Drop a partitioning index if it is used only to define partitions. When you drop a partitioning index, DB2 automatically converts the associated index-controlled partitioned table space to a table-controlled partitioned table space.
- You can create a data-partitioned secondary index (DPSI) as the clustering index so that the data rows are ordered within each partition of the table space to match the ordering of the keys of the DPSI.
- **GUIP** Create any new tables in a partitioned table space by using the PARTITION BY clause and the PARTITION ENDING AT clause in the CREATE TABLE statement to specify the partitioning key and the limit key values. **GUIP**

Part 2. Security and auditing

Chapter 4. Getting started with DB2 security

DB2 *security* refers to the protection of sensitive data, operation systems, and other resources that are critical to your business by controlling access to DB2 objects, subsystems, and other assets.

DB2 security is set through a security plan, implemented through privilege and authority management, and reinforced through the audit of accesses to protected data. A *security plan* defines the security objectives of your organization and specifies the policies and techniques that you use to meet these objectives. A *security audit* traces all data access and determines whether your security plan works as designed and implemented.

If you are new to DB2 security, skim through the succeeding topics for a brief overview of the techniques that you can use to manage access to your DB2 and protect your data before reading the scenario.

DB2 security solutions

With each new release, DB2 gets bigger, faster, and more secure.

Over the years, DB2 recognizes and addresses the following security problems:

- Privilege theft or mismanagement
- Application or application server tampering
- Data or log tampering
- Storage media theft
- Unauthorized access to objects

DB2 offers the following security solutions to address the problems:

- Authentication
- Authorization
- Data integrity
- Confidentiality
- System integrity
- Audit

What's new in DB2 Version 9.1 security?

DB2 Version 9.1 provides critical enhancements to security and auditing.

Trusted contexts

The following enhancements strengthen DB2 security in the z/OS environment.

A *trusted context* is a database entity based on a system authorization ID and a set of connection trust attributes. You can create and use a trusted context to establish a trusted connection between DB2 and an external entity, such as a middleware server. When a trusted connection is established, you can reuse the authorization, switch users of the connection, and manage objects by other users without the database server needing to authenticate the IDs; these authorization IDs already

acquire the necessary database privileges that are associated with the specific trusted context.

Roles

A *role* is a database entity, available only in a trusted context, that groups together one or more privileges. A role can own database objects which helps eliminate the need for individual users to own and control database objects.

You can assign a role to an individual user or a group of users by defining a trusted context. A role thus offers a mechanism other than authorization IDs through which you can assign privileges and authorities. As a result, it provides the flexibility of authorization methods and helps simplify the management of authentication.

Improved auditing

The flexibility of granting dynamic SQL privileges to roles provides improved auditing controls. Other improvements include new filtering keywords, such as the `ROLE` keyword, to provide "exclude" trace-filtering capabilities. In addition, application servers can provide the non-RACF user IDs to be included in both DB2 accounting and RACF audit ICTX records.

Secure Socket Layer support

DB2 exploits the z/OS Application Transparent - Transport Layer Security (AT-TLS) function in the TCP/IP stack and provides TLS for DB2 clients that require secure connections. AT-TLS performs TLS on behalf of the application by invoking the z/OS system Secure Socket Layer (SSL) in the TCP transport layer of the stack.

The DB2 SSL support provides protected connections between DB2 servers. With SSL support, a DB2 server can optionally listen on a secondary secure port for inbound SSL connections. Similarly, a DB2 requester can optionally send encrypted data across the network through an SSL connection to the server.

Protection against denial-of-service attacks

In a denial-of-service attack, an attacker attempts to prevent legitimate users from accessing information or services. By targeting a DB2 server and its network connection, an attacker might be able to prevent you from accessing data or other services that the server provides. If this situation occurs, the DB2 server guards against the attacks and provide a more secure operating environment for legitimate users.

Related concepts

"Trusted contexts" on page 301

"Roles in a trusted context" on page 166

"AT-TLS configuration" on page 331

Related tasks

"Managing denial-of-service attacks" on page 286

DB2 data access control

You can enable or disable data access control within DB2.

Access to data can originate from a user through an interactive terminal session, an application program that is running in batch mode, or an IMS or CICS transaction. Given the variety of access originators, the term *process* is used to represent all access to data. For example, within a DB2 subsystem, a process can be a primary authorization ID, one or more secondary IDs, a role, or an SQL ID.

A process can gain access to DB2 data through several routines. As shown in the following diagram, DB2 provides different ways for you to control access from all but the data set protection route.

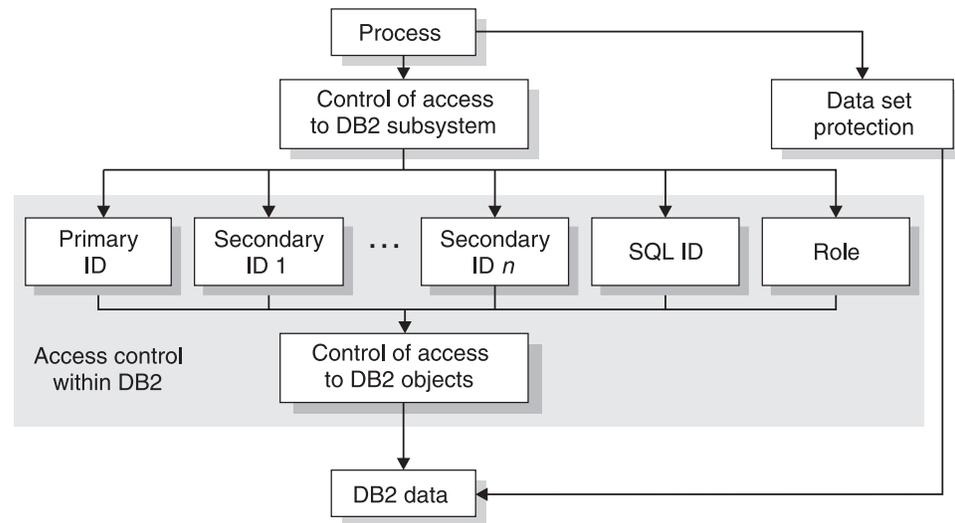


Figure 14. DB2 data access control

One of the ways that DB2 controls access to data is by using authorization IDs or roles. DB2 relies on IDs or roles to determine whether to allow or prohibit certain processes. DB2 assigns privileges and authorities to IDs or roles so that the owning users can take actions on objects. In this sense, it is an ID or a role, not a user, that owns an object. In other words, DB2 does not base access control on a specific user or person who need access. For example, if you allow other users to use your IDs, DB2 recognizes only the IDs, not the people or programs that use them.

ID-based privilege control within DB2

DB2 provides a wide range of granularity when you grant privileges to an ID within DB2. You can grant privileges and authorities to groups, secondary IDs, or to roles.

For example, you could, separately and specifically, grant to an ID the privilege to retrieve data from the table, to insert rows, to delete rows, or to update specific columns. By granting or not granting privileges on views of the table, you can specify exactly what an ID can do to the table, down to the granularity of specific fields. You can also grant to an ID specific privileges on databases, plans, packages, and the entire DB2 subsystem. If you grant or revoke privileges on a procedure or procedure package, all versions of that procedure or procedure package have those privileges.

DB2 also defines sets of related privileges, called *administrative authorities*. When you grant one of the administrative authorities to a person's ID, that person has all of the privileges that are associated with that administrative authority. You can efficiently grant many privileges by granting one administrative authority.

You can also efficiently grant multiple privileges by granting the privilege to execute an application plan or a package. When an ID executes a plan or package, the ID implicitly uses all of the privileges that the owner needed when binding the plan or package. Therefore, granting to an ID the privilege to execute a plan or package can provide a finely detailed set of privileges and can eliminate the need to grant other privileges separately.

Example: Assume that an application plan issues the INSERT and SELECT statements on several tables. You need to grant INSERT and SELECT privileges only to the plan owner. However, any authorization ID that is later granted the EXECUTE privilege on the plan can perform those same INSERT and SELECT statements by executing the plan. You do not need to explicitly grant the INSERT and SELECT privileges to the ID.

Recommendation: Instead of granting privileges to many primary authorization IDs, consider associating each of those primary IDs with the same secondary ID or a role if running in a trusted context. Then grant the privileges to the secondary ID or role. You can associate a primary ID with one or more secondary IDs or roles when the primary ID gains access to the DB2 subsystem. DB2 makes the association within an exit routine. The assignment of privileges to the secondary ID or role is controlled entirely within DB2.

Role-based privilege control within DB2

A privilege, which is assigned to an ID, enables the user of that ID to execute particular types of SQL statements or to access the objects of another user. A role groups privileges together so that they can be simultaneously granted to and revoked from multiple users.

RACF does not manage roles. A role is defined through the SQL CREATE ROLE statement and a trusted connection. A role cannot be used outside of a trusted context unless the user in a role grants privileges to an ID.

Privileges and authorities

You can control access within DB2 by granting or revoking privileges and related authorities that you assign to authorization IDs or roles. A *privilege* allows the capability to perform a specific operation, sometimes on a specific object.

Privileges can be explicit or implicit. An *explicit privilege* is a specific type of privilege. Each explicit privilege has a name and is the result of a GRANT statement or a REVOKE statement.

An *implicit privilege* comes from the ownership of objects, including plans and packages. For example, users are granted implicit privileges on objects that are referenced by a plan or package when they are authorized to execute the plan or package.

An administrative *authority* is a set of privileges, often covering a related set of objects. Authorities often include privileges that are not explicit, have no name, and cannot be specifically granted. For example, when an ID is granted the SYSOPR administrative authority, the ID is implicitly granted the ability to terminate any utility job.

Ownership-based privilege control within DB2

Ownership of an object carries with it a set of related privileges on the object. DB2 provides separate controls for creation and ownership of objects.

When you create an object, you can grant the ownership of the object to your ID, another ID, or the role that is assigned to the ID.

Multilevel security

Multilevel security allows you to classify objects and users with security labels that are based on hierarchical security levels and non-hierarchical security categories.

Multilevel security prevents unauthorized users from accessing information at a higher classification than their authorization, and it prevents users from declassifying information. Using multilevel security with row-level granularity, you can define security for DB2 objects and perform security checks, including row-level security checks. Row-level security checks allow you to control which users have authorization to view, modify, or perform other actions on specific rows of data.

Authorization control through exit routines

You can control access to DB2 by using a DB2-supplied exit routine or an exit routine that you write.

If your installation uses one of the access control authorization exit routines, you can use it to control authorization and authentication checking, instead of using other techniques and methods.

DB2 subsystem access control

You can control whether a process can gain access to a specific DB2 subsystem from outside of DB2. A common approach is to grant access through RACF or a similar security system.

A RACF system provides several advantages. For example, you can use RACF for the following objectives:

- Identify and verify the identifier that is associated with a process
- Connect those identifiers to RACF group names
- Log and report unauthorized attempts to access protected resources

Profiles for access to DB2 from various environments and DB2 address spaces are defined as resources to RACF. Each request to access DB2 is associated with an ID. RACF determines whether the ID is authorized for DB2 resources. If the ID is authorized, RACF permits access to DB2.

You can also consider using the security capabilities of IMS or CICS to manage access to DB2:

- *IMS terminal security* lets you limit the entry of a transaction code to a particular logical terminal (LTERM) or group of LTERMs in the system. To protect a particular program, you can authorize a transaction code that is to be entered only from any terminal on a list of LTERMs. Alternatively, you can associate each LTERM with a list of the transaction codes that a user can enter from that LTERM. IMS then passes the validated LTERM name to DB2 as the initial primary authorization ID

- *CICS transaction code security* works with RACF to control the transactions and programs that can access DB2. Within DB2, you can use the ENABLE and DISABLE options of the bind operation to limit access to specific CICS subsystems.

Managing access requests from local applications

If you request access to a local DB2 subsystem, your request is often subject to several checks before you are granted access.

If you run DB2 under TSO and use the TSO logon ID as the DB2 primary authorization ID, TSO verifies your ID when you log on. When you gain access to DB2, you can use a self-written or IBM-supplied DSN3@ATH exit routine that is connected to DB2 to perform the following actions:

- Check the authorization ID again
- Change the authorization ID
- Associate the authorization ID with secondary IDs

After these actions are performed, the authorization ID can use the services of an external security system again.

Managing access requests from remote applications

You can require remote users to pass several access checks before they reach DB2. You can use RACF or a similar security subsystem to control access from a remote location.

While controlling access from a remote locations, RACF can do the following:

- Verify an ID that is associated with a remote attachment request and check the ID with a password
- Generate *PassTickets* on the sending side. PassTickets can be used instead of passwords. A PassTicket lets a user gain access to a host system without sending the RACF password across the network.
- Verify a Kerberos ticket if your distributed environment uses Kerberos to manage user access and perform user authentication

You can also control access authentication by using the *DB2 communications database* (CDB). The CDB is a set of tables in the DB2 catalog that are used to establish conversations with remote database management systems. The CDB can translate IDs before it sends them to the remote system.

You can use the RACF DSNR general resource class for DB2 for access authentication. With RACF DSNR, you can control access to the DB2 server by the IDs that are defined to the ssnm.DIST profile with READ. In addition, you can use the port of entry (POE) checking by RACF and the z/OS communications server to protect against unauthorized remote connections to DB2.

Data set protection

The data in a DB2 subsystem is contained in data sets. The data sets can be accessed without going through DB2. If your data is sensitive, you need to control all routes to DB2 data that DB2 does not control.

RACF for data protection

If you use RACF or a similar security system to control access to DB2, the most effective way is to control access to your data sets.

If you want to use RACF for data set protection outside of DB2, define RACF profiles for data sets, and permit access to the data sets for certain DB2 IDs.

Data encryption

If your data is very sensitive, consider encrypting the data. Encryption protects against unauthorized access to data sets and to backup copies outside of DB2.

You have the following encryption options for protecting sensitive data:

- Built-in data encryption functions
- DB2 support for the Secure Socket Layer (SSL) protocol through the z/OS Communications Server IP Application Transparent Transport Layer (AT-TLS) service
- DB2 edit procedures or field procedures, which can use the Integrated Cryptographic Service Facility (ICSF)
- IBM Data Encryption for IMS and DB2 Databases tool
- IBM Encryption Facility for z/OS
- IBM System Storage™ TS1120 encryption solution

You can consider compressing your data sets before encrypting the data. Data compression is not a substitute for encryption. In some cases, the compression method does not actually shorten the data. In those cases, the data is left uncompressed and readable. If you encrypt and compress your data, compress it first. After you obtain the maximum compression, encrypt the result. When you retrieve your data, first decrypt the data. After the data is decrypted, decompress the result.

Scenario: Securing data access at Spiffy Computer

This scenario describes a simple approach to secure local and remote access to the sensitive data of employees, payroll operations, and payroll management at Spiffy Computer. It shows how to enforce a security plan by using authorization IDs, privileges and authorities, and the audit trace.

You should base your security plan, techniques, and procedures on your actual security objectives; do not view this sample security plan as an exact model for your security needs. Instead, use it to understand various possibilities and address problem areas that you might encounter when you implement your security plan.

Determining security objectives

An important step to defining and implementing an effective security plan is to determine your security objectives.

Suppose that the Spiffy Computer Company management team determines the following security objectives:

- Managers can see, but not update, all of the employee data for members of their own departments.
- Managers of managers can see all of the data for employees of departments that report to them.
- The employee table resides at a central location. Managers at remote locations can query the data in the table.

- The payroll operations department makes changes to the employee table. Members of the payroll operations department can update any column of the employee table except for the salary, bonus, and commission columns.
- Members of payroll operations can update any row except for rows that are for members of their own department. Because changes to the table are made only from a central location, distributed access does not affect payroll operations.
- Changes to the salary, bonus, and commission columns are made through a process that involves the payroll update table. When an employee's compensation changes, a member of the payroll operations department can insert rows in the payroll update table. For example, a member of the payroll operations department might insert a row in the compensation table that lists an employee ID and an updated salary. Next, the payroll management group can verify inserted rows and transfer the changes to the employee table.
- No one else can see the employee data. The security plan cannot fully achieve this objective because some ID must occasionally exercise SYSADM authority. While exercising SYSADM authority, an ID can retrieve any data in the system. The security plan uses the trace facility to monitor the use of that power.

Securing manager access to employee data

As a security measurement, the Spiffy Computer Company sets clear restrictions on how its managers can access employee data.

Specifically, it imposes the following security restrictions on managers:

- Managers can retrieve, but not change, all information in the employee table for members of their own departments.
- Managers of managers have the same privileges for their own departments and for the departments that directly report to them.

Creating views of employee data

The Spiffy security planners decide to use *views* for implementing the restrictions on managers' access to employee data.

To create a view of employee data for every employee that reports to a manager, the Spiffy security planners perform the following steps:

1. Add a column that contains manager IDs to DSN8910.DEPT, as shown in the following statement:

```
ALTER TABLE DSN8910.DEPT
  ADD MGRID CHAR(8) FOR SBCS DATA NOT NULL WITH DEFAULT;
```

2. Create a view that selects employee information about employees that work for a given manager, as shown in the following statement:

```
CREATE VIEW DEPTMGR AS
  SELECT * FROM DSN8910.EMP, DSN8910.DEPT
  WHERE WORKDEPT = DEPTNO
  AND MGRID = USER;
```

3. Ensure that every manager has the SELECT privilege on the view.

Granting managers the SELECT privilege

The security planners for Spiffy Computer Company can take an "individual" or "functional" approach when they grant the SELECT privilege on a view to managers.

With an individual approach, they can grant privileges to individual IDs and revoke them if the user of the ID leaves the company or transfers to another position. With a functional approach, they can create RACF groups, and grant

privileges to the group IDs, with the intention of never revoking them. When an individual ID needs those privileges, connect that ID to the group; disconnect the ID when its user leaves or transfers.

The Spiffy security planners know that the functional approach is usually more convenient in the following situations:

- Each function, such as the manager function, requires many different privileges. When functional privileges are revoked from one user, they must be granted to another user.
- Several users need the same set of privileges.
- The privileges are given with the grant option, or the privileges let users create objects that must persist after their original owners leave or transfer. In both cases, revoking the privileges might not be appropriate. The revokes cascade to other users. To change ownership, you might need to drop objects and re-create them.

Some of the Spiffy requirements for securing manager access suggest the functional approach. However, in this case, the function needs only one privilege. The privilege does not carry the grant option, and the privilege does not allow new objects to be created.

Therefore, the Spiffy security planners choose the individual approach, and plan to re-examine their decision later. Spiffy security planners grant all managers the SELECT privilege on the views for their departments.

Example: To grant the SELECT privilege on the DEPTMGR view to the manager with ID EMP0060, the planners use the following GRANT statement:

```
GRANT SELECT ON DEPTMGR TO EMP0060;
```

Managing distributed access

Some managers must use views to query data in the central employee table from remote locations. The security plan must ensure that this type of distributed access is secure. The Spiffy security planners must implement a sound plan for distributed access.

Planning for distributed access:

The Spiffy security plan needs to define how the managers can securely access employee data in a distributed environment.

To secure distributed access to employee data, the Spiffy security planners must address the following questions:

- Which IDs should hold privileges on which views?
- How do the central location and the remote locations divide security responsibilities for IDs?

The Spiffy security planners answer those questions with the following decisions:

- IDs that are managed at the central location hold privileges on views for departments that are at remote locations. For example, the ID MGRD11 has the SELECT privilege on the view DEPTD11.
- If the manager of Department D11 uses a remote system, the ID at that system must be translated to MGRD11. Then a request is sent to the central system. All other IDs are translated to CLERK before they are sent to the central system.
- The communications database (CDB) manages the translated IDs, like MGRD11.

- An ID from a remote system must be authenticated on any request to the central system.

Implementing distributed access at the central server:

To enable distributed access to sensitive employee data, the Spiffy security plan requires certain security measures to be implemented at the central server location.

The following actions must occur at the central server location:

- The central DB2 subsystem must authenticate every incoming ID with RACF.
- For SNA connections, the Spiffy security planners must include an entry in table SYSIBM.LUNAMES in the CDB; the entry in the LUNAME column identifies the LU name of every remote location. The entry must specify that connections must be verified.

Example: The following table shows an entry in SYSIBM.LUNAMES for LUREMOTE.

Table 32. The SYSIBM.LUNAMES table at the central location

LUNAME	USERNAMES	SECURITY_IN	ENCRYPTPSWDS
LUREMOTE	blank	V	N

The value of V for SECURITY_IN indicates that incoming remote connections must include verification. The value of N for ENCRYPTPSWDS indicates that passwords are not in internal RACF encrypted format.

The security plan treats all remote locations alike, so it does not require encrypted passwords. The option to require encrypted passwords is available only between two DB2 subsystems that use SNA connections.

- For TCP/IP connections, the Spiffy security planners must set the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5 to NO. This setting ensures that the incoming requests that use TCP/IP are not accepted without authentication.
- The Spiffy security planners must grant all privileges and authorities that are required by the manager of Department D11 to the ID, MGRD11. The security planners must grant similar privileges to IDs that correspond to the remaining managers.

Implementing distributed access at remote locations:

To enable distributed access to sensitive employee data, the Spiffy security plan requires certain security measures to be implemented at the remote locations.

The following actions must occur at the remote locations to enable distributed access for the Spiffy security plan:

- For SNA connections, the Spiffy security planners must include an entry in table SYSIBM.LUNAMES for the LU name of the central location. The entry must specify an outbound ID translation for attachment requests to that location.

Example: The following table shows an entry in SYSIBM.LUNAMES for LUCENTRAL.

Table 33. The SYSIBM.LUNAMES table at the remote location

LUNAME	USERNAMES	SECURITY_OUT
LUCENTRAL	O	P

The value of O for USERNAMES indicates that translation checking is

performed on outbound IDs, but not on inbound IDs. The value of P for SECURITY_OUT indicates that outbound connection requests contain a user password and a RACF PassTicket.

- For TCP/IP connections, the Spiffy security planners must include an entry in table SYSIBM.IPNAMES for the LU name that is used by the central location. The content of the LUNAME column is used to generate RACF PassTickets. The entry must specify outbound ID translation for requests to that location.

Example: The following table shows an entry in SYSIBM.IPNAMES for LUCENTRAL.

Table 34. The SYSIBM.IPNAMES table at the remote location

LINKNAME	USERNAMES	SECURITY_OUT	IPADDR
LUCENTRAL		R	central.vnet.ibm.com

- The Spiffy security planners must include entries in table SYSIBM.USERNAMES to translate outbound IDs.

Example: The following table shows two entries in SYSIBM.USERNAMES.

Table 35. The SYSIBM.USERNAMES table at the remote location

TYPE	AUTHID	LINKNAME	NEWAUTHID
O	MEL1234	LUCENTRAL	MGRD11
O	blank	LUCENTRAL	CLERK

MEL1234 is translated to MGRD11 before it is sent to the LU that is specified in the LINKNAME column. All other IDs are translated to CLERK before they are sent to that LU.

Exception: For a product other than DB2 for z/OS, the actions at the remote location might be different. If you use a different product, check the documentation for that product. The remote product must satisfy the requirements that are imposed by the central subsystem.

Auditing manager access

The Spiffy payroll data is extremely sensitive. The security plan requires that the audit trace is automatically started for all classes whenever DB2 is started.

To ensure that an audit record exists for every access to the employee table, the Spiffy security planners create the employee table with AUDIT ALL. Every week, the security planners scan the records and determine the number of accesses by each manager.

The report highlights any number of accesses outside an expected range. The Spiffy system operator makes a summary of the reports every two months, and scans it for unusual patterns of access. A large number of accesses or an unusual pattern might reveal use of a manager's logon ID by an unauthorized employee.

Securing access to payroll operations and management

As a security measurement, the Spiffy security plan sets clear restrictions on how members of the payroll operations department access and handle sensitive payroll information.

The plan imposes the following restrictions on members of the payroll operations department:

- Members of the payroll operations department can update any column of the employee table except for SALARY, BONUS, and COMM.
- Members of payroll operations can update any row except for rows that are for members of their own department.

Because changes to the table are made only from the central location, distributed access does not affect payroll operations.

Creating views of payroll operations

The Spiffy security planners decide to use *views* for implementing the security objectives for members of the payroll operations department.

The PAYDEPT view shows all the columns of the employee table except for job, salary, bonus, and commission. The view does not show the rows for members of the payroll operations department.

Example: The WORKDEPT value for the payroll operations department is P013. The owner of the employee table uses the following statement to create the PAYDEPT view:

```
CREATE VIEW PAYDEPT AS
  SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME, WORKDEPT,
         PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE
  FROM DSN8910.EMP
  WHERE WORKDEPT<>'P013'
  WITH CHECK OPTION;
```

The CHECK OPTION ensures that every row that is inserted or updated through the view conforms to the definition of the view.

A second view, the PAYMGR view, gives Spiffy payroll managers access to any record, including records for the members of the payroll operations department.

Example: The owner of the employee table uses the following statement to create the PAYMGR view:

```
CREATE VIEW PAYMGR AS
  SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME, WORKDEPT,
         PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE
  FROM DSN8910.EMP
  WITH CHECK OPTION;
```

Neither PAYDEPT nor PAYMGR provides access to compensation amounts. When a row is inserted for a new employee, the compensation amounts remain null. An update process can change these values at a later time. The owner of the employee table creates, owns, and grants privileges on both views.

Securing compensation accounts with update tables

The Spiffy security plan does not allow members of payroll operations to update compensation amounts directly. Instead, a separate payroll update table contains the employee ID, job, salary, bonus, and commission.

Members of payroll operations make all job, salary, and bonus changes to the payroll update table, except those for their own department. After they verify the prospective changes, the managers of payroll operations run an application program. The program reads the payroll update table and makes the corresponding changes to the employee table. Only the payroll update program has the privilege of updating job, salary, and bonus in the employee table.

The Spiffy Computer Company calculates commission amounts separately by using a complicated formula. The formula considers the employee's job, department, years of service with the company, and responsibilities for various projects. The formula is embedded in the commission program, which is run regularly to insert new commission amounts in the payroll update table. The plan owner must have the SELECT privilege on the employee table and other tables to run the commission program.

Securing compensation updates with other measures

By separating potential salary changes into the payroll update table, the Spiffy security planners allow payroll management to verify changes before they go into effect.

At Spiffy Computer Company, managers check the changes against a written change request that is signed by a required level of management. The Spiffy security planners consider that check to be the most important control on salary updates, but the plan also includes the following controls:

- The employee ID in the payroll update table is a foreign key column that refers to the employee ID in the employee table. Enforcing the referential constraint prevents an employee ID from being changed to an invalid value.
- The employee ID in the payroll update table is also a primary key for that table. Therefore, the values in the employee ID column must be unique. Because of enforced uniqueness, every change that is made for any one employee during a given operating period must appear in the same row of the table. No two rows can carry conflicting changes.

The Spiffy security plan documents an allowable range of salaries, bonuses, and commissions for each job level. To keep the values within the allowable ranges, the Spiffy security planners use table check constraints for the salaries, bonuses, and commissions. The planners use this approach because it is both simple and easy to control.

In a similar situation, you might also consider the following ways to ensure that updates and inserts stay within certain ranges:

- Keep the ranges in a separate DB2 table. To verify changes, query the payroll update table and the table of ranges. Retrieve any rows for which the planned update is outside the allowed range.
- Build the ranges into a validation routine. Apply the validation routine to the payroll update table to automatically reject any insert or update that is outside the allowed range.
- Embody the ranges in a view of the payroll table, using WITH CHECK OPTION, and make all updates to the view. The ID that owns the employee table also owns the view.
- Create a trigger to prevent salaries, bonuses, and commissions from increasing by more than the percent that is allowed for each job level.

Granting privileges to payroll operations and management

The Spiffy security plan for the payroll operations department strongly suggests the functional approach, for the following reasons:

- Payroll operations members require several privileges, including the SELECT, INSERT, UPDATE, and DELETE privileges on the PAYDEPT view.
- Several members of the department require the same set of privileges.
- If members of the department leave, others are hired or transferred to replace the departing members.

Therefore, the security plan calls for the creation of two RACF groups, with one for the payroll operations and another for the payroll management.

Creating a RACF group for payroll operations:

The security plan calls for the creation of a RACF group for the payroll operations department. DB2USER can define the group and retain its ownership, or it can assign the ownership to an ID that is used by payroll management.

The owner of the employee table can grant the privileges that the group requires. The owner grants all required privileges to the group ID, with the intent not to revoke them. The primary IDs of new members of the department are connected to the group ID, which becomes a secondary ID for each of them. The primary IDs of members who leave the department are disconnected from the group ID.

Example: The following statement grants the SELECT, INSERT, UPDATE, and DELETE privileges on the PAYDEPT view to the payroll operations group ID PAYOPS:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON PAYDEPT TO PAYOPS;
```

This statement grants the privileges without the GRANT OPTION to keep members of payroll operations from granting privileges to other users.

Creating a RACF group for payroll management:

The payroll managers require different privileges and a different RACF group ID. The Spiffy security planners add a RACF group for payroll managers and name it PAYMGRS.

The security planners associate the payroll managers' primary IDs with the PAYMGRS secondary ID. Next, privileges on the PAYMGR view, the compensation application, and the payroll update application are granted to PAYMGRS. The payroll update application must have the appropriate privileges on the update table.

Example: The following statement grants the SELECT, INSERT, UPDATE, and DELETE privileges on the PAYMGR view to the payroll managers' group ID PAYMGRS:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON PAYMGR TO PAYMGRS;
```

Example: The following statement grants the EXECUTE privilege on the compensation application:

```
GRANT EXECUTE ON PLAN COMPENS TO PAYMGRS;
```

Auditing payroll operations and management

Like the employee table, the payroll update table is created with AUDIT ALL. For both tables, the audit trace reports the number of accesses by the payroll operations and payroll management groups. The Spiffy security planners scan the reports of payroll access for large numbers or unusual patterns of access.

Managing access privileges of other authorities

In addition to the privileges of managers, the payroll operations department, and payroll management, the security plan considers the privileges of the following roles:

Managing access by the DBADM authority

An ID with the DBADM authority on a database has many privileges on that database and its tables. These privileges include the SELECT, INSERT, DELETE, UPDATE, and ALTER statements on any table in the database, and the CREATE and DROP statements on indexes for those tables.

For security reasons, the Spiffy security planners prefer not to grant all of the privileges that come with DBADM authority on DSN8D91A. DSN8D91A is the database that holds the employee table and the payroll update table.

The Spiffy security planners prefer to grant DBCTRL authority on the database because granting DBCTRL authority does not expose as many security risks as granting DBADM authority. DBCTRL authority allows an ID to support the database without allowing the ID to retrieve or change the data in the tables. However, database DSN8D91A contains several additional tables. These additional tables require some of the privileges that are included in DBADM authority but not included in DBCTRL authority.

The Spiffy security planners decide to compromise between the greater security of granting DBCTRL authority and the greater flexibility of granting DBADM authority. To balance the benefits of each authority, the Spiffy security planners create an administrative ID with some, but not all of the DBADM privileges. The security plan calls for a RACF group ID with the following authorities and privileges:

- DBCTRL authority over DSN8D81A
- The INDEX privilege on all tables in the database except the employee table and the payroll update table
- The SELECT, INSERT, UPDATE, and DELETE privileges on certain tables, excluding the employee table and the payroll update table

An ID with SYSADM authority grants the privileges to the group ID.

In a similar situation, you also might consider putting the employee table and the payroll update table in a separate database. Then you can grant DBADM authority on DSN8D91A, and grant DBCTRL authority on the database that contains the employee table and the payroll update table.

Managing access by the SYSADM authority

An ID with SYSADM authority can access data from any table in the entire DB2 subsystem, including the employee table and the payroll update table. The Spiffy security planners want to minimize the security risk that is associated with granting SYSADM authority by granting the authority to as few users as possible.

The planners know that the subsystem might require SYSADM authority only for certain tasks and only for relatively short periods. They also know that the privileges that are associated with the SYSADM authority give an ID control over all of the data in a subsystem.

To limit the number of users with SYSADM authority, the Spiffy security plan grants the authority to DB2OWNER, the ID that is responsible for DB2 security. That does not mean that only IDs that are connected to DB2OWNER can exercise privileges that are associated with SYSADM authority. Instead, DB2OWNER can grant privileges to a group, connect other IDs to the group as needed, and later disconnect them.

The Spiffy security planners prefer to have multiple IDs with SYSCTRL authority instead of multiple IDs with SYSADM authority. IDs with SYSCTRL authority can exercise most of the SYSADM privileges and can assume much of the day-to-day work. IDs with SYSCTRL authority cannot access data directly or run plans unless the privileges for those actions are explicitly granted to them. However, they can run utilities, examine the output data sets, and grant privileges that allow other IDs to access data. Therefore, IDs with SYSCTRL authority can access some sensitive data, but they cannot easily access the data. As part of the Spiffy security plan, DB2OWNER grants SYSCTRL authority to selected IDs.

The Spiffy security planners also use ROLES, RACF group IDs, and secondary IDs to relieve the need to have SYSADM authority continuously available. SYSADM grants the necessary privileges to a ROLE, RACF group ID, or secondary ID. IDs that have this ROLE, RACF group ID, or secondary ID can then bind plans and packages it owns.

Managing access by object owners

The Spiffy security plan must consider the ID that owns and grants privileges on the tables, views, and programs. The ID that owns these objects has many implicit privileges on the objects, including the SELECT and UPDATE privileges on the employee table. The owner of the objects can also grant privileges on the objects to other users.

The Spiffy security planners want to limit the number of IDs that have privileges on the employee table and the payroll update table to the smallest convenient value. To meet that objective, they decide that the owner of the employee table should issue all of the CREATE VIEW and GRANT statements. They also decide to have the owner of the employee table own the plans and packages that are associated with employee data. The employee table owner implicitly has the following privileges, which the plans and packages require:

- The owner of the payroll update program must have the SELECT privilege on the payroll update table and the UPDATE privilege on the employee table.
- The owner of the commission program must have the UPDATE privilege on the payroll update table and the SELECT privilege on the employee table.
- The owners of several other payroll programs must have the proper privileges to do payroll processing, such as printing payroll checks, writing summary reports, and so on.

To bind these plans and packages, an ID must have the BIND or BINDADD privileges. The list of privileges that are required by the owner of the employee table suggests the functional approach. The Spiffy security planners create a RACF group for the owner of the employee table.

Managing access by other users

Exceptions occur when any user other than the following users accesses the employee table or the payroll table:

- Department managers
- Members of the payroll operations department
- Payroll managers
- The payroll update program

The audit report lists each exception in full. Auditors check each exception to determine whether it was a planned operation by the users with SYSADM or DBADM authority, or the employee table owner.

The audit report also lists denials of access to the tables. Those denials represent attempts by unauthorized IDs to use the tables. Some are possibly accidental; others can be attempts to violate the security system.

After running the periodic reports, the security planners archive the audit records. The archives provide a complete audit trail of access to the employee data through DB2.

Chapter 5. Managing access through authorization IDs or roles

DB2 controls access to its objects and data through authorization identifiers (IDs) or roles and the privileges that are assigned to them. Each privilege and its associated authorities enable you to take specific actions on an object. Therefore, you can manage access to DB2 objects through authorization IDs or roles.

As the following diagram shows, you can grant privileges and authorities to IDs or roles and control access to data in four primary ways:

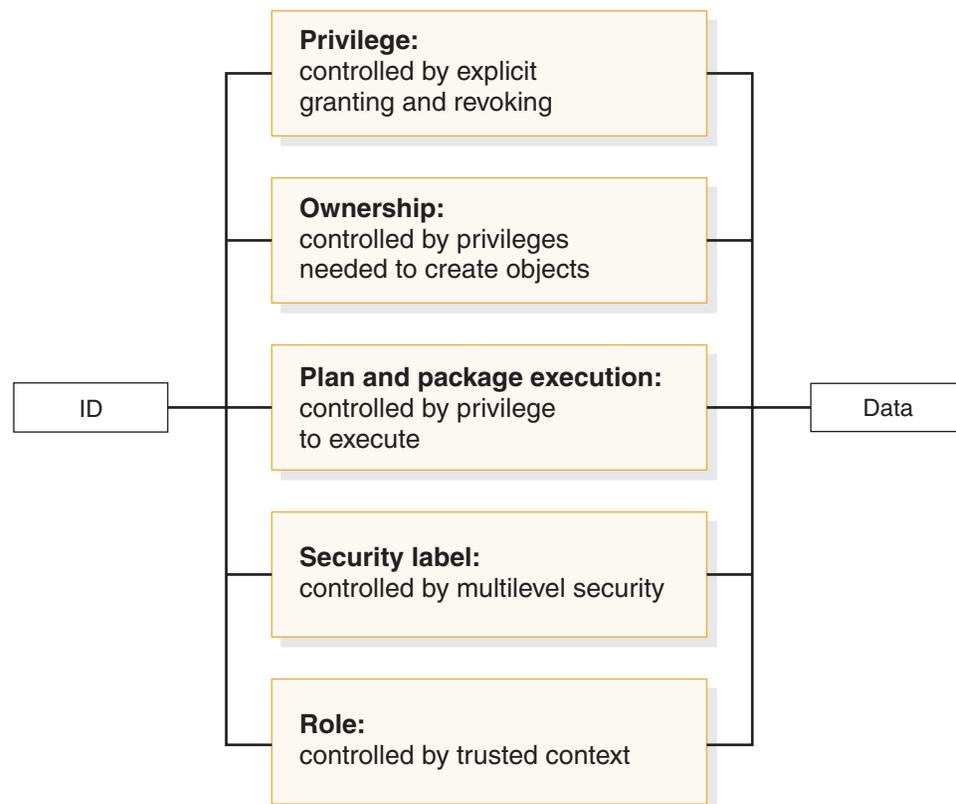


Figure 15. Access to objects and data within DB2

1. Granting and revoking explicit privileges through authorization IDs or roles. DB2 has primary authorization IDs, secondary authorization IDs, roles, and SQL IDs. Some privileges can be exercised by only one type of ID or a role; other privileges can be exercised by multiple IDs or roles. The DB2 catalog records the privileges that IDs are granted and the objects that IDs own.
2. Managing implicit privileges through ownership of objects other than plans and packages.
3. Managing implicit privileges through ownership of plans and packages.
4. Controlling access through security labels.

Certain privileges and authorities are assigned when you install DB2. You can reassign these authorities by changing the DSNZPARM subsystem parameter.

As a security planner, you must be aware of these ways to manage privileges and authorities through authorization IDs and roles before you write a security plan. After you decide how to authorize access to data, you can implement it through your security plan.

Authorization IDs and roles

You can control access to DB2 objects by assigning privileges and authorities to an authorization ID or a role.

Authorization IDs

Every process that connects to or signs on to DB2 is represented by one or more DB2 short identifiers (IDs) that are called *authorization IDs*. Authorization IDs are assigned to a process by default procedures or by user-written exit routines.

When authorization IDs are assigned, every process receives exactly one ID that is called the *primary authorization ID*. All other IDs are *secondary authorization IDs*. Furthermore, one ID (either primary or secondary) is designated as the current *SQL ID*. You can change the value of the SQL ID during your session. More details about these IDs are as follows:

Role A role is available within a trusted context. You can define a role and assign it to authorization IDs in a trusted context. When associated with a role and using the trusted connection, an authorization ID inherits all the privileges granted to that role.

Primary authorization ID

Generally, the primary authorization ID identifies a process. For example, statistics and performance trace records use a primary authorization ID to identify a process.

Secondary authorization ID

A secondary authorization ID, which is optional, can hold additional privileges that are available to the process. For example, a secondary authorization ID can be a Resource Access Control Facility (RACF) group ID.

SQL ID

An SQL ID holds the privileges that are exercised when certain dynamic SQL statements are issued. The SQL ID can be set equal to the primary ID or any of the secondary IDs. If an authorization ID of a process has SYSADM authority, the process can set its SQL ID to any authorization ID.

RACF ID

The RACF ID is generally the source of the primary and secondary authorization IDs (RACF groups). When you use the RACF Access Control Module or multilevel security, the RACF ID is used directly.

Roles in a trusted context

A *role* is a database entity that groups one or more privileges together in a trusted context. System administrators can use roles to control access to enterprise objects in a way that parallels the structure of the enterprise.

A role is available only in a trusted context. A *trusted context* is an independent database entity that you can define based on a system authorization ID and connection trust attributes. The trust attributes specify a set of characteristics about

a specific connection. These attributes include the IP address, domain name, or SERVAUTH security zone name of a remote client and the job or task name of a local client.

DB2 for z/OS extends the trusted context concept to allow for the assignment of a role to a trusted context. An authorization ID that uses the trusted context can inherit the privileges that are assigned to this role, in addition to the privileges that are granted to the ID.

Using roles provides the flexibility for managing context-specific privileges and simplifies the processing of authorization. Specific roles can be assigned to the authorization IDs that use the trusted connection. When your authorization ID is associated with an assigned role in the trusted context, you inherit all privileges that are granted by that role, instead of those by the default role, because the role-based privileges override the privileges that are associated with the default role.

Privileges and authorities

You can control access within DB2 by granting or revoking privileges and related authorities that you assign to authorization IDs or roles. A *privilege* allows the capability to perform a specific operation, sometimes on a specific object.

Privileges can be explicit or implicit. An *explicit privilege* is a specific type of privilege. Each explicit privilege has a name and is the result of a GRANT statement or a REVOKE statement.

An *implicit privilege* comes from the ownership of objects, including plans and packages. For example, users are granted implicit privileges on objects that are referenced by a plan or package when they are authorized to execute the plan or package.

An administrative *authority* is a set of privileges, often covering a related set of objects. Authorities often include privileges that are not explicit, have no name, and cannot be specifically granted. For example, when an ID is granted the SYSOPR administrative authority, the ID is implicitly granted the ability to terminate any utility job.

Explicit privileges

You can explicitly grant privileges on objects to authorization IDs or roles.

You can explicitly grant privileges on the following objects:

- Collections
- Databases
- Distinct types or JAR
- Functions or procedures
- Packages
- Plans
- Routines
- Schemas
- Sequences
- Systems
- Tables and views
- Usage
- Use

Explicit collection privileges

DB2 supports the following collection privileges:

GUPI

Table 36. Explicit collection privileges

Collection privilege	Operations allowed for a named package collection
CREATE IN	The BIND PACKAGE subcommand, to name the collection

GUPI

Explicit database privileges

DB2 supports the following database privileges:

GUPI

Table 37. Explicit database privileges

Database privilege	Operations allowed on a named database
CREATETAB	The CREATE TABLE statement, to create tables in the database.
CREATETS	The CREATE TABLESPACE statement, to create table spaces in the database
DISPLAYDB	The DISPLAY DATABASE command, to display the database status
DROP	The DROP and ALTER DATABASE statements, to drop or alter the database
IMAGCOPY	The QUIESCE, COPY, and MERGECOPY utilities, to prepare for, make, and merge copies of table spaces in the database; the MODIFY RECOVERY utility, to remove records of copies
LOAD	The LOAD utility, to load tables in the database
RECOVERDB	The RECOVER, REBUILD INDEX, and REPORT utilities, to recover objects in the database and report their recovery status
REORG	The REORG utility, to reorganize objects in the database
REPAIR	The REPAIR and DIAGNOSE utilities (except REPAIR DBD and DIAGNOSE WAIT) to generate diagnostic information about, and repair data in, objects in the database
STARTDB	The START DATABASE command, to start the database
STATS	The RUNSTATS, CHECK, LOAD, REBUILD INDEX, REORG INDEX, and REORG TABLESPACE, and MODIFY STATISTICS utilities, to gather statistics, check indexes and referential constraints for objects in the database, and delete unwanted statistics history records from the corresponding catalog tables
STOPDB	The STOP DATABASE command, to stop the database

Database privileges that are granted on DSNDB04 apply to all implicitly created databases. For example, if you have the DBADM authority on DSNDB04, you can select data from any table in any implicitly created database. If you have the STOPDB privilege on DSNDB04, you can stop any implicitly created database. However, you cannot grant the same authorities or privileges to others on any implicitly created database.

GUIP**Explicit package privileges**

DB2 supports the following package privileges:

GUIP

Table 38. Explicit package privileges

Package privilege	Operations allowed for a named package
BIND	The BIND, REBIND, and FREE PACKAGE subcommands, and the DROP PACKAGE statement, to bind or free the package, and, depending on the installation option BIND NEW PACKAGE, to bind a new version of a package
COPY	The COPY option of BIND PACKAGE, to copy a package
EXECUTE	Inclusion of the package in the PKLIST option of BIND PLAN
GRANT ALL	All package privileges

GUIP**Explicit plan privileges**

DB2 supports the following plan privileges:

GUIP

Table 39. Explicit plan privileges

Plan privilege	Subcommands allowed for a named application plan
BIND	BIND, REBIND, and FREE PLAN, to bind or free the plan
EXECUTE	RUN, to use the plan when running the application

GUIP**Explicit routine privileges**

DB2 supports the following routine privileges:

GUIP

Table 40. Explicit routine privileges

Routine privileges	Objects available for usage
EXECUTE ON FUNCTION	A user-defined function
EXECUTE ON PROCEDURE	A stored procedure

GUIP**Explicit schema privileges**

DB2 supports the following schema privileges:

GUIP

Table 41. Explicit schema privileges

Schema privileges	Operations available for usage
CREATEIN	Create distinct types, user-defined functions, triggers, and stored procedures in the designated schemas
ALTERIN	Alter user-defined functions or stored procedures, or specify a comment for distinct types, user-defined functions, triggers, and stored procedures in the designated schemas
DROPIN	Drop distinct types, user-defined functions, triggers, and stored procedures in the designated schemas

GUIP

Explicit system privileges

DB2 supports the following system privileges:

GUIP

Table 42. Explicit system privileges

System privilege	Operations allowed on the system
ARCHIVE	The ARCHIVE LOG command, to archive the current active log, the DISPLAY ARCHIVE command, to give information about input archive logs, the SET LOG command, to modify the checkpoint frequency specified during installation, and the SET ARCHIVE command, to control allocation and deallocation of tape units for archive processing.
BINDADD	The BIND subcommand with the ADD option, to create new plans and packages
BINDAGENT	The BIND, REBIND, and FREE subcommands, and the DROP PACKAGE statement, to bind, rebind, or free a plan or package, or copy a package, on behalf of the grantor. The BINDAGENT privilege is intended for separation of function, not for added security. A bind agent with the EXECUTE privilege might be able to gain all the authority of the grantor of BINDAGENT.
BSDS	The RECOVER BSDS command, to recover the bootstrap data set
CREATEALIAS	The CREATE ALIAS statement, to create an alias for a table or view name
CREATEDBA	The CREATE DATABASE statement, to create a database and have DBADM authority over it
CREATEDBC	The CREATE DATABASE statement, to create a database and have DBCTRL authority over it
CREATESG	The CREATE STOGROUP statement, to create a storage group
CREATETMTAB	The CREATE GLOBAL TEMPORARY TABLE statement, to define a created temporary table
DEBUGSESSION	The DEBUGINFO connection attribute, to control debug session activity for native SQL and Java™ stored procedures
DISPLAY	The DISPLAY ARCHIVE, DISPLAY BUFFERPOOL, DISPLAY DATABASE, DISPLAY LOCATION, DISPLAY LOG, DISPLAY THREAD, and DISPLAY TRACE commands, to display system information

Table 42. Explicit system privileges (continued)

System privilege	Operations allowed on the system
MONITOR1	Receive trace data that is not potentially sensitive
MONITOR2	Receive all trace data
RECOVER	The RECOVER INDOUBT command, to recover threads
STOPALL	The STOP DB2 command, to stop DB2
STOSPACE	The STOSPACE utility, to obtain data about space usage
TRACE	The START TRACE, STOP TRACE, and MODIFY TRACE commands, to control tracing

GUIP

Explicit table and view privileges

DB2 supports the following table and view privileges:

GUIP

Table 43. Explicit table and view privileges

Table or view privilege	SQL statements allowed for a named table or view
ALTER	ALTER TABLE, to change the table definition
DELETE	DELETE, to delete rows
INDEX	CREATE INDEX, to create an index on the table
INSERT	INSERT, to insert rows
REFERENCES	ALTER or CREATE TABLE, to add or remove a referential constraint that refers to the named table or to a list of columns in the table
SELECT	SELECT, to retrieve data
TRIGGER	CREATE TRIGGER, to define a trigger on a table
UPDATE	UPDATE, to update all columns or a specific list of columns
GRANT ALL	SQL statements of all privileges

GUIP

Explicit usage privileges

DB2 supports the following usage privileges:

GUIP

Table 44. Explicit usage privileges

Usage privileges	Objects available for usage
USAGE ON DISTINCT TYPE	A distinct type
USAGE ON JAR (Java class for a routine)	A Java class
USAGE ON SEQUENCE	A sequence

GUIP

Explicit use privileges

DB2 supports the following use privileges:

GUIP

Table 45. Explicit use privileges

Use privileges	Objects available for use
USE OF BUFFERPOOL	A buffer pool
USE OF STOGROUP	A storage group
USE OF TABLESPACE	A table space

GUIP

Implicit privileges through object ownership

When you create a DB2 object by issuing an SQL statement, you establish its name and its ownership. By default, the owner implicitly holds certain privileges on the object.

GUIP

However, this general rule does not apply to a plan or package that is not created with SQL CREATE statements. In other words, when you own an object other than a plan or package, you have implicit privileges over the object. The following table describes the implicit privileges of ownership for each type of object:

Table 46. Implicit privileges of ownership by object type

Object type	Implicit privileges of ownership
Alias	To drop the alias
Database	DBCTRL or DBADM authority over the database, depending on the privilege (CREATEDBC or CREATEDBA) that is used to create it. DBCTRL authority does not include the privilege to access data in tables in the database.
Distinct type	To use or drop a distinct type
Index	To alter, comment on, or drop the index
JAR (Java class for a routine)	To replace, use, or drop the JAR
Package	To bind, rebind, free, copy, execute, drop, or comment on the package
Plan	To bind, rebind, free, execute, or comment on the plan
Role	To create, alter, commit, drop, or comment on the role
Sequence	To alter, comment on, use, or drop the sequence
Storage group	To alter or drop the group and to name it in the USING clause of a CREATE INDEX or CREATE TABLESPACE statement
Stored procedure	To execute, alter, drop, start, stop, or display a stored procedure
Synonym	To use or drop the synonym

Table 46. Implicit privileges of ownership by object type (continued)

Object type	Implicit privileges of ownership
Table	<ul style="list-style-type: none"> • To alter or drop the table or any indexes on it • To lock the table, comment on it, or label it • To create an index or view for the table • To select or update any row or column • To insert or delete any row • To use the LOAD utility for the table • To define referential constraints on any table or set of columns • To create a trigger on the table • To comment on the table
Table space	To alter or drop the table space and to name it in the IN clause of a CREATE TABLE statement
Trusted context	To create, alter, commit, revoke, or comment on the trusted context
User-defined functions	To execute, alter, drop, start, stop, or display a user-defined function
View	<ul style="list-style-type: none"> • To drop, comment on, or label the view, or to select any row or column • To update any row or column, insert or delete any row (if the view is not read-only)

GUIP

Administrative authorities

Within DB2, privileges are grouped into nine administrative authorities.

GUIP As shown in the following diagram, the administrative authorities form a branched hierarchy:

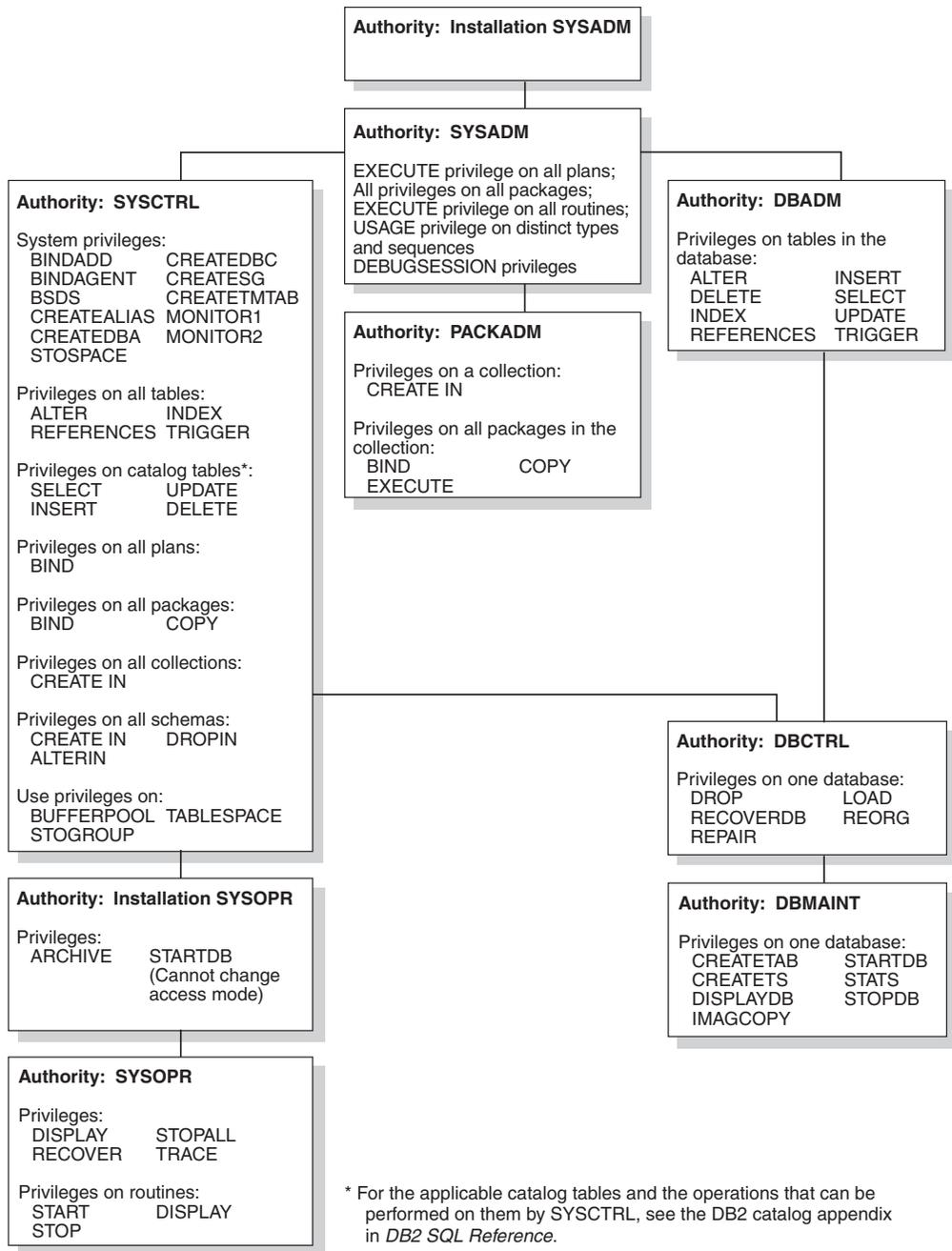


Figure 16. The hierarchy of administrative authorities

An authority at the top of the hierarchy has all the privileges that the authorities below it have. For example, DBADM has the privileges that DBCTRL and DBMAINT have, in addition to the DBADM privileges. The following table lists the nine administrative authorities and the specific privileges that are vested in each of them by the hierarchy:

Table 47. Administrative authorities and the authorities vested in them by the hierarchy

Administrative authority	Included authorities
Installation SYSADM	SYSADM, SYSCTRL, DBADM, Installation SYSOPR, SYSOPR, PACKADM, DBADM, DBCTRL, DBMAINT

Table 47. Administrative authorities and the authorities vested in them by the hierarchy (continued)

Administrative authority	Included authorities
SYSADM	SYSCTRL, DBADM, Installation SYSOPR, SYSOPR, PACKADM, DBADM, DBCTRL, DBMAINT
SYSCTRL	Installation SYSOPR, SYSOPR, DBADM, DBCTRL, DBMAINT
Installation SYSOPR	SYSOPR
SYSOPR	(none)
DBADM	DBCTRL, DBMAINT
DBCTRL	DBMAINT
DBMAINT	(none)
PACKADM	(none)

GUPI

Installation SYSADM

The installation SYSADM authority is assigned to one or two IDs when DB2 is installed; it cannot be assigned to a role. These IDs have all the privileges of the SYSADM authority.

GUPI No other IDs can revoke the installation SYSADM authority; you can remove the authority only by changing the module that contains the subsystem initialization parameters (typically DSNZPARM).

In addition, DB2 does not record the installation SYSADM authority in the catalog. Therefore, the catalog does not need to be available to check installation SYSADM authority. The authority outside of the catalog is crucial. For example, if the directory table space DBD01 or the catalog table space SYSDBAUT is stopped, DB2 might not be able to check the authority to start it again. In this case, only an installation SYSADM can start it.

IDs with the installation SYSADM authority can also perform the following actions:

- Run the CATMAINT utility
- Access DB2 when the subsystem is started with ACCESS(MAINT)
- Start databases DSNDB01 and DSNDB06 when they are stopped or in restricted status
- Run the DIAGNOSE utility with the WAIT statement
- Start and stop the database that contains the application registration table (ART) and the object registration table (ORT). **GUPI**

SYSADM

The SYSADM authority includes all SYSCTRL, PACKADM, and DBADM privileges, including access to all data.

GUPI With the SYSADM authority, an authorization ID can perform the following actions and grant other IDs the privileges to perform them:

- Use all the privileges of DBADM over any database
- Use EXECUTE privileges on all packages
- Use EXECUTE privileges on all routines
- Use USAGE privilege on distinct types
- Use BIND on any plan and COPY on any package
- Use privileges over views that are owned by others
- Set the current SQL ID to any valid value
- Create and drop synonyms and views for other IDs on any table
- Use any valid value for OWNER in BIND or REBIND
- Drop database DSNDB07

An ID with the SYSADM authority can also perform the following actions but cannot grant to other IDs the privileges to perform them:

- Drop or alter any DB2 object, except system databases
- Issue a COMMENT ON statement for any table, view, index, column, package, plan
- Issue a LABEL ON statement for any table or view
- Terminate any utility job

Although an ID with the SYSADM authority cannot grant the preceding privileges explicitly, it can accomplish this goal by granting to other IDs the SYSADM

authority. 

SYSCTRL

The SYSCTRL authority is designed for administering a system that contains sensitive data. A user with the SYSCTRL authority has nearly complete control of the DB2 subsystem. However, that user cannot access user data directly unless the privilege to do so is explicitly granted.

 A user with the SYSCTRL authority can perform the following actions:

- Act as installation SYSOPR (when the catalog is available) or DBCTRL over any database
- Run any allowable utility on any database
- Issue a COMMENT ON, LABEL ON, or LOCK TABLE statement for any table
- Create a view on any catalog table for itself or for other IDs
- Create tables and aliases for itself or for others IDs
- Bind a new plan or package and name any ID as the owner of the plan or package

Without additional privileges, A user with the SYSCTRL authority cannot perform the following actions:

- Execute SQL statements that change data in any user tables or views
- Run plans or packages
- Set the current SQL ID to a value that is not one of its primary or secondary IDs
- Start or stop the database that contains the application registration table (ART) and the object registration table (ORT)
- Act fully as SYSADM or as DBADM over any database
- Access DB2 when the subsystem is started with ACCESS(MAINT)

The SYSCTRL authority is intended to separate system control functions from administrative functions. However, SYSCTRL is not a complete solution for a high-security system. If any plans have their EXECUTE privilege granted to PUBLIC, an ID with the SYSCTRL authority can grant itself the SYSADM authority. The only control over such actions is to audit the activity of IDs with high levels of authority. 

Installation SYSOPR

The installation SYSOPR authority is assigned to one or two IDs when DB2 is installed; it cannot be assigned to a role. These IDs have all the privileges of the SYSOPR authority.

 No IDs can revoke the installation SYSOPR authority; you can remove it only by changing the module that contains the subsystem initialization parameters (typically DSNZPARM).

In addition, the installation SYSOPR authority is not recorded in the DB2 catalog. Therefore, the catalog does not need to be available to check the installation SYSOPR authority.

IDs with the installation SYSOPR authority can perform the following actions:

- Access DB2 when the subsystem is started with ACCESS(MAINT).
- Run all allowable utilities on the directory and catalog databases (DSNDB01 and DSNDB06).
- Run the REPAIR utility with the DBD statement.
- Start and stop the database that contains the application registration table (ART) and the object registration table (ORT).
- Issue dynamic SQL statements that are not controlled by the DB2 governor.
- Issue a START DATABASE command to recover objects that have LPL entries or group buffer pool RECOVERY-pending status. These IDs cannot change the access mode. 

SYSOPR

A user with the SYSOPR authority can issue all DB2 commands except ARCHIVE LOG, START DATABASE, STOP DATABASE, and RECOVER BSDS.

 In addition, that user can run the DSN1SDMP utility and terminate any utility job. With the GRANT option, that user can grant these privileges to others.



DBADM

The DBADM authority includes the DBCTRL privileges over a specific database. A user with the DBADM authority can access any tables in a specific database by using SQL statements.

 With the DBADM authority, you can also perform the following actions:

- Drop or alter any table space, table, or index in the database
- Issue a COMMENT, LABEL, or LOCK TABLE statement for any table in the database
- Issue a COMMENT statement for any index in the database

If the value of the DBADM CREATE AUTH field on the DSNTIPP installation panel is set to YES during the DB2 installation, an ID with the DBADM authority can create the following objects:

- A view for another ID. The view must be based on at least one table, and that table must be in the database under DBADM authority.
- An alias for another ID on any table in the database.

An ID with DBADM authority on one database can create a view on tables and views in that database and other databases only if the ID has all the privileges that are required to create the view. For example, an ID with DBADM authority cannot create a view on a view that is owned by another ID.

If a user has the DBADM authority with the GRANT option, that user can grant these privileges to others. 

DBCTRL

The DBCTRL authority includes the DBMAINT privileges on a specific database. A user with the DBCTRL authority can run utilities that can change the data.

 If the value of the DBADM CREATE AUTH field on the DSNTIPP installation panel is set to YES during the DB2 installation, an ID with DBCTRL authority can create an alias for another user ID on any table in the database.

If a user has the DBCTRL authority with the GRANT option, that user can grant those privileges to others. 

DBMAINT

A user with the DBMAINT authority can grant to an ID the privileges on a specific database.

 With the DBMAINT authority, that user can perform the following actions within that database:

- Create objects
- Run utilities that don't change data
- Issue commands
- Terminate all utilities on the database except DIAGNOSE, REPORT, and STOSPACE

If a user has the DBMAINT authority with the GRANT option, that user can grant those privileges to others. 

PACKADM

The PACKADM authority has the package privileges on all packages in specific collections and the CREATE IN privilege on these collections.

 If the BIND NEW PACKAGE installation option is set to BIND, the PACKADM authority also has the privilege to add new packages or new versions of existing packages.

If a user has the PACKADM authority with the GRANT option, that user can grant those privileges to others. 

Utility authorities for DB2 catalog and directory

The DB2 catalog is in the DSNDB06 database. Authorities that are granted on DSNDB06 also cover database DSNDB01, which contains the DB2 directory.

GUPI An ID with the SYSCTRL or SYSADM authority can control access to the catalog in the following ways:

- By granting privileges or authorities on that database or on its tables or views
- By binding plans or packages that access the catalog

An ID with the SYSADM authority can control access to the directory by granting privileges to run utilities on DSNDB06, but that ID cannot grant privileges on DSNDB01 directly.

The following table shows the utilities IDs with different authorities that can run on the DSNDB01 and DSNDB06 databases. Do not run REPAIR DBD against DSNDB01 and DSNDB06 because they are system databases; you will receive a system restriction violation message if you do. Also, you can use the LOAD utility to add lines to SYSIBM.SYSSTRINGS, but you cannot run it on other DSNDB01 or DSNDB06 tables.

Table 48. Utility privileges on the DB2 catalog and directory

Utilities	Authorities		
	Installation SYSOPR, SYSCTRL, SYSADM, Installation SYSADM	DBCTRL, DBADM on DSNDB06	DBMAINT on DSNDB06
LOAD	No	No	No
REPAIR DBD	No	No	No
CHECK DATA	Yes	No	No
CHECK LOB	Yes	No	No
REORG TABLESPACE	Yes	No	No
STOSPACE	Yes	No	No
REBUILD INDEX	Yes	Yes	No
RECOVER	Yes	Yes	No
REORG INDEX	Yes	Yes	No
REPAIR	Yes	Yes	No
REPORT	Yes	Yes	No
CHECK INDEX	Yes	Yes	Yes
COPY	Yes	Yes	Yes
MERGECOPY	Yes	Yes	Yes
MODIFY	Yes	Yes	Yes
QUIESCE	Yes	Yes	Yes
RUNSTATS	Yes	Yes	Yes

GUPI

Privileges by authorization ID and authority

When a process gains access to DB2, it has a primary authorization ID, one or more secondary authorization IDs, an SQL ID, and perhaps a specific role if running in a trusted context. A plan or package also has an owner that can be an authorization ID or role. To be able to perform certain actions, a single ID or role must hold the required privileges. To perform other actions, a set of IDs or roles must hold the required privileges.

GUIP For better performance, consider limiting the number of secondary IDs in your catalog table. A process can have up to 1012 secondary IDs. The more secondary IDs that must be checked, the longer the check takes. Also, make sure that the role and the current SQL ID have the necessary privileges for dynamic SQL statements. Because the role and the current SQL ID are checked first, the operation is fastest if they have all the necessary privileges. **GUIP**

Privileges required for common job roles and tasks

The labels of the administrative authorities often suggest the job roles and responsibilities of the users who are empowered with the authorities.

GUIP For example, you might expect a system administrator to have the SYSADM authority. However, some organizations do not divide job responsibilities in the same way. The following table lists some of common job roles, the tasks that usually accompany them, and the DB2 authorities or privileges that are needed to perform those tasks.

Table 49. Required privileges for common jobs and tasks

Job title	Tasks	Required privileges
System operator	Issues commands to: <ul style="list-style-type: none"> • Start and stop DB2 • Control traces • Display databases and threads • Recover indoubt threads • Start, stop, and display routines 	SYSOPR authority
System administrator	Performs emergency backup, with access to all data.	SYSADM authority
Security administrator	Authorizes other users, for some or all levels below.	SYSCTRL authority
Database administrator	Designs, creates, loads, reorganizes, and monitors databases, tables, and other objects.	<ul style="list-style-type: none"> • DBADM authority on a database. The DBADM authority on DSNDB04 allows you access to objects in all implicitly created databases. • Use of storage groups and buffer pools
System programmer	<ul style="list-style-type: none"> • Installs a DB2 subsystem. • Recovers the DB2 catalog. • Repairs data. 	Installation SYSADM, which is assigned when DB2 is installed. (Consider securing the password for an ID with this authority so that the authority is available only when needed.)

Table 49. Required privileges for common jobs and tasks (continued)

Job title	Tasks	Required privileges
Application programmer	<ul style="list-style-type: none"> • Develops and tests DB2 application programs. • Creates tables of test data. 	<ul style="list-style-type: none"> • BIND on existing plans or packages, or BINDADD • CREATE IN on some collections • Privileges on some objects • CREATETAB on some database, with a default table space provided • CREATETAB on DSNDB04. It enables you to create tables in DSNDB04 and all implicitly created databases
Production binder	Binds, rebinds, and frees application packages and plans	A ROLE, secondary ID, or RACF group of which the binder has BINDADD, CREATE IN on collections privileges required by application packages and plans
Package administrator	Manages collections and the packages in them, and delegates the responsibilities.	PACKADM authority
User analyst	Defines the data requirements for an application program, by examining the DB2 catalog.	<ul style="list-style-type: none"> • SELECT on the SYSTABLES, SYSCOLUMNS, and SYSVIEWS catalog tables • CREATETMTAB system privilege to create temporary tables
Program end user	Executes an application program.	EXECUTE for the application plan
Information center consultant	<ul style="list-style-type: none"> • Defines the data requirements for a query user. • Provides the data by creating tables and views, loading tables, and granting access. 	<ul style="list-style-type: none"> • DBADM authority over some databases • SELECT on the SYSTABLES, SYSCOLUMNS, and SYSVIEWS catalog tables
Query user	<ul style="list-style-type: none"> • Issues SQL statements to retrieve, add, or change data. • Saves results as tables or in global temporary tables. 	<ul style="list-style-type: none"> • SELECT, INSERT, UPDATE, DELETE on some tables and views • CREATETAB, to create tables in other than the default database • CREATETAB, to create tables in the implicitly created database • CREATETMTAB system privilege to create temporary tables • SELECT on SYSTABLES, SYSCOLUMNS, or views thereof. QMF™ provides the views.

GUPI

Checking access authorization for data definition statements

DB2 checks for the necessary authorization privileges and authorities when you request access to DB2 objects through data definition statements.

GUPI

DB2 determines the access to the following objects at both bind and run time:

- Alias
- Table
- Explicitly created auxiliary table
- Explicitly created table space

- Explicitly created index
- Storage group
- Database

DB2 determines access to the following objects only at run time:

- Buffer pool that is involved with an implicitly created table space
- Buffer pool and storage group that are involved with an implicitly created auxiliary index and LOB table space
- Buffer pool and storage group that are involved with implicitly created XML indexes and XML table space
- Trigger
- Function
- Procedure
- Sequence
- View
- Trusted context
- JAR
- Role
- Distinct type
- Table, buffer pool, and storage group for an implicitly created unique key index, primary key index, or ROWID index. 

Privileges required for handling plans and packages

An ID, or a role if running in a trusted context, needs specific privileges to perform actions on plans and packages.

 The following table lists the IDs and describes the privileges that they need for performing each type of plan or package operation. A user-defined function, stored procedure, or trigger package does not need to be included in a package list. A trigger package cannot be deleted by FREE PACKAGE or DROP PACKAGE. The DROP TRIGGER statement must be used to delete the trigger package.

Table 50. Required privileges for basic operations on plans and packages

Operation	ID or role	Required privileges
Execute a plan	Primary ID, any secondary ID, or role	Any of the following privileges: <ul style="list-style-type: none"> • Ownership of the plan • EXECUTE privilege for the plan • SYSADM authority
Bind embedded SQL statements, for any bind operation	Plan or package owner	Any of the following privileges: <ul style="list-style-type: none"> • Applicable privileges required by the statements • Authorities that include the privileges • Ownership that implicitly includes the privileges <p>Object names include the value of QUALIFIER, where it applies.</p>

Table 50. Required privileges for basic operations on plans and packages (continued)

Operation	ID or role	Required privileges
Include package in PKLIST ¹	Plan owner	Any of the following privileges: <ul style="list-style-type: none"> • Ownership of the package • EXECUTE privilege for the package • PACKADM authority over the package collection • SYSADM authority
BIND a new plan using the default owner or primary authorization ID	Primary ID or role	BINDADD privilege, or SYSCTRL or SYSADM authority
BIND a new package using the default owner or primary authorization ID	Primary ID or role	If the value of the field BIND NEW PACKAGE on installation panel DSNTIPP is BIND, any of the following privileges: <ul style="list-style-type: none"> • BIND privilege and CREATE IN privilege for the collection • PACKADM authority for the collection • SYSADM or SYSCTRL authority If BIND NEW PACKAGE is BINDADD, any of the following privileges: <ul style="list-style-type: none"> • BINDADD privilege and either the CREATE IN or PACKADM privilege for the collection • SYSADM or SYSCTRL authority
BIND REPLACE or REBIND for a plan or package using the default owner or primary authorization ID	Primary ID, any secondary ID, or role	Any of the following privileges: <ul style="list-style-type: none"> • Ownership of the plan or package • BIND privilege for the plan or package • BINDAGENT from the plan or package owner • PACKADM authority for the collection (for a package only) • SYSADM or SYSCTRL authority.
BIND a new version of a package, with default owner	Primary ID or role	If BIND NEW PACKAGE is BIND, any of the following privileges: <ul style="list-style-type: none"> • BIND privilege on the package or collection • BINDADD privilege and CREATE IN privilege for the collection • PACKADM authority for the collection • SYSADM or SYSCTRL authority If BIND NEW PACKAGE is BINDADD, any of the following: <ul style="list-style-type: none"> • BINDADD privilege and either the CREATE IN or PACKADM privilege for the collection • SYSADM or SYSCTRL authority
FREE or DROP a package ²	Primary ID, any secondary ID, or role	Any of the following privileges: <ul style="list-style-type: none"> • Ownership of the package • BINDAGENT from the package owner • PACKADM authority for the collection • SYSADM or SYSCTRL authority
COPY a package	Primary ID, any secondary ID, or role	Any of the following: <ul style="list-style-type: none"> • Ownership of the package • COPY privilege for the package • BINDAGENT from the package owner • PACKADM authority for the collection • SYSADM or SYSCTRL authority

Table 50. Required privileges for basic operations on plans and packages (continued)

Operation	ID or role	Required privileges
FREE a plan	Primary ID, any secondary ID, or role	Any of the following privileges: <ul style="list-style-type: none"> • Ownership of the plan • BIND privilege for the plan • BINDAGENT from the plan owner • SYSADM or SYSCTRL authority
Name a new OWNER other than the primary authorization ID for any bind operation	Primary ID, any secondary ID, or role	Any of the following privileges: <ul style="list-style-type: none"> • New owner is the primary or any secondary ID • BINDAGENT from the new owner • SYSADM or SYSCTRL authority

GUIP

Privileges required for using dynamic SQL statements

An ID needs specific privileges to issue dynamic SQL statements.

GUIP The following table lists the IDs and describes the privileges that they need for issuing each type of SQL statement:

Table 51. Required privileges for basic operations on dynamic SQL statements

Operation	ID or role	Required privileges
GRANT	Current SQL ID or role	Any of the following privileges: <ul style="list-style-type: none"> • The applicable privilege with the grant option • An authority that includes the privilege, with the grant option (not needed for SYSADM or SYSCTRL) • Ownership that implicitly includes the privilege
REVOKE	Current SQL ID or role	Must either have granted the privilege that is being revoked, or hold SYSCTRL or SYSADM authority.
CREATE, for unqualified object name	Current SQL ID or role	Applicable table, database, or schema privilege
Qualify name of object created	ID or role named as owner	Applicable table or database privilege. If not in a trusted context defined with the ROLE AS OBJECT OWNER clause, the current SQL ID has the SYSADM authority, the qualifier can be any ID at all, and the ID does not need to have any privilege. If in a trusted context defined with the ROLE AS OBJECT OWNER clause, the role requires the CREATEIN privilege on the qualifier. In this case the role is the owner of the object to be created.
Other dynamic SQL if DYNAMICRULES uses run behavior	All primary IDs, role, secondary IDs, and the current SQL ID together	As required by the statement. Unqualified object names are qualified by the value of the special register CURRENT SQLID.

Table 51. Required privileges for basic operations on dynamic SQL statements (continued)

Operation	ID or role	Required privileges
Other dynamic SQL if DYNAMICRULES uses bind behavior	Plan or package owner	As required by the statement. DYNAMICRULES behavior determines how unqualified object names are qualified.
Other dynamic SQL if DYNAMICRULES uses define behavior	Function or procedure owner	As required by the statement. DYNAMICRULES behavior determines how unqualified object names are qualified.
Other dynamic SQL if DYNAMICRULES uses invoke behavior	ID of the SQL statement that invoked the function or procedure or role	As required by the statement. DYNAMICRULES behavior determines how unqualified object names are qualified.



Managing explicit privileges

You can use the SQL GRANT or REVOKE statements to grant and remove privileges if you enable authorization checking during the DB2 installation. You can grant or revoke privileges to and from authorization IDs or roles if running in a trusted context. You can revoke only privileges that are explicitly granted.

You can grant privileges in the following ways:

- Grant a specific privilege on one object in a single statement
- Grant a list of privileges
- Grant privileges on a list of objects
- Grant ALL, for all the privileges of accessing a single table, or for all privileges that are associated with a specific package

If you grant privileges on a procedure or a package, all versions of that procedure or package have those privileges. DB2 ignores duplicate grants and keeps only one record of a grant in the catalog. The suppression of duplicate records applies not only to explicit grants, but also to the implicit grants of privileges that are made when a package is created.

For example, suppose that Susan grants the SELECT privilege on the EMP table to Ray. Then suppose that Susan grants the same privilege to Ray again, without revoking the first grant. When Susan issues the second grant, DB2 ignores it and maintains the record of the first grant in the catalog.

Database privileges that are granted on DSNDB04 apply to all implicitly created databases. For example, if you have the DBADM authority on DSNDB04, you can select data from any table in any implicitly created database. If you have the STOPDB privilege on DSNDB04, you can stop any implicitly created database. However, you cannot grant the same authorities or privileges to others on any implicitly created database.

Granting privileges to a role

You can grant privileges to a role by using the GRANT statement. You can associate primary authorization IDs with a role in the definition of the trusted context and then use the GRANT statement with the ROLE option to grant privileges to the role.

You can improve access control by granting privileges to roles. When you grant certain privileges to a role, you make those privileges available to all users that are associated with the role in the specific trusted context.

You can also simplify the administration of granting privileges by using roles rather than individual authorization IDs. To make a role a grantor, you need to specify the ROLE AS OBJECT OWNER clause when you define the trusted context. For a static GRANT statement, the grantor is the role that owns the plan or package. For a dynamic GRANT statement, the role for the primary authorization ID that executes the GRANT statement becomes the grantor.

Granting privileges to the PUBLIC ID

You can grant privileges to the PUBLIC ID. When you grant privileges to PUBLIC, the privileges become available to all IDs at the local DB2 site, including the owner IDs of packages that are bound from a remote location.

When you grant any privilege to the PUBLIC ID, DB2 catalog tables record the grantee of the privilege as PUBLIC. DB2 also grants the following implicit table privileges to PUBLIC for declared temporary tables:

- All table privileges on the tables and the authority to drop the tables
- The CREATETAB privilege to define temporary tables in the work file database
- The USE privilege to use the table spaces in the work file database

You do not need any additional privileges to access the work file database and the temporary tables that are in it. You cannot grant or revoke table privileges for temporary tables. The DB2 catalog does not record these implicit privileges for declared temporary tables.

Because PUBLIC is a special identifier that is used by DB2 internally, you should not use PUBLIC as a primary ID or secondary ID. When a privilege is revoked from PUBLIC, authorization IDs to which the privilege was specifically granted retain the privilege.

However, when an ID uses PUBLIC privileges to perform actions, the actions and the resulting objects depend on the privileges that are currently in effect for PUBLIC. If PUBLIC loses a privilege, objects that are created with that privilege can be dropped or invalidated. The following examples demonstrate how certain objects depend on PUBLIC not losing its privileges.

Example: Suppose that Juan has the ID USER1 and that Meg has the ID USER2. Juan creates a table TAB1 and grants ALL PRIVILEGES on it to PUBLIC. Juan does not explicitly grant any privileges on the table to Meg's ID, USER2. Using the PUBLIC privileges, Meg creates a view on TAB1. Because the ID USER2 requires the SELECT privilege on TAB1 to create the view, the view is dropped if PUBLIC loses the privilege.

Example: Suppose that Kim has the ID USER3. Kim binds a plan and names it PLAN1. PLAN1 contains a program that refers to TAB1 from the previous

example. PLAN1 is not valid unless all of the proper privileges are held on objects to which the plan refers, including TAB1. Although Kim does not have any privileges on TAB1, Kim can bind the plan by using the PUBLIC privileges on TAB1. However, if PUBLIC loses its privilege, the plan is invalidated.

Granting privileges to remote users

A query that arrives at your local DB2 through the distributed data facility (DDF) is accompanied by an authorization ID. After connection processing, the ID can be translated to another value and associated with secondary authorization IDs. DB2 also uses the ID to determine if the connection is associated with a trusted context.

As the end result of these processes, the remote query is associated with a set of IDs that is known to your local DB2 subsystem. You assign privileges to these IDs in the same way that you assign privileges to IDs that are associated with local queries.

GUPI You can issue the following command to grant SELECT, INSERT, UPDATE, and DELETE table privileges to any ID anywhere that uses DB2 private protocol access to your data:

```
GRANT privilege TO PUBLIC AT ALL LOCATIONS;
```

Some differences exist in the privileges for a query that uses system-directed access:

- Although the query can use privileges granted TO PUBLIC AT ALL LOCATIONS, it cannot use privileges granted TO PUBLIC.
- The query can exercise only the SELECT, INSERT, UPDATE, and DELETE privileges at the remote location.

These restrictions do not apply to queries that are run by a package that is bound at your local DB2 subsystem. Those queries can use any privilege that is granted to their associated IDs or any privilege that is granted to PUBLIC. **GUPI**

Granting privileges through views

Any of the table privileges except ALTER, REFERENCES, TRIGGER, and INDEX can be granted on a view. By creating a view and granting privileges on it, you can give an ID access to only a specific combination of data. This capability is sometimes called *field-level access control* or *field-level sensitivity*.

GUPI For example, suppose that you want the ID MATH110 to be able to extract the following column data from the sample employee table for statistical investigation: HIREDATE, JOB, EDLEVEL, SEX, SALARY, BONUS, and COMM for DSN8910.EMP. However, you want to impose the following restrictions:

- No access to employee names or identification numbers
- No access to data for employees hired before 1996
- No access to data for employees with an education level less than 13
- No access to data for employees whose job is MANAGER or PRES

You can create and name a view that shows exactly that combination of data. Perform the following steps to grant privileges to the view that you create:

1. Issue the following CREATE statement to create the desired view:

```
CREATE VIEW SALARIES AS
  SELECT HIREDATE, JOB, EDLEVEL, SEX, SALARY, BONUS, COMM
  FROM DSN8910.EMP
```

```

WHERE HIREDATE > '1995-12-31' AND
      EDLEVEL >= 13 AND
      JOB <> 'MANAGER' AND
      JOB <> 'PRES';

```

- Issue the following statement to grant the SELECT privilege on the SALARIES view to MATH110:

```
GRANT SELECT ON SALARIES TO MATH110;
```

After you grant the privilege, MATH110 can execute SELECT statements on the restricted set of data only. Alternatively, you can give an ID access to only a specific combination of data by using multilevel security with row-level

granularity. **GUIP**

Granting privileges with the GRANT statement

You can assign privileges to an ID or a role by issuing the GRANT statement.

Suppose that the Spiffy Computer Company wants to create a database to hold information that is usually posted on hallway bulletin boards. For example, the database might hold notices of upcoming holidays and bowling scores.

To create and maintain the tables and programs that are needed for this application, the Spiffy Computer Company develops the security plan shown in the following diagram.

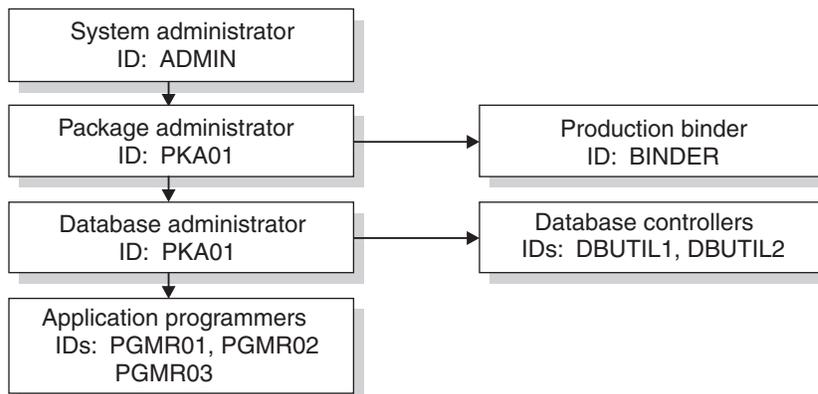


Figure 17. Security plan for the Spiffy Computer Company

The Spiffy Computer Company's system of privileges and authorities associates each role with an authorization ID. For example, the System Administrator role has the ADMIN authorization ID.

```

User ID: ADMIN
Authority: SYSADM
Privileges: Ownership of SG1

```

GUIP The system administrator uses the ADMIN authorization ID, which has the SYSADM authority, to create a storage group (SG1) and to issue the following statements:

- GRANT PACKADM ON COLLECTION BOWLS TO PKA01 WITH GRANT OPTION;

This statement grants to PKA01 the CREATE IN privilege on the collection BOWLS and BIND, EXECUTE, and COPY privileges on all packages in the collection. Because ADMIN used the WITH GRANT OPTION clause, PKA01 can grant those privileges to others.

2. GRANT CREATEDBA TO DBA01;

This statement grants to DBA01 the privilege to create a database and to have DBADM authority over that database.

3. GRANT USE OF STOGROUP SG1 TO DBA01 WITH GRANT OPTION;

This statement allows DBA01 to use storage group SG1 and to grant that privilege to others.

4. GRANT USE OF BUFFERPOOL BP0, BP1 TO DBA01 WITH GRANT OPTION;

This statement allows DBA01 to use buffer pools BP0 and BP1 and to grant that privilege to others.

5. GRANT CREATE IN COLLECTION DSN8CC91 TO ROLE ROLE1;

This statement grants to ROLE1 the privilege to create new packages in collections DSN8CC91. 

User ID: PKA01
Authority: PACKADM over the collection BOWLS

The package administrator, PKA01, controls the binding of packages into collections. PKA01 can use the CREATE IN privilege on the collection BOWLS and the BIND, EXECUTE, and COPY privileges on all packages in the collection. PKA01 also has the authority to grant these privileges to others.

User ID: DBA01
Authority: DBADM over DB1
Privileges: CREATEDBA
Use of SG1 with GRANT
Use of BP0 and BP1 with GRANT
Ownership of DB1

The database administrator, DBA01, using the CREATEDBA privilege, creates the database DB1. When DBA01 creates DB1, DBA01 automatically has DBADM authority over the database.

User ID: DBUTIL1, DBUTIL2
Authority: DBCTRL over DB1

The database administrator at Spiffy Computer Company wants help with running the COPY and RECOVER utilities. Therefore DBA01 grants DBCTRL authority over database DB1 to DBUTIL1 and DBUTIL2.

 To grant DBCTRL authority, the database administrator issues the following statement:

```
GRANT DBCTRL ON DATABASE DB1 TO DBUTIL1, DBUTIL2;
```

Granting privileges to secondary IDs

The Spiffy Computer Company uses RACF to manage external access to DB2. Therefore, Spiffy can use secondary authorization IDs to define user groups and associate primary authorization IDs with those user groups.

The primary authorization IDs are the RACF user IDs. The secondary authorization IDs are the names of the groups with which the primary IDs are associated.

Spiffy can grant DB2 privileges to primary IDs indirectly, by granting privileges to secondary IDs that are associated with the primary IDs. This approach associates privileges with a *functional ID* rather than an *individual ID*. Functional IDs, also called group IDs, are granted privileges based on the function that certain job roles serve in the system. Multiple primary IDs can be associated with a functional ID and receive the privileges that are granted to that functional ID. In contrast, individual IDs are connected to specific people. Their privileges need to be updated as people join the company, leave the company, or serve different roles within the company. Functional IDs have the following advantages:

- Functional IDs reduce system maintenance because they are more permanent than individual IDs. Individual IDs require frequent updates, but each functional ID can remain in place until Spiffy redesigns its procedures.

Example: Suppose that Joe retires from the Spiffy Computer Company. Joe is replaced by Mary. If Joe's privileges are associated with functional ID DEPT4, those privileges are maintained in the system even after Joe's individual ID is removed from the system. When Mary enters the system, she will have all of Joe's privileges after her ID is associated with the functional ID DEPT4.

- Functional IDs reduce the number of grants that are needed because functional IDs often represent groups of individuals.
- Functional IDs reduce the need to revoke privileges and re-create objects when they change ownership.

Example: Suppose that Bob changes jobs within the Spiffy Computer Company. Bob's individual ID has privileges on many objects in the system and owns three databases. When Bob's job changes, he no longer needs privileges over these objects or ownership of these databases. Because Bob's privileges are associated with his individual ID, a system administrator needs to revoke all of Bob's privileges on objects and drop and re-create Bob's databases with a new owner. If Bob received privileges by association with a functional ID, the system administrator would only need to remove Bob's association with the functional ID.

Granting privileges to user groups

You can simplify the assignment and management of privileges by creating user groups and granting privileges to the groups. In this way, you can simply assign the same set of privileges to all the users of a given group at the same time.

Suppose that the database administrator at Spiffy wants several employees in the Software Support department to create tables in the DB1 database. The database administrator creates DEVGROU as a RACF group ID for this purpose. To simplify the process, the database administrator decides that each CREATE TABLE statement should implicitly create a unique table space for the table. Hence, DEVGROU needs the CREATETAB privilege, the CREATETS privilege, the privilege to use the SG1 storage group and, the privilege to use one of the buffer

pools, BP0, for the implicitly created table spaces. The following diagram shows this group and their privileges:

RACF Group ID: DEVGROU Privileges: (All without GRANT) CREATETAB on DB1 CREATETS on DB1 Use of SG1 Use of BP0
--

GUIP The database administrator, DBA01, owns database DB1 and has the privileges to use storage group SG1 and buffer pool BP0. The database administrator holds both of these privileges with the GRANT option. The database administrator issues the following statements:

1. GRANT CREATETAB, CREATETS ON DATABASE DB1 TO DEVGROU;
2. GRANT USE OF STOGROUP SG1 TO DEVGROU;
3. GRANT USE OF BUFFERPOOL BP0 TO DEVGROU; **GUIP**

Because the system and database administrators at Spiffy still need to control the use of those resources, the preceding statements are issued without the GRANT option.

Three programmers in the Software Support department write and test a new program, PROGRAM1. Their IDs are PGMR01, PGMR02, and PGMR03. Each programmer needs to create test tables, use the SG1 storage group, and use one of the buffer pools. All of those resources are controlled by DEVGROU, which is a RACF group ID.

Therefore, granting privileges over those resources specifically to PGMR01, PGMR02, and PGMR03 is unnecessary. Each ID should be associated with the RACF group DEVGROU and receive the privileges that are associated with that functional ID. The following diagram shows the DEVGROU and its members:

RACF group ID: DEVGROU Group members: PGMR01, PGMR02, PGMR03

The security administrator connects as many members as desired to the group DEVGROU. Each member can exercise all the privileges that are granted to the group ID.

Granting privileges for binding plans

Three programmers can share the tasks that are done by the DEVGROU ID. Someone creates a test table, DEVGROU.T1, in database DB1 and loads it with test data. Someone writes a program, PROGRAM1, to display bowling scores that are contained in T1. Someone must bind the plan and packages that accompany the program.

GUIP Binding requires an additional privilege. ADMIN, who has SYSADM authority, grants the required privilege by issuing the following statement:
GRANT BINDADD TO DEVGROU;

GUIP

With that privilege, any member of the RACF group DEVGROUP can bind plans and packages that are to be owned by DEVGROUP. Any member of the group can rebind a plan or package that is owned by DEVGROUP. The following diagram shows the BINDADD privilege granted to the group:

```
RACF group ID: DEVGROUP
Privilege: BINDADD
```

The Software Support department proceeds to create and test the program.

Granting privileges for rebinding plans and packages

Spiffy has a different set of tables, which contain actual data that is owned by the ROLE PRODCN. PROGRAM1 was written with unqualified table names.

For example, table T1 was referred to as simply T1, not DEVGROUP.T1. The new packages and plan must refer to table PRODCN.T1. To move the completed program into production, someone must perform the following steps:

- Rebind the application plan with the owner PRODCN.
- Rebind the packages into the collection BOWLS, again with the owner PRODCN.

Spiffy gives that job to a production binder with the ID BINDER. BINDER needs privileges to bind a plan or package that DEVGROUP owns, to bind a plan or package with OWNER (PRODCN), and to add a package to the collection BOWLS. BINDER acquires these abilities through its RACF DEVGROUP group and ROLE PRODCN. ROLE PRODCN needs to have all the necessary privileges.

GUIP

Suppose that ID BINDER has ROLE PRODCN when binding in a trusted context and that ROLE PRODCN has the following privileges:

```
DB2 Role: PRODCN
Privileges: BINDADD
           CREATE IN collection BOWLS
           Privileges on SQL objects referenced in application
```

BINDER can bind plans and packages for owner ROLE PRODCN because it performs binds in a trusted context with ROLE PRODCN.

PACKADM, the package administrator for BOWLS, can grant the CREATE privilege with the following statement:

```
GRANT CREATE ON COLLECTION BOWLS TO ROLE PRODCN;
```

With the plan in place, the database administrator at Spiffy wants to make the PROGRAM1 plan available to all employees by issuing the following statement:

```
GRANT EXECUTE ON PLAN PROGRAM1 TO PUBLIC;
```

More than one ID has the authority or privileges that are necessary to issue this statement. For example, ADMIN has SYSADM authority and can grant the EXECUTE privilege. Also, any ID in a trusted context with ROLE PRODCN that

owns PROGRAM1 can issue the statement. When EXECUTE is granted to PUBLIC, other IDs do not need any explicit authority on T1.

Finally, the plan to display bowling scores at Spiffy Computer Company is complete. The production plan, PROGRAM1, is created, and all IDs have the authority to execute the plan. 

Granting privileges for accessing distributed data

Some time after the system and database administrators at Spiffy install their security plan, the president of Spiffy Computer Company tells them that other applications on other systems must connect to the local DB2 subsystem. She wants people at every location to be able to access bowling scores through PROGRAM1 on the local subsystem.

 The administrators perform the following steps to enable access from all Spiffy locations:

1. Add a CONNECT statement to the program, naming the location at which table PRODCNTN.T1 resides. In this case, the table and the package reside at only the central location.
2. Issue the following statement so that PKA01, who has PACKADM authority, can grant the required privileges to DEVGROUPE:

```
GRANT CREATE IN COLLECTION BOWLS TO DEVGROUPE;
```
3. Bind the SQL statements in PROGRAM1 as a package.
4. Bind the SQL statements in PROGRAM1 as a package by the package owner:

```
GRANT EXECUTE ON PACKAGE PROGRAM1 TO PUBLIC;
```

Any system that is connected to the original DB2 location can run PROGRAM1 and execute the package by using DRDA[®] access. However, if the remote system is another DB2, a plan must be bound there that includes the package in its package list. 

Revoking privileges with the REVOKE statement

You can use the REVOKE statement to remove the privileges that you explicitly grant to an ID or a role.

 For example, you can revoke the privilege that you grant to an ID by issuing the following statement:

```
REVOKE authorization-specification FROM auth-id
```

Generally, you can revoke only the privileges that you grant. If you revoke privileges on a procedure or package, the privileges are revoked from all versions of that procedure or package.

However, an ID with the SYSADM or SYSCTRL authority can revoke a privilege that has been granted by another ID with the following statement:

```
REVOKE authorization-specification FROM auth-id BY auth-id
```

The BY clause specifies the authorization ID that originally granted the privilege. If two or more grantors grant the same privilege to an ID, executing a single REVOKE statement does not remove the privilege. To remove it, each grant of the privilege must be revoked. 

The WITH GRANT OPTION clause of the GRANT statement allows an ID to pass the granted privilege to others. If the privilege is removed from the ID, its deletion can cascade to others, with side effects that are not immediately evident. For example, when a privilege is removed from authorization ID X, it is also removed from any ID to which X granted it, unless that ID also has the privilege from some other source.

Example: Suppose that DBA01 grants DBCTRL authority with the GRANT option on database DB1 to DBUTIL1. Then DBUTIL1 grants the CREATETAB privilege on DB1 to PGMR01. If DBA01 revokes DBCTRL from DBUTIL1, PGMR01 loses the CREATETAB privilege. If PGMR01 also granted the CREATETAB privilege to OPER1 and OPER2, they also lose the privilege.

Example: Suppose that PGMR01 from the preceding example created table T1 while holding the CREATETAB privilege. If PGMR01 loses the CREATETAB privilege, table T1 is not dropped, and the privileges that PGMR01 has as owner of the table are not deleted. Furthermore, the privileges that PGMR01 grants on T1 are not deleted. For example, PGMR01 can grant SELECT on T1 to OPER1 as long as PGMR01 owns the table. Even when the privilege to create the table is revoked, the table remains, the privilege remains, and OPER1 can still access T1.

Example: Consider the following REVOKE scenario:

1. Grant #1: SYSADM, SA01, grants SELECT on TABLE1 to USER01 with the GRANT option.
2. Grant #2: USER01 grants SELECT on TABLE1 to USER02 with the GRANT option.
3. Grant #3: USER02 grants SELECT on TABLE1 back to SA01.
4. USER02 then revokes SELECT on TABLE1 from SA01.

The cascade REVOKE process of Grant #3 determines if SA01 granted SELECT to anyone else. It locates Grant #1. Because SA01 did not have SELECT from any other source, this grant is revoked. The cascade REVOKE process then locates Grant #2 and revokes it for the same reason. In this scenario, the single REVOKE action by USER02 triggers and results in the cascade removal of all the grants even though SA01 has the SYSADM authority. The SYSADM authority is not considered.

Revoking privileges granted by multiple IDs

You can revoke the same privileges that are granted to multiple IDs at the same time.

GUIP Suppose that DBUTIL1 grants the CREATETAB privilege to PGMR01 and that DBUTIL2 also grants the CREATETAB privilege to PGMR01. The second grant is recorded in the catalog, with its date and time, but it has no other effect until the grant from DBUTIL1 to PGMR01 is revoked. After the first grant is revoked, DB2 must determine the authority that PGMR01 used to grant CREATETAB to OPER1. The following diagram illustrates the situation; the arrows represent the granting of the CREATETAB privilege.

1. DB2 does not cascade a revoke of the SYSADM authority from the installation SYSADM authorization IDs.

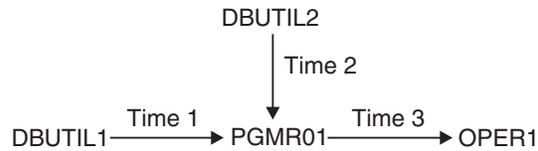


Figure 18. Authorization granted by two or more IDs

Suppose that DBUTIL1 issues the GRANT statement at Time 1 and that DBUTIL2 issues the GRANT statement at Time 2. DBUTIL1 and DBUTIL2 both use the following statement to issue the grant:

```
GRANT CREATETAB ON DATABASE DB1 TO PGMR01 WITH GRANT OPTION;
```

At Time 3, PGMR01 grants the privilege to OPER1 by using the following statement:

```
GRANT CREATETAB ON DATABASE DB1 TO OPER1;
```

After Time 3, DBUTIL1's authority is revoked, along with all of the privileges and authorities that DBUTIL1 granted. However, PGMR01 also has the CREATETAB privilege from DBUTIL2, so PGMR01 does not lose the privilege. The following criteria determine whether OPER1 loses the CREATETAB privilege when DBUTIL1's authority is revoked:

- If Time 3 *comes after* Time 2, OPER1 does not lose the privilege. The recorded dates and times show that, at Time 3, PGMR01 could have granted the privilege entirely on the basis of the privilege that was granted by DBUTIL2. That privilege was not revoked.
- If Time 3 is *precedes* Time 2, OPER1 does lose the privilege. The recorded dates and times show that, at Time 3, PGMR01 could have granted the privilege only on the basis of the privilege that was granted by DBUTIL1. That privilege was revoked, so the privileges that are dependent on it are also revoked. **GUPI**

Revoking privileges granted by all IDs

An ID with the SYSADM or SYSCTRL authority can revoke privileges that are granted by other IDs.

GUPI To revoke the CREATETAB privileges that are granted to PGMR01 on database DB1 by all IDs, use the following statement:

```
REVOKE CREATETAB ON DATABASE DB1 FROM PGMR01 BY ALL;
```

However, you might want to revoke only privileges that are granted by a specific ID. To revoke privileges that are granted by DBUTIL1 and to leave intact the same privileges if they were granted by any other ID, use the following statement:

```
REVOKE CREATETAB, CREATETS ON DATABASE DB1 FROM PGMR01 BY DBUTIL1;
```

GUPI

Revoking privileges granted by a role

You can use the REVOKE statement to revoke privileges that are granted by a role in a trusted context.

To revoke privileges that are granted by a role, you can issue the REVOKE statement in the trusted context that was defined with the ROLE AS OBJECT OWNER clause. Also, make sure the role that revokes a privilege matches the one that grants the privilege. For a static REVOKE statement, the revoker is the role

that owns the plan or package. For a dynamic REVOKE statement, the role for the primary authorization ID that executes the REVOKE statement becomes the revoker.

An authorization ID or role that has the SYSADM or SYSCTRL authority can use the BY (ROLE *role-name*) clause of the REVOKE statement to revoke privileges that are granted by a role.

Revoking all privileges from a role

You can revoke all privileges that are assigned to a role by dropping the role itself or by using the REVOKE statement.

GUIP When you attempt to drop a role, make sure that the role does not own any objects. If the role owns objects, the DROP statement is terminated. If the role does not own any objects, the role is dropped. As a result, all privileges that are held by this role are revoked, and the revocation is cascaded. **GUIP**

Revoking privileges for views

If a table privilege is revoked from the owner of a view on the table, the corresponding privilege on the view is revoked. The privilege on the view is revoked not only from the owner of the view, but also from all other IDs to which the owner granted the privilege.

GUIP If the SELECT privilege on the base table is revoked from the owner of the view, the view is dropped. However, if another grantor granted the SELECT privilege to the view owner before the view was created, the view is not dropped.

Example: Suppose that OPER2 has the SELECT and INSERT privileges on table T1 and creates a view of the table. If the INSERT privilege on T1 is revoked from OPER2, all insert privileges on the view are revoked. If the SELECT privilege on T1 is revoked from OPER2, and if OPER2 did not have the SELECT privilege from another grantor before the view was created, the view is dropped.

If a view uses a user-defined function, the view owner must have the EXECUTE privilege on the function. If the EXECUTE privilege is revoked, the revoke fails because the view is using the privilege and the RESTRICT clause prevents the revoke.

An authorization ID with the SYSADM authority can create a view for another authorization ID. In this case, the view could have both a creator and an owner. The owner is automatically given the SELECT privilege on the view. However, the privilege on the base table determines whether the view is dropped.

Example: Suppose that IDADM, with SYSADM authority, creates a view on TABLX with OPER as the owner of the view. OPER now has the SELECT privilege on the view, but not necessarily any privileges on the base table. If SYSADM is revoked from IDADM, the SELECT privilege on TABLX is gone and the view is dropped.

If one ID creates a view for another ID, the catalog table SYSIBM.SYSTABAUTH needs either one or two rows to record the associated privileges. The number of rows that DB2 uses to record the privilege is determined by the following criteria:

- If IDADM creates a view for OPER when OPER has enough privileges to create the view by itself, only one row is inserted in SYSTABAUTH. The row shows only that OPER granted the required privileges.

- If IDADM creates a view for OPER when OPER does not have enough privileges to create the view by itself, two rows are inserted in SYSTABAUTH. One row shows IDADM as GRANTOR and OPER as GRANTEE of the SELECT privilege. The other row shows any other privileges that OPER might have on the view because of privileges that are held on the base table. 

Revoking privileges for materialized query tables

If the SELECT privilege on a source table is revoked from the owner of a materialized query table, the corresponding privilege on the materialized query table is revoked. The SELECT privilege on the materialized query table is revoked not only from the owner of the materialized query table, but also from all other IDs to which the owner granted the SELECT privilege.

 If the SELECT privilege on the source table is revoked from the owner of a materialized query table, the materialized query table is dropped. However, if another grantor granted the SELECT privilege to the materialized query table owner before the materialized query table was created, the materialized query table is not dropped.

Example: Suppose that OPER7 has the SELECT privilege on table T1 and creates a materialized query table T2 by selecting from T1. If the SELECT privilege on T1 is revoked from OPER7, and if OPER7 did not have the SELECT privilege from another grantor before T2 was created, T2 is dropped.

If a materialized query table uses a user-defined function, the owner of the materialized query table must have the EXECUTE privilege on the function. If the EXECUTE privilege is revoked, the revoke fails because the materialized query table is using the privilege and the RESTRICT clause prevents the revoke. 

Revoking privileges for plans or packages

If the owner of an application plan or package loses a privilege that is required by the plan or package, and the owner does not have that privilege from another source, DB2 invalidates the plan or package.

 **Example:** Suppose that OPER2 has the SELECT and INSERT privileges on table T1 and creates a plan that uses SELECT, but not INSERT. When privileges are revoked from OPER2, the plan is affected in the following ways:

- If the SELECT privilege is revoked, DB2 invalidates the plan.
- If the INSERT privilege is revoked, the plan is unaffected.
- If the revoked privilege was EXECUTE on a user-defined function, DB2 marks the plan or package **inoperative** instead of invalid.

If authorization data is cached for a package and an ID loses EXECUTE authority on the package, that ID is removed from the cache. Similarly, if authorization data is cached for routines, a revoke or cascaded revoke of EXECUTE authority on a routine, or on all routines in a schema (schema.*), from any ID causes the ID to be removed from the cache.

If authorization data is cached for plans, a revoke of EXECUTE authority on the plan from any ID causes the authorization cache to be invalidated.

If an application is caching dynamic SQL statements, and a privilege is revoked that was needed when the statement was originally prepared and cached, that

statement is removed from the cache. Subsequent PREPARE requests for that statement do not find it in the cache and therefore execute a full PREPARE. If the plan or package is bound with KEEP DYNAMIC(YES), which means that the application does not need to explicitly re-prepare the statement after a commit operation, you might get an error on an OPEN, DESCRIBE, or EXECUTE of that statement following the next commit operation. The error can occur because a prepare operation is performed implicitly by DB2. If you no longer have sufficient authority for the prepare, the OPEN, DESCRIBE, or EXECUTE request fails. 

Revoking the SYSADM authority from IDs with the installation SYSADM authority

If you revoke the SYSADM authority from an ID with the installation SYSADM authority, DB2 does not cascade the revoke.

You can change the ID that holds the installation SYSADM authority or delete extraneous IDs with the SYSADM authority without cascading the revoke that these processes required.

Changing IDs with the installation SYSADM authority:

Perform these steps to change the ID that holds the installation SYSADM authority.

To change the ID that holds the installation SYSADM authority:

1. Select a new ID that you want it to have the installation SYSADM authority.
2. Grant SYSADM authority to the ID that you selected.
3. Revoke SYSADM authority from the ID that currently holds the installation SYSADM authority.
4. Update the SYSTEM ADMIN 1 or SYSTEM ADMIN 2 field on installation panel DSNTIPP with the new ID that you want to grant installation SYSADM authority.

Related information

Protection panel: DSNTIPP (Installation Guide)

Deleting extraneous IDs with the SYSADM authority:

You can perform the following steps to delete extraneous IDs with the SYSADM authority:

1. Write down the ID that currently holds installation SYSADM authority.
2. Change the authority of the ID that you want to delete from SYSADM to installation SYSADM. You can change the authority by updating the SYSTEM ADMIN 1 or SYSTEM ADMIN 2 field on installation panel DSNTIPP. Replace the ID that you write down in step 1 with the ID that you want to delete.
3. Revoke SYSADM authority from the ID that you want to delete.
4. Change the SYSTEM ADMIN 1 or SYSTEM ADMIN 2 field on installation panel DSNTIPP back to its original value.

Related information

Protection panel: DSNTIPP (Installation Guide)

Restrictions on privilege revocation

You can specify the RESTRICT clause of the REVOKE statement to impose limitations on privilege revocation.

GUIP Whether specified or not, the RESTRICT clause of the REVOKE statement always applies to the following objects:

- User-defined functions
- JARs (Java classes for a routine)
- Stored procedures
- Distinct types
- Sequences

When an attempt is made to revoke a privilege on one of these objects, DB2 determines whether the revokee owns an object that is dependent on the privilege. If such a dependency exists, the REVOKE statement proceeds only if the revokee also holds this privilege from another grantor or holds this privilege indirectly (such as if PUBLIC has this privilege, or if the revokee has SYSADM authority).

Example: Consider the following scenario:

1. UserA creates a user-defined function named UserA.UDFA.
2. UserA grants EXECUTE on UserA.UDFA to UserB.
3. User B then creates a user-defined function UserB.UDFB that is sourced on UserA.UDFA.

At this point, UserA attempts to revoke the EXECUTE privilege on UserA.UDFA from UserB. The revoke succeeds or fails based on the following criteria:

- If UserB has the EXECUTE privilege on UserA.UDFA only from UserA, the revoke fails with an accompanying message that indicates that a dependency on this privilege.
- If UserB has the EXECUTE privilege on UserA.UDFA from another source, directly or indirectly, the EXECUTE privilege that was granted by UserA is revoked successfully.

For distinct types, the following objects that are owned by the revokee can have dependencies:

- A table that has a column that is defined as a distinct type
- A user-defined function that has a parameter that is defined as a distinct type
- A stored procedure that has a parameter that is defined as a distinct type
- A sequence that has a parameter that is defined as a distinct type

For user-defined functions, the following objects that are owned by the revokee can have dependencies:

- Another user-defined function that is sourced on the user-defined function
- A view that uses the user-defined function
- A table that uses the user-defined function in a check constraint or user-defined default clause
- A trigger package that uses the user-defined function

For JARs (Java classes for a routine), the following objects that are owned by the revokee can have dependencies:

- A Java user-defined function that uses a JAR
- A Java stored procedure that uses a JAR

For stored procedures, a trigger package that refers to the stored procedure in a CALL statement can have dependencies.

For sequences, the following objects that are owned by the revokee can have dependencies:

- Triggers that contain NEXT VALUE or PREVIOUS VALUE expressions that specify a sequence
- Inline SQL routines that contain NEXT VALUE or PREVIOUS VALUE expressions that specify a sequence

One way to ensure that the REVOKE statement succeeds is to drop the object that has a dependency on the privilege. To determine which objects are dependent on which privileges before attempting the revoke, use the following SELECT statements.

For a distinct type:

- List all tables that are owned by the revokee USRT002 that contain columns that use the distinct type USRT001.UDT1:

```
SELECT * FROM SYSIBM.SYSCOLUMNS WHERE
  TBCreator = 'USRT002'      AND
  Typeschema = 'USRT001'    AND
  Typename = 'UDT1'         AND
  Coltype = 'DISTINCT';
```

- List the user-defined functions that are owned by the revokee USRT002 that contain a parameter that is defined as distinct type USRT001.UDT1:

```
SELECT * FROM SYSIBM.SYSPARMS WHERE
  Owner = 'USRT002'        AND
  Typeschema = 'USRT001'  AND
  Typename = 'UDT1'       AND
  Routinetype = 'F';
```

- List the stored procedures that are owned by the revokee USRT002 that contain a parameter that is defined as distinct type USRT001.UDT1:

```
SELECT * FROM SYSIBM.SYSPARMS WHERE
  Owner = 'USRT002'        AND
  Typeschema = 'USRT001'  AND
  Typename = 'UDT1'       AND
  Routinetype = 'P';
```

- List the sequences that are owned by the revokee USRT002 that contain a parameter that is defined as distinct type USRT001.UDT1:

```
SELECT SYSIBM.SYSSEQUENCES.SCHEMA, SYSIBM.SYSSEQUENCES.NAME
FROM SYSIBM.SYSSEQUENCES, SYSIBM.SYSDATATYPES WHERE
  SYSIBM.SYSSEQUENCES.DATATYPEID = SYSIBM.SYSDATATYPES.DATATYPEID AND
  SYSIBM.SYSDATATYPES.SCHEMA = 'USRT001' AND
  SYSIBM.SYSDATATYPES.NAME = 'UDT1';
```

For a user-defined function:

- List the user-defined functions that are owned by the revokee USRT002 that are sourced on user-defined function USRT001.SPECUDF1:

```
SELECT * FROM SYSIBM.SYSROUTINES WHERE
  Owner = 'USRT002'        AND
  Sourceschema = 'USRT001' AND
  Sourcespecific = 'SPECUDF1' AND
  Routinetype = 'F';
```

- List the views that are owned by the revokee USRT002 that use user-defined function USRT001.SPECUDF1:

```
SELECT * FROM SYSIBM.SYSVIEWDEP WHERE
  DCreator = 'USRT002'    AND
  BScheme = 'USRT001'    AND
  BName = 'SPECUDF1'     AND
  BType = 'F';
```

- List the tables that are owned by the revokee USRT002 that use user-defined function USRT001.A_INTEGER in a check constraint or user-defined default clause:

```
SELECT * FROM SYSIBM.SYSCONSTDEP WHERE
DTBCREATOR = 'USRT002'      AND
BSHEMA = 'USRT001'         AND
BNAME = 'A_INTEGER'        AND
BTYPE = 'F';
```

- List the trigger packages that are owned by the revokee USRT002 that use user-defined function USRT001.UDF4:

```
SELECT * FROM SYSIBM.SYSPACKDEP WHERE
DOWNER = 'USRT002'         AND
BQUALIFIER = 'USRT001'    AND
BNAME = 'UDF4'             AND
BTYPE = 'F';
```

For a JAR (Java class for a routine), list the routines that are owned by the revokee USRT002 and that use a JAR named USRT001.SPJAR:

```
SELECT * FROM SYSIBM.SYSROUTINES WHERE
OWNER = 'USRT002'          AND
JARCHEMA = 'USRT001'      AND
JAR_ID = 'SPJAR';
```

For a stored procedure that is used in a trigger package, list the trigger packages that refer to the stored procedure USRT001.WLMLOCN2 that is owned by the revokee USRT002:

```
SELECT * FROM SYSIBM.SYSPACKDEP WHERE
DOWNER = 'USRT002'        AND
BQUALIFIER = 'USRT001'    AND
BNAME = 'WLMLOCN2'        AND
BTYPE = 'O';
```

For a sequence:

- List the sequences that are owned by the revokee USRT002 and that use a trigger named USRT001.SEQ1:

```
SELECT * FROM SYSIBM.SYSPACKDEP WHERE
BNAME = 'SEQ1'
BQUALIFIER = 'USRT001'
BTYPE = 'Q'
DOWNER = 'USRT002'
DTYPE = 'T';
```

- List the sequences that are owned by the revokee USRT002 and that use an inline SQL routine named USRT001.SEQ1:

```
SELECT * FROM SYSIBM.SYSSEQUENCESDEP WHERE
DCREATOR = 'USRT002'
DTYPE = 'F'
BNAME = 'SEQ1'
BSHEMA = 'USRT001';
```

GUPI

Managing implicit privileges

You acquire privileges implicitly through ownership of objects, including ownership of plans and packages. You can control access to data by managing those privileges through object ownership and stored procedures, also known as routines.

Managing implicit privileges through object ownership

Ownership of an object carries with it a set of related privileges on the object. DB2 provides separate controls for creation and ownership of objects.

In general, when you create an object, the owner of the object can be your primary authorization ID, one of your secondary authorization IDs, or the role that you are associated with in a trusted context.

Ownership of objects with unqualified names

If an object name is unqualified, the object type and the way it is created determine its ownership.

GUIP If the name of a table, view, index, alias, or synonym is unqualified, you can establish the object's ownership in the following ways:

- If you issue the CREATE statement dynamically, perhaps using SPUFI, QMF, or some similar program, the owner of the created object is your current SQL ID. That ID must have the privileges that are needed to create the object.
- If you issue the CREATE statement statically, by running a plan or package that contains it, the ownership of the created object depends on the option that is used for the bind operation. You can bind the plan or package with either the QUALIFIER option, the OWNER option, or both.
 - If the plan or package is bound with the QUALIFIER option only, the authorization ID in the QUALIFIER option is the owner of the object. The QUALIFIER option allows the binder to name a qualifier to use for all unqualified names of tables, views, indexes, aliases, or synonyms that appear in the plan or package.
 - If the plan or package is bound with the OWNER option only, the authorization ID in the OWNER option is the owner of the object.
 - If the plan or package is bound with both the QUALIFIER option and the OWNER option, the authorization ID in the QUALIFIER option is the owner of the object.
 - If neither option is specified, the authorization ID of the binder of the plan or package is implicitly the object owner.

If the name of a user-defined function, stored procedure, distinct type, sequence, or trigger is unqualified, you can establish the ownership of one of these objects in these ways:

- If you issue the CREATE statement dynamically, the owner of the created object is your current SQL ID. That ID must have the privileges that are needed to create the object.
- If you issue the CREATE statement statically, by running a plan or package that contains it, the owner of the object is the plan or package owner. You can use the OWNER bind option to explicitly name the object owner. If you do not use the OWNER bind option, the binder of the package or plan is implicitly the object owner.

If the name of a user-defined function, stored procedure, distinct type, sequence, or trigger is unqualified, the implicit qualifier is determined based on the schema name in dynamic statements and the PATH bind option in static statements. The owner of a JAR (Java class for a routine) that is used by a stored procedure or a user-defined function is the current SQL ID of the process that performs the

INSTALL_JAR function. **GUIP**

Ownership of objects with qualified names

If an object name is qualified, the type of object indicates its ownership.

GUIP If you create a table, view, index, or alias with a qualified name, the owner of the object is the *schema name*. The schema name identifies the schema to which the object belongs. You can consider all of the objects that are qualified by the same schema name as a group of related objects.

If you create a distinct type, user-defined function, stored procedure, sequence, or trigger with a qualified name, the owner of the object is the authorization ID of the process. The owner of a JAR (Java class for a routine) that is used by a stored procedure or a user-defined function is the current SQL ID of the process that performs the `INSTALL_JAR` function. **GUIP**

Ownership of objects within a trusted context

You can simplify the administration of authorization by having roles as object owners.

GUIP If the owner of an object is an authorization ID and you need to transfer the ownership to another ID, you need to drop the object first and re-create it with the new authorization ID as the owner. You don't need to take these steps if the owner is a role because all users that are associated with that role have the owner privilege.

The definition of a trusted context determines the ownership of objects that are created in the trusted context. Assume that you issue the `CREATE` statement dynamically and that the trusted context is defined with the `ROLE AS OBJECT OWNER` clause. In this case, the associated role is the owner of the objects, regardless of whether the objects are explicitly qualified.

In contrast, assume that you issue the `CREATE` statement statically and that the plan or package is bound in the trusted context with the `ROLE AS OBJECT OWNER` clause. In this case, the role that owns the plan or package also owns the objects that are created, regardless of whether the objects are explicitly qualified.

GUIP

Changing object ownership

The privileges that are implicit in ownership cannot be revoked; you cannot replace an object's owner while the object exists. Therefore, you can change package or plan ownership only when a package or plan exists.

GUIP You might want to share ownership privileges on an object instead of replacing its owner. If so, make the owning ID a secondary ID to which several primary IDs are connected. You can change the list of primary IDs connected to the secondary ID without dropping and re-creating the object.

To change ownership of an object:

1. Drop the object, which usually deletes all privileges on it².
2. Re-create the object with a new owner. **GUIP**

2. Dropping a package does not delete all privileges on it if another version of the package still remains in the catalog.

You can change an object owner from an authorization ID to a role by using the CATMAINT UPDATE utility with the OWNER option. To do so, you must also have the installation SYSADM authority, define a trusted context with the ROLE AS OBJECT OWNER clause, and run the process in the new function mode.

For more information about this new function and its limitations, see *DB2 Utility Guide and Reference*.

Granting implicit privileges of object ownership

Certain implicit privileges of ownership correspond to the privileges that can be granted by a GRANT statement. For those that do correspond, the owner of the object can grant the privilege to another user.

GUPI **Example:** The owner of a table can grant the SELECT privilege on the table to any other user. To grant the SELECT privilege on TABLE3 to USER4, the owner of the table can issue the following statement:

```
G RANT SELECT ON TABLE3 TO USER4
```

GUPI

Managing implicit privileges through plan or package ownership

If you are a plan or package owner, you must hold privileges to perform actions on a plan or package. You can grant to any ID the privileges to execute a plan or package.

GUPI When the EXECUTE privilege on a plan or package is granted to an ID, the ID can execute a plan or package without holding the privileges for every action that the plan or package performs. However, the ID is restricted by the SQL statements in the original program.

Example: The program might contain the following statement:

```
EXEC SQL
  SELECT * INTO :EMPREC FROM DSN8910.EMP
  WHERE EMPNO='000010';
```

The statement puts the data for employee number 000010 into the host structure EMPREC. The data comes from table DSN8910.EMP, but the ID does not have unlimited access to DSN8910.EMP. Instead, the ID that has EXECUTE privilege for this plan can access rows in the DSN8910.EMP table only when EMPNO = '000010'.

If any of the privileges that are required by the plan or package are revoked from the owner, the plan or the package is invalidated. The plan or package must be rebound, and the new owner must have the required privileges. **GUPI**

Establishing or changing plan or package ownership

You can use the BIND and REBIND subcommands to create or change an application plan or a package.

GUPI On either subcommand, you can use the OWNER option to name the owner of the resulting plan or package. Consider the following factors when naming an owner:

- Any user can name the primary ID or any secondary ID.
- An ID with the BINDAGENT privilege can name the grantor of that privilege.
- An ID with SYSCTRL or SYSADM authority can name any authorization ID on a BIND command, but not on a REBIND command.

If you omit the OWNER option, your primary ID becomes the owner on BIND, and the previous owner retains ownership on REBIND.

Some systems that can bind a package at a DB2 system do not support the OWNER option. When the OWNER option is not supported, the primary authorization ID is always the owner of the package because a secondary ID cannot be named as the owner. 

Establishing plan and package ownership in a trusted context

You can issue the BIND and REBIND commands in a trusted context with the ROLE AS OBJECT OWNER clause to specify the ownership of a plan or package. In this trusted context, you can specify only a role, not an authorization ID, as the OWNER of a plan or package.

 If you specify the OWNER option, the specified role becomes the owner of the plan or package. If you don't specify the OWNER option, the role that is associated with the binder becomes the owner. If the ROLE AS OBJECT OWNER clause is omitted for the trusted context, the current rules for plan and package ownership apply.

Considerations: If you want a role to own the package at the remote DB2, you need to define the role ownership in the trusted context at the remote server. Make sure to establish the connection to the remote DB2 as trusted when binding or re-binding the package at the remote server.

If you specify the OWNER option in a trusted connection during the remote BIND processing, the outbound authorization ID translation is not performed for the OWNER.

If the plan owner is a role and the application uses a package bound at a remote DB2 for z/OS server, the privilege of the plan owner to execute the package is not considered at the remote DB2 server. The privilege set of the authorization ID (either the package owner or the process runner determined by the DYNAMICRULES behavior) at the DB2 for z/OS server must have the EXECUTE privilege on the package at the DB2 server. 

How DB2 resolves unqualified names

A plan or package can contain SQL statements that use unqualified table and view names.

 For static SQL, the default qualifier for those names is the owner of the plan or package. However, you can use the QUALIFIER option of the BIND command to specify a different qualifier. For static statements, the PATH bind option determines the path that DB2 searches to resolve unqualified distinct types, user-defined functions, stored procedures, sequences, and trigger names.

When you perform bind operations on packages or plans that contain static SQL, you should use group and ROLE authority rather than individual ID authority

whenever possible. The combinations of OWNER, QUALIFIER, SCHEMA, and ROLE ownership provide you more flexibility.

For plans or packages that contain dynamic SQL, DYNAMICRULES behavior determines how DB2 qualifies unqualified object names. For unqualified distinct types, user-defined functions, stored procedures, sequences, and trigger names in dynamic SQL statements, DB2 uses the schema name as the qualifier. DB2 finds the schema name in the CURRENT PATH special register. For unqualified tables, views, aliases, and indexes, DB2 uses the CURRENT SCHEMA special register as the qualifier.

Exception: ALTER, CREATE, DROP, COMMENT ON, GRANT, and REVOKE statements follow different conventions for assigning qualifiers. For static SQL, you must specify the qualifier for these statements in the QUALIFIER bind option. For dynamic SQL, the qualifier for these statements is the value in the CURRENT SCHEMA special register. 

Validating authorization for executing plans or packages

The plan or package owner must have authorization to execute all static SQL statements that are embedded in the plan or package. A bind operation always checks whether a local object exists and whether the owner has the required privileges on it.

 However, you do not need to have the authorization when the plan or package is bound. The objects to which the plan or package refers do not even need to exist at bind time. If the initial checking fails, an error message is returned. You can choose whether the failure prevents the bind operation from completion by using the VALIDATE option on the BIND PLAN and BIND PACKAGE commands.

The following values for the VALIDATE option determine how DB2 is to handle existence and authorization errors:

RUN If you choose RUN for the VALIDATE option, the bind succeeds even when existence or authorization errors exist. DB2 checks existence and authorization at run time.

BIND If you choose BIND for the VALIDATE option, which is recommended, the bind fails when existence or authorization errors exist. **Exception:** If you use the SQLERROR(CONTINUE) option on the BIND PACKAGE command, the bind succeeds, but the package's SQL statements that have errors cannot execute.

The corresponding existence and authorization checks for remote objects are always made at run time. Authorization to execute dynamic SQL statements is also checked at run time. Applications that use the Resource Recovery Services attachment facility (RRSAF) to connect to DB2 do not require a plan. 

Checking authorization at a DB2 database server:

A remote requester, either a DB2 for z/OS server or other requesting system, runs a package at the DB2 intermediate server. As shown in the following diagram, a statement in the package uses an alias or a three-part name to request services from a DB2 database server.

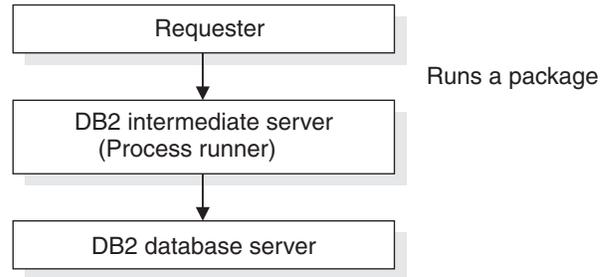
GUPI

Figure 19. Execution at a second DB2 server

The ID that is checked for the required privileges to run at the DB2 database server can be:

- The owner of the plan, if not a role, that is running at the requester site (if the requester is DB2 for z/OS)

If the owner of the plan is a role and the application uses a package bound at a remote DB2 for z/OS server, the authorization ID at the DB2 for z/OS server must have the EXECUTE privilege on the package at the DB2 server. The authorization ID can be the package owner or the process runner that is determined by the DYNAMICRULES behavior.

- The owner of the package that is running at the DB2 server

In addition, if a remote alias is used in the SQL statement, the alias must be defined at the requester site. The ID that is used depends on the following factors:

- Whether the requester is a DB2 for z/OS server or a different system
- The value of the DYNAMICRULES bind option
- Whether the HOPAATH parameter at the DB2 server site is set to BOTH or RUNNER when the DSNTIJUZ installation job is run. The default value is BOTH.
- Whether the SQL statement that is executed at the DB2 database server is static or dynamic

Note: When the DBPROTOCOL(PRIVATE) bind option is in effect for execution at the DB2 database server or the second DB2 server, the ID that is checked for the privileges that are needed to run at the second server can be:

- The owner of the plan that is running at the requester (if the requester is DB2 for z/OS)
- The owner of the package that is running at the DB2 server
- The authorization ID of the process that runs the package at the first DB2 server (the “process runner”)

GUPI

Checking authorization for executing an RRSAF application without a plan:

RRSAF provides the capability for an application to connect to DB2 and run without a DB2 plan.

If an RRSAF application does not have a plan, the following authorization rules are true:

- For the following types of packages, the primary or secondary authorization ID and role of the process are used for checking authorization to execute the package:
 - A local package
 - A remote package that is on a DB2 for z/OS system and is accessed using DRDA
- At a DB2 for z/OS system, the authorization to execute the DESCRIBE TABLE statement includes checking the primary and secondary authorization IDs.
- For a double hop situation, the authorization ID that must hold the required privileges to execute SQL statements at the second server is determined as if the requester is not a DB2 for z/OS system.

Caching authorization IDs for better performance

You can specify that DB2 caches authorization IDs for plans, packages, or routines (user-defined functions and stored procedures). Caching IDs can help improve performance, especially when IDs are frequently reused.

One cache exists for each plan, one global cache exists for packages, and a global cache exists for routines. The global cache for packages and routines are allocated at the DB2 startup. For a data sharing group, each member does its own authorization caching.

Caching authorization IDs for plans:

Authorization checking is fastest when the EXECUTE privilege is granted to PUBLIC and when the plan is reused by an ID or role that already appears in the cache.

GUIP You can set the size of the plan authorization cache by using the BIND PLAN subcommand. The default cache size is specified by an installation option, with an initial default setting of 3072 bytes. **GUIP**

Caching authorization IDs for packages:

Caching authorization IDs for packages can provide runtime benefits for handling the following objects:

- **GUIP** Stored procedures
- Remotely bound packages
- Local packages in a package list in which the plan owner does not have execute authority on the package at bind time, but does at run time
- Local packages that are not explicitly listed in a package list, but are implicitly listed by *collection-id.**, **.**, or **.package-id*

You can set the size of the package authorization cache by using the PACKAGE AUTH CACHE field on the DSNTIPP installation panel. The default value, 100 KB, is enough storage to support about 690 *collection-id.package-id* entries or *collection-id.** entries.

You can cache more package authorization information by using any of the following strategies:

- Granting package execute authority to *collection.**
- Granting package execute authority to PUBLIC for some packages or collections
- Increasing the size of the cache

- Granting package authority to a secondary ID or role when running in a trusted context.

The QTPACAUT field in the package accounting trace indicates how often DB2 succeeds at reading package authorization information from the cache. 

Caching authorization IDs for routines:

The routine authorization cache stores authorization IDs with the EXECUTE privilege on a specific routine.

 A routine is identified as *schema.routine-name.type*, where the routine name is one of the following names:

- The specific function name for user-defined functions
- The procedure name for stored procedures
- '*' for all routines in the schema

You can set the size of the routine authorization cache by using the ROUTINE AUTH CACHE field on the DSNTIPP installation panel. The initial default size of 100 KB is enough storage to support about 690 *schema.routine.type* or *schema.*.type* entries.

You can cache more authorization information about routines by using the following strategies:

- Granting EXECUTE on *schema.**
- Granting routine execute authority to PUBLIC for some or all routines in the schema
- Increasing the size of the cache
- Granting package authority to a secondary ID or role when running in a trusted context. 

Authorizing plan or package access through applications

Because an ID executes a package or plan by running an application program, implementing control measures in an application program can be useful.

 **Example:** Consider the following SQL statement:

```
EXEC SQL
  SELECT * INTO :EMPREC FROM DSN8910.EMP
  WHERE EMPNO='000010';
```

The statement permits access to the row of the employee table WHERE EMPNO='000010'. If you replace the value 000010 with a host variable, the program could supply the value of the variable and permit access to various employee numbers. Routines in the program could limit that access to certain IDs, certain times of the day, certain days of the week, or other special circumstances.

Stored procedures provide an alternative to controls in the application. By encapsulating several SQL statements into a single message to the DB2 server, a stored procedure can protect sensitive portions of the application program. Also, stored procedures can include access to non-DB2 resources, as well as DB2.

Recommendation: Do not use programs to extend security. Whenever possible, use other techniques, such as stored procedures or views, as a security mechanism. Using programs to extend security has the following drawbacks:

- Program controls are separate from other access controls, can be difficult to implement properly, are difficult to audit, and are relatively simple to bypass.
- Almost any debugging facility can be used to bypass security checks in a program.
- Other programs might use the plan without doing the needed checking.
- Errors in the program checks might allow unauthorized access.
- Because the routines that check security might be quite separate from the SQL statement, the security check could be entirely disabled without requiring a bind operation for a new plan.
- A BIND REPLACE operation for an existing plan does not necessarily revoke the existing EXECUTE privileges on the plan. (Revoking those privileges is the default, but the plan owner has the option to retain them. For packages, the EXECUTE privileges are always retained.)

For those reasons, if the program accesses any sensitive data, the EXECUTE privileges on the plan and on packages are also sensitive. They should be granted only to a carefully planned list of IDs. 

Restricting access of plans or packages to particular systems:

If you use controls in an application program, you can limit the access of a plan or package to the particular systems for which it was designed.

 DB2 does not ensure that only specific programs are used with a plan, but program-to-plan control can be enforced in IMS and CICS. DB2 provides a consistency check to avoid accidental mismatches between program and plan, but the consistency check is not a security check.

You can use the the ENABLE and DISABLE options on the BIND and REBIND subcommands to restrict access of plans and packages to a particular system.

Example: The ENABLE IMS option allows the plan or package to run from any IMS connection. Unless other systems are also named, ENABLE IMS does not allow the plan or package to run from any other type of connection.

Example: DISABLE BATCH prevents a plan or package from running through a batch job, but it allows the plan or package to run from all other types of connection.

You can exercise even finer control with the ENABLE and DISABLE options. You can enable or disable particular IMS connection names, CICS application IDs, requesting locations, and so forth. 

Authorization checking for executing packages remotely:

The privileges that are required for a remote bind (BIND PACKAGE *location.collection*) must be granted at the server location.

GUPI The ID that owns the package must have all of the privileges that are required to run the package at the server, and BINDADD³ and CREATE IN privileges at the server.

Exceptions:

- For a BIND COPY operation, the owner must have the COPY privilege at the local DB2 site or subsystem, where the package that is being copied resides.
- If the creator of the package is not the owner, the creator must have SYSCTRL authority or higher, or must have been granted the BINDAGENT privilege by the owner. That authority or privilege is granted at the local DB2.

Binding a plan with a package list (BIND PLAN PKLIST) is done at the local DB2, and bind privileges must be held there. Authorization to execute a package at a remote location is checked at execution time, as follows:

- For DB2 private protocol, the owner of the plan at the requesting DB2 must have EXECUTE privilege for the package at the DB2 server.
- For DRDA, if the server is a DB2 for z/OS subsystem, the authorization ID of the process (primary ID or any secondary ID) must have EXECUTE privilege for the package at the DB2 server.
- If the server is not DB2 for z/OS, the primary authorization ID must have whatever privileges are needed. Check that product’s documentation. **GUPI**

Managing implicit privileges through routines

You can control authorization checking by using a DB2-supplied exit routine or an exit routine that you write. If your installation uses one of these access control authorization exit routines, you can use it, instead of other methods, to control authorization checking.

Privileges required for executing routines

A number of steps are involved in implementing, defining, and invoking user-defined functions and stored procedures, which are also called *routines*.

GUPI The following table summarizes the common tasks and the privileges that are required for executing routines.

Table 52. Common tasks and required privileges for routines

Role	Tasks	Required privileges
Implementer	If SQL is in the routine: codes, precompiles, compiles, and link-edits the program to use as the routine. Binds the program as the routine package. If no SQL is in the routine: codes, compiles, and link-edits the program.	If binding a package, BINDADD system privilege and CREATE IN on the collection.
Definer	Issues a CREATE FUNCTION statement to define a user-defined function or CREATE PROCEDURE statement to define a stored procedure.	CREATEIN privilege on the schema. EXECUTE authority on the routine package when invoked.
Invoker	Invokes a routine from an SQL application.	EXECUTE authority on the routine.

3. Or BIND, depending on the installation option BIND NEW PACKAGE.

The routine *implementer* typically codes the routine in a program and precompiles the program. If the program contains SQL statements, the implementer binds the DBRM. In general, the authorization ID that binds the DBRM into a package is the package owner. The implementer is the routine package owner. As package owner, the implementer implicitly has EXECUTE authority on the package and has the authority to grant EXECUTE privileges to other users to execute the code within the package.

The implementer grants EXECUTE authority on the routine package to the definer. EXECUTE authority is necessary only if the package contains SQL. For user-defined functions, the definer requires EXECUTE authority on the package. For stored procedures, the EXECUTE privilege on the package is checked for the definer and other IDs.

The routine *definer* owns the routine. The definer issues a CREATE FUNCTION statement to define a user-defined function or a CREATE PROCEDURE statement to define a stored procedure. The definer of a routine is determined as follows:

- If the SQL statement is embedded in an application program, the definer is the authorization ID of the owner of the plan or package.
- If the SQL statement is dynamically prepared, the definer is the SQL authorization ID that is contained in the CURRENT SQLID special register. If the SQL statement is executed in a trusted context that is specified with the ROLE AS OBJECT OWNER clause, the definer is the role in effect.

The definer grants EXECUTE authority on the routine to the invoker, that is, any user that needs to invoke the routine.

The routine *invoker* invokes the routine from an SQL statement in the invoking plan or package. The invoker for a routine is determined as follows:

- For a static statement, the invoker is the plan or package owner.
- For a dynamic statement, the invoker depends on DYNAMICRULES behavior.

GUIP

Granting privileges through routines

The following example describes how to get a routine up and running, and how to use and assign the required privileges and authorizations. The routine in the example is an external user-defined function. The steps in the example are divided by the implementer, definer, and invoker roles.

Implementing a user-defined function:

To implement a user-defined function, the implementer performs the following steps:

GUIP

1. The implementer codes a program that implements the user-defined function. Assume that the implementer codes the following external user-defined function in C and names the function C_SALARY:

```

/*****
 * This routine accepts an employee serial number and a percent raise. *
 * If the employee is a manager, the raise is not applied. Otherwise, *
 * the new salary is computed, truncated if it exceeds the employee's *
 * manager's salary, and then applied to the database. *
 *****/
void C_SALARY                               /* main routine */

```

```

( char      *employeeSerial      /* in: employee serial no. */
 decimal   *percentRaise         /* in: percentage raise   */
 decimal   *newSalary,          /* out: employee's new salary */
 short int *niEmployeeSerial    /* in: indic var, empl ser */
 short int *niPercentRaise      /* in: indic var, % raise   */
 short int *niNewSalary,        /* out: indic var, new salary */
 char      *sqlstate,          /* out: SQLSTATE          */
 char      *fnName,            /* in: family name of function*/
 char      *specificName,      /* in: specific name of func */
 char      *message            /* out: diagnostic message */
)
{
EXEC SQL BEGIN DECLARE SECTION;
char      hvEMPNO-7-;          /* host var for empl serial */
decimal   hvSALARY;           /* host var for empl salary */
char      hvWORKDEPT-3-;      /* host var for empl dept no. */
decimal   hvManagerSalary;    /* host var,emp's mgr's salary*/
EXEC SQL END DECLARE SECTION;

sqlstate = 0;
memset( message,0,70 );
/*****
 * Copy the employee's serial into a host variable
 *****/
strcpy( hvEMPNO,employeeSerial );
/*****
 * Get the employee's work department and current salary
 *****/
EXEC SQL SELECT  WORKDEPT, SALARY
                INTO :hvWORKDEPT, :hvSALARY
                FROM EMP
                WHERE EMPNO = :hvEMPNO;
/*****
 * See if the employee is a manager
 *****/
EXEC SQL SELECT  DEPTNO
                INTO :hvWORKDEPT
                FROM DEPT
                WHERE MGRNO = :hvEMPNO;
/*****
 * If the employee is a manager, do not apply the raise
 *****/
if( SQLCODE == 0 )
{
    newSalary = hvSALARY;
}
/*****
 * Otherwise, compute and apply the raise such that it does not
 * exceed the employee's manager's salary
 *****/
else
{
/*****
 * Get the employee's manager's salary
 *****/
EXEC SQL SELECT  SALARY
                INTO :hvManagerSalary
                FROM EMP
                WHERE EMPNO = (SELECT MGRNO
                                FROM DSN8610.DEPT
                                WHERE DEPTNO = :hvWORKDEPT);
/*****
 * Compute proposed raise for the employee
 *****/
newSalary = hvSALARY * (1 + percentRaise/100);
/*****
 * Don't let the proposed raise exceed the manager's salary
 *****/
}
}

```

```

*****/
if( newSalary > hvManagerSalary
    newSalary = hvManagerSalary;
/*****
* Apply the raise *
*****/
hvSALARY = newSalary;
EXEC SQL UPDATE EMP
        SET SALARY = :hvSALARY
        WHERE EMPNO = :hvEMPNO;
}

return;
} /* end C_SALARY */

```

The implementer requires the UPDATE privilege on table EMP. Users with the EXECUTE privilege on function C_SALARY do not need the UPDATE privilege on the table.

2. Because this program contains SQL, the implementer performs the following steps:
 - a. Precompile the program that implements the user-defined function.
 - b. Link-edit the user-defined function with DSNRLI (RRS attachment facility), and name the program's load module C_SALARY.
 - c. Bind the DBRM into package MYCOLLID.C_SALARY.

After performing these steps, the implementer is the *function package owner*.

3. The implementer then grants EXECUTE privilege on the user-defined function package to the definer.

```

GRANT EXECUTE ON PACKAGE MYCOLLID.C_SALARY
  TO definer

```

As package owner, the implementer can grant execute privileges to other users, which allows those users to execute code within the package. For example:

```

GRANT EXECUTE ON PACKAGE MYCOLLID.C_SALARY
  TO other_user

```

GUPI

Defining a user-defined function:

To define a user-defined function, the definer performs the following steps:

GUPI

1. The definer executes the following CREATE FUNCTION statement to define the user-defined function salary_change to DB2:

```

CREATE FUNCTION
  SALARY_CHANGE(
    VARCHAR( 6 )
    DECIMAL( 5,2 ) )
  RETURNS
    DECIMAL( 9,2 )
  SPECIFIC schema.SALCHANGE
  LANGUAGE C
  DETERMINISTIC
  MODIFIES SQL DATA
  EXTERNAL NAME C_SALARY
  PARAMETER STYLE DB2SQL
  RETURNS NULL ON NULL CALL
  NO EXTERNAL ACTION
  NO SCRATCHPAD

```

```

NO FINAL CALL
ALLOW PARALLEL
NO COLLID
ASUTIME LIMIT 1
STAY RESIDENT NO
PROGRAM TYPE SUB
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

```

After executing the CREATE FUNCTION statement, the definer owns the user-defined function. The definer can execute the user-defined function package because the user-defined function package owner, in this case the implementer, granted to the definer the EXECUTE privilege on the package that contains the user-defined function.

2. The definer then grants the EXECUTE privilege on SALARY_CHANGE to all function invokers.

```

GRANT EXECUTE ON FUNCTION SALARY_CHANGE
  TO invoker1, invoker2, invoker3, invoker4

```

GUPI

Using a user-defined function:

To use a user-defined function, the invoker performs the following steps:

1. **GUPI** The invoker codes an application program, named SALARY_ADJ. The application program contains a static SQL statement that invokes the user-defined function SALARY_CHANGE. SALARY_CHANGE gives an employee a 10% raise if the employee is not a manager. The static SQL statement follows:

```

EXEC SQL SELECT  FIRSTNAME,
                  LASTNAME
                  SALARY_CHANGE( :hvEMPNO, 10.0 )
INTO :hvFIRSTNAME,
     :hvLASTNAME,
     :hvSALARY
FROM EMP
WHERE EMPNO = :hvEMPNO;

```

2. The invoker then precompiles, compiles, link-edits, and binds the invoking application's DBRM into the *invoking package* or *invoking plan*. An invoking package or invoking plan is the package or plan that contains the SQL that invokes the user-defined function. After performing these steps, the invoker is the owner of the invoking plan or package.

Restriction: The invoker must hold the SELECT privilege on the table EMP and the EXECUTE privilege on the function SALARY_CHANGE. **GUPI**

Authorization ID validation:

DB2 uses the rules for static SQL to determine the authorization ID (invoker) that executes the user-defined function package. For a static statement, the invoker is the authorization ID of the plan or package owner.

GUPI The invoking package SALARY_ADJ contains a static SQL SELECT statement that invokes the user-defined function SALARY_CHANGE.

- While execution occurs in invoking package SALARY_ADJ, DB2 uses the authorization ID of the invoker (the package owner).

The invoker requires the EXECUTE privilege on the user-defined function SALARY_CHANGE, which the package SALARY_ADJ invokes. Because the user-defined function definer has the EXECUTE privilege on the user-defined function package C_SALARY, the invoker does not require the explicit EXECUTE privilege.

- When execution changes to the user-defined function package C_SALARY, DB2 uses the authorization ID of the implementer (the package owner). The package owner is the authorization ID with authority to execute all static SQL in the user-defined function package C_SALARY. 

Authorization behaviors for dynamic SQL statements

The two key factors that influence authorization behaviors are the DYNAMICRULES value and the runtime environment of a package. The combination of the DYNAMICRULES value and the runtime environment determine the values for the dynamic SQL attributes. Those attribute values are called the *authorization behaviors*.

 The DYNAMICRULES option on the BIND or REBIND command determines the values that apply at run time for the following dynamic SQL attributes:

- The authorization ID or role 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

The DYNAMICRULES option also determines whether dynamic SQL statements can include GRANT, REVOKE, ALTER, CREATE, DROP, and RENAME statements.

In addition to the DYNAMICRULES value, the runtime environment of a package controls how dynamic SQL statements behave at run time. The two possible runtime environments are:

- The package runs as part of a stand-alone program.
- The package runs as a stored procedure or user-defined function package, or runs under a stored procedure or user-defined function.

A package that runs under a stored procedure or user-defined function is a package whose associated program meets one of the following conditions:

- The program is called by a stored procedure or user-defined function.
- The program is in a series of nested calls that start with a stored procedure or user-defined function. 

Run behavior:

DB2 processes dynamic SQL statements by using their standard attribute values. These attributes are collectively called the *run behavior*.

The run behavior consists of the following attributes:

-  DB2 uses the authorization IDs (primary, secondary and the current SQL ID) of the application process to check for authorization of dynamic SQL statements. It also checks the role in effect if running in a trusted context.

- Dynamic SQL statements use the values of application programming options that were specified during installation. The installation option USE FOR DYNAMICRULES has no effect.
- The GRANT, REVOKE, CREATE, ALTER, DROP, and RENAME statements can be executed dynamically. 

Bind behavior:

DB2 processes dynamic SQL statements by using the *bind behavior*.

 The bind behavior consists of the following attributes:

- DB2 uses the authorization ID or role of the plan or package for authorization checking of dynamic SQL statements.
- Unqualified table, view, index, and alias names in dynamic SQL statements are implicitly qualified by the default schema, which is the value of the bind option QUALIFIER. If you do not specify the QUALIFIER bind option, DB2 uses the plan or package owner as the qualifier.
The values of the authorization ID or role and the qualifier for unqualified objects are the same as those that are used for embedded or static SQL statements.
- The bind behavior consists of the common attribute values for bind, define, and invoke behaviors. 

Define behavior:

When the package is run as or under a stored procedure or a user-defined function package, DB2 processes dynamic SQL statements by using the *define behavior*.

 The define behavior consists of the following attribute values:

- DB2 uses the authorization ID or role of the user-defined function or the stored procedure owner for authorization checking of dynamic SQL statements in the application package.
- The default qualifier for unqualified objects is the user-defined function or the stored procedure owner.
- Define behavior consists of the common attribute values for bind, define, and invoke behaviors.

When the package is run as a stand-alone program, DB2 processes dynamic SQL statements using bind behavior or run behavior, depending on the

DYNAMICRULES value specified. 

Invoke behavior:

When the package is run as or under a stored procedure or a user-defined function package, DB2 processes dynamic SQL statements by using the *invoke behavior*.

 The invoke behavior consists of the following attribute values:

- DB2 uses the authorization ID of the user-defined function or the stored procedure invoker to check the authorization for dynamic SQL statements in the application package. It uses the following rules:
 - The current SQL ID of the invoker is checked for the required authorization.

- Secondary authorization IDs and roles that are associated with the primary authorization ID are also checked if they are needed for the required authorization.
- The default qualifier for unqualified objects is the user-defined function or the stored procedure invoker.
- Invoke behavior consists of the common attribute values for bind, define, and invoke behaviors.

When the package is run as a stand-alone program, DB2 processes dynamic SQL statements using bind behavior or run behavior, depending on the DYNAMICRULES specified value. 

Common attribute values for bind, define, and invoke behaviors:

The following attribute values apply to dynamic SQL statements in plans or packages that have the bind, define, or invoke behavior:

-  You can execute the statement SET CURRENT SQLID in a package or plan that is bound with any DYNAMICRULES value. However, DB2 does not use the current SQL ID as the authorization ID for dynamic SQL statements. DB2 always uses the current SQL ID as the qualifier for the EXPLAIN output PLAN_TABLE.
- If the value of installation option USE FOR DYNAMICRULES is YES, DB2 uses the application programming default values that were specified during installation to parse and semantically verify dynamic SQL statements. If the value of USE for DYNAMICRULES is NO, DB2 uses the precompiler options to parse and semantically verify dynamic SQL statements.
- The GRANT, REVOKE, CREATE, ALTER, DROP, and RENAME statements cannot be executed dynamically.

The following table shows the DYNAMICRULES values, runtime environments, and the dynamic SQL behaviors that they yield.

Table 53. How DYNAMICRULES and the runtime environment determine dynamic SQL statement behavior

DYNAMICRULES value	Behavior of dynamic SQL statements	
	Stand-alone program environment	User-defined function or stored procedure environment
BIND	Bind behavior	Bind behavior
RUN	Run behavior	Run behavior
DEFINEBIND	Bind behavior	Define behavior
DEFINERUN	Run behavior	Define behavior
INVOKEBIND	Bind behavior	Invoke behavior
INVOKERUN	Run behavior	Invoke behavior

The following table shows the dynamic SQL attribute values for each type of dynamic SQL behaviors.

Table 54. Definitions of dynamic SQL statement behaviors

Dynamic SQL attribute	Setting for dynamic SQL attributes			
	Bind behavior	Run behavior	Define behavior	Invoke behavior
Authorization ID	Plan or package owner	Authorization IDs of the process and role, if applicable	User-defined function or stored procedure owner	Authorization ID of invoker ¹
Default qualifier for unqualified objects	Bind OWNER or QUALIFIER value	Current Schema register determines the qualifier	User-defined function or stored procedure owner	Authorization ID of invoker or role
CURRENT SQLID ²	Not applicable	Applies	Not applicable	Not applicable
Source for application programming options	Determined by DSNHDECP parameter DYNRULS ³	Install panel DSNTIPF	Determined by DSNHDECP parameter DYNRULS ³	Determined by DSNHDECP parameter DYNRULS ³
Can execute GRANT, REVOKE, CREATE, ALTER, DROP, RENAME?	No	Yes	No	No

1. If the invoker is the primary authorization ID of the process or the current SQL ID, the following rules apply:
 - The ID or role of the invoker is checked for the required authorization.
 - Secondary authorization IDs are also checked if they are needed for the required authorization.
2. DB2 uses the current SQL ID as the authorization ID for dynamic SQL statements only for plans and packages that have DYNAMICRULES run behavior. For the other dynamic SQL behaviors, DB2 uses the authorization ID that is associated with each dynamic SQL behavior, as shown in this table.

The initial current SQL ID is independent of the dynamic SQL behavior. For stand-alone programs, the current SQL ID is initialized to the primary authorization ID. You can execute the SET CURRENT SQLID statement to change the current SQL ID for packages with any dynamic SQL behavior, but DB2 uses the current SQL ID only for plans and packages with run behavior.
3. The value of DSNHDECP parameter DYNRULS, which you specify in field USE FOR DYNAMICRULES in installation panel DSNTIPF, determines whether DB2 uses the precompiler options or the application programming defaults for dynamic SQL statements. 

Determining authorization IDs for dynamic SQL statements in routines:

 Suppose that A is a stored procedure and C is a program that is neither a user-defined function nor a stored procedure. Also suppose that subroutine B is called by both stored procedure A and program C. Subroutine B, which is invoked by a language call, is neither a user-defined function nor a stored procedure. AP is the package that is associated with stored procedure A, and BP is the package that is associated with subroutine B. A, B, and C execute as shown in the following diagram.

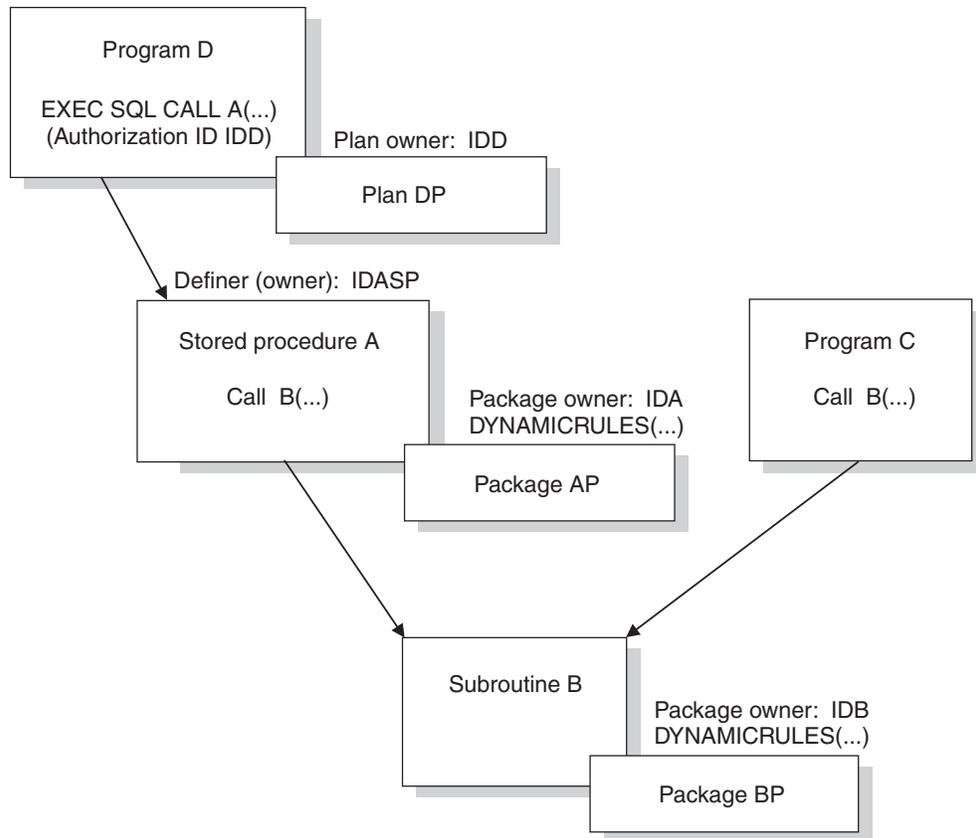


Figure 20. Authorization for dynamic SQL statements in programs and routines

Stored procedure A was defined by IDASP and is therefore owned by IDASP. The stored procedure package AP was bound by IDA and is therefore owned by IDA. Package BP was bound by IDB and is therefore owned by IDB. The authorization ID under which EXEC SQL CALL A runs is IDD, the owner of plan DP.

The authorization ID under which dynamic SQL statements in package AP run is determined in the following way:

- If package AP uses DYNAMICRULES bind behavior, the authorization ID for dynamic SQL statements in package AP is IDA, the owner of package AP.
- If package AP uses DYNAMICRULES run behavior, the authorization ID for dynamic SQL statements in package AP is the value of CURRENT SQLID when the statements execute.
- If package AP uses DYNAMICRULES define behavior, the authorization ID for dynamic SQL statements in package AP is IDASP, the definer (owner) of stored procedure A.
- If package AP uses DYNAMICRULES invoke behavior, the authorization ID for dynamic SQL statements in package AP is IDD, the invoker of stored procedure A.

The authorization ID under which dynamic SQL statements in package BP run is determined in the following way:

- If package BP uses DYNAMICRULES bind behavior, the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.

- If package BP uses DYNAMICRULES run behavior, the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
- If package BP uses DYNAMICRULES define behavior:
 - When subroutine B is called by stored procedure A, the authorization ID for dynamic SQL statements in package BP is IDASP, the definer of stored procedure A.
 - When subroutine B is called by program C:
 - If package BP uses the DYNAMICRULES option DEFINERUN, DB2 executes package BP using DYNAMICRULES run behavior, which means that the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
 - If package BP uses the DYNAMICRULES option DEFINEBIND, DB2 executes package BP using DYNAMICRULES bind behavior, which means that the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.
- If package BP uses DYNAMICRULES invoke behavior:
 - When subroutine B is called by stored procedure A, the authorization ID for dynamic SQL statements in package BP is IDD, the authorization ID under which EXEC SQL CALL A executed.
 - When subroutine B is called by program C:
 - If package BP uses the DYNAMICRULES option INVOKERUN, DB2 executes package BP using DYNAMICRULES run behavior, which means that the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
 - If package BP uses the DYNAMICRULES option INVOKEBIND, DB2 executes package BP using DYNAMICRULES bind behavior, which means that the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.

Now suppose that B is a user-defined function, as shown in the following diagram.

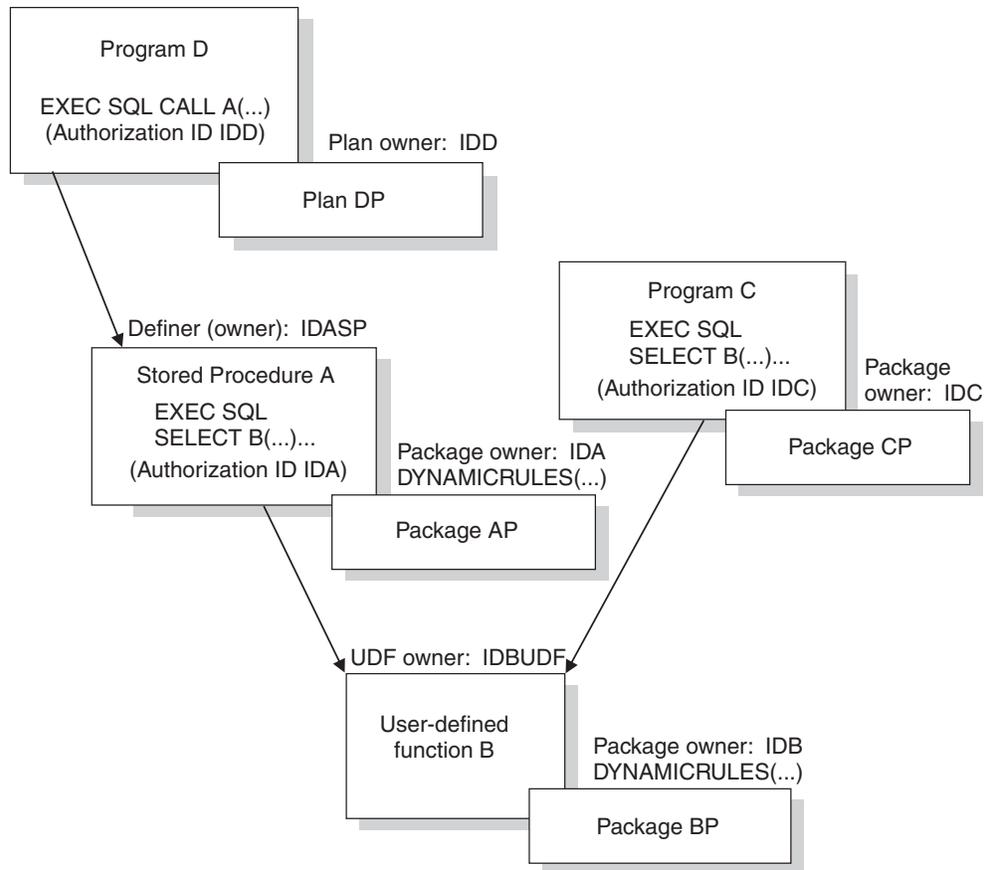


Figure 21. Authorization for dynamic SQL statements in programs and nested routines

User-defined function B was defined by IDBUDF and is therefore owned by ID IDBUDF. Stored procedure A invokes user-defined function B under authorization ID IDA. Program C invokes user-defined function B under authorization ID IDC. In both cases, the invoking SQL statement (EXEC SQL SELECT B) is static.

The authorization ID under which dynamic SQL statements in package BP run is determined in the following way:

- If package BP uses DYNAMICRULES bind behavior, the authorization ID for dynamic SQL statements in package BP is IDB, the owner of package BP.
- If package BP uses DYNAMICRULES run behavior, the authorization ID for dynamic SQL statements in package BP is the value of CURRENT SQLID when the statements execute.
- If package BP uses DYNAMICRULES define behavior, the authorization ID for dynamic SQL statements in package BP is IDBUDF, the definer of user-defined function B.
- If package BP uses DYNAMICRULES invoke behavior:
 - When user-defined function B is invoked by stored procedure A, the authorization ID for dynamic SQL statements in package BP is IDA, the authorization ID under which B is invoked in stored procedure A.
 - When user-defined function B is invoked by program C, the authorization ID for dynamic SQL statements in package BP is IDC, the owner of package CP, and is the authorization ID under which B is invoked in program C. **GUPI**

Related tasks

- “Simplifying access authorization for routines”
- “Using composite privileges”
- “Performing multiple actions in one statement”

Simplifying access authorization for routines:

You can simplify authorization for routines in several ways without violating any of the authorization standards at your installation.

GUIP Consider the following strategies to simplify authorization:

- Have the implementer bind the user-defined function package using DYNAMICRULES define behavior. With this behavior in effect, DB2 only needs to check the definer’s ID to execute dynamic SQL statements in the routine. Otherwise, DB2 needs to check the many different IDs that invoke the user-defined function.
- If you have many different routines, group those routines into schemas. Then grant EXECUTE on the routines in the schema to the appropriate users. Users have execute authority on any functions that you add to that schema.

Example: To grant the EXECUTE privilege on a schema to PUBLIC, issue the following statement:

```
GRANT EXECUTE ON FUNCTION schemaname.* TO PUBLIC;
```

GUIP

Related reference

“Determining authorization IDs for dynamic SQL statements in routines” on page 219

Using composite privileges:

An SQL statement can name more than one object. A SELECT operation, for example, can join two or more tables, or an INSERT statement can use a subquery.

GUIP These operations require privileges on all of the tables that are included in the statement. However, you might be able to issue such a statement dynamically even though one of your IDs alone does not have all the required privileges.

If the DYNAMICRULES run behavior is in effect when the dynamic statement is prepared and your primary ID, any associated role, or any of your secondary IDs has all the needed privileges, the statement is validated. However, if you embed the same statement in a host program and try to bind it into a plan or package, the validation fails. The validation also fails for the dynamic statement if DYNAMICRULES bind, define, or invoke behavior is in effect when you issue the dynamic statement. In each case, all the required privileges must be held by the single authorization ID, determined by DYNAMICRULES behavior. **GUIP**

Related reference

“Determining authorization IDs for dynamic SQL statements in routines” on page 219

Performing multiple actions in one statement:

A REBIND or FREE subcommand can name more than one plan or package. If no owner is named, the set of privileges associated with the primary ID, the associated role, and the secondary IDs must include the BIND privilege for each object.

GUPI **Example:** Suppose that a user with a secondary ID of HQFINANCE has the BIND privilege on plan P1 and that another user with a secondary ID of HQHR has the BIND privilege on plan P2. Assume that someone with HQFINANCE and HQHR as secondary authorization IDs issues the following command:

```
REBIND PLAN(P1,P2)
```

P1 and P2 are successfully rebound, even though neither the HQFINANCE nor HQHR has the BIND privilege for both plans. **GUPI**

Related reference

“Determining authorization IDs for dynamic SQL statements in routines” on page 219

Retrieving privilege records in the DB2 catalog

You can query the DB2 catalog tables by using the SQL SELECT statement. Executing the SQL statement requires appropriate privileges and authorities. You can control access to the catalog by granting and revoking these privileges and authorities.

Catalog tables with privilege records

The following catalog tables contain information about the privileges that IDs can hold.

GUPI

Table 55. Privileges information in DB2 catalog tables

Table name	Records privileges held for or authorization related to
SYSIBM.SYSCOLAUTH	Updating columns
SYSIBM.SYSDBAUTH	Databases
SYSIBM.SYSPLANAUTH	Plans
SYSIBM.SYSPACKAUTH	Packages
SYSIBM.SYSRESAUTH	Buffer pools, storage groups, collections, table spaces, JARs, and distinct types
SYSIBM.SYSROUTINEAUTH	User-defined functions and stored procedures
SYSIBM.SYSSCHEMAAUTH	Schemas
SYSIBM.SYSTABAUTH	Tables and views
SYSIBM.SYSUSERAUTH	System authorities
SYSIBM.SYSSEQUENCEAUTH	Sequences
SYSIBM.SYSCONTEXT	Associating a role with a trusted context
SYSIBM.SYSCTXTTRUSTATTRS	Associating trust attributes with a trusted context
SYSIBM.SYSCONTEXTAUTHIDS	Associating users with a trusted context

Retrieving all authorization IDs or roles with granted privileges

All authorization catalog tables include columns named GRANTEE and GRANTEETYPE. If GRANTEETYPE is blank, the value of GRANTEE is the primary or secondary authorization ID that has been granted a privilege. If GRANTEETYPE is "L", the value of GRANTEE is a role. You can modify the WHERE clause to retrieve all roles with the same privileges.

GUPI No single catalog table contains information about all privileges. However, to retrieve all IDs or roles with privileges, you can issue the SQL code as shown in the following example:

```

SELECT GRANTEE, 'PACKAGE ' FROM SYSIBM.SYSPACKAUTH
      WHERE GRANTEETYPE IN (' ', 'L')
UNION
SELECT GRANTEE, 'TABLE   ' FROM SYSIBM.SYSTABAUTH
      WHERE GRANTEETYPE IN (' ', 'L')
UNION
SELECT GRANTEE, 'COLUMN  ' FROM SYSIBM.SYSCOLAUTH
      WHERE GRANTEETYPE IN (' ', 'L')
UNION
SELECT GRANTEE, 'ROUTINE ' FROM SYSIBM.SYSROUTINEAUTH
      WHERE GRANTEETYPE IN (' ', 'L')
UNION
SELECT GRANTEE, 'PLAN    ' FROM SYSIBM.SYSPLANAUTH
      WHERE GRANTEETYPE IN (' ', 'L')
UNION
SELECT GRANTEE, 'SYSTEM  ' FROM SYSIBM.SYSUSERAUTH
      WHERE GRANTEETYPE IN (' ', 'L')
UNION
SELECT GRANTEE, 'DATABASE' FROM SYSIBM.SYSDBAUTH
      WHERE GRANTEETYPE IN (' ', 'L')
UNION
SELECT GRANTEE, 'SCHEMA  ' FROM SYSIBM.SYSSCHEMAAUTH
      WHERE GRANTEETYPE IN (' ', 'L')
UNION
SELECT GRANTEE, 'USER    ' FROM SYSIBM.SYSRESAUTH
      WHERE GRANTEETYPE IN (' ', 'L')
UNION
SELECT GRANTEE, 'SEQUENCE' FROM SYSIBM.SYSSEQUENCEAUTH
      WHERE GRANTEETYPE IN (' ', 'L');

```

Periodically, you should compare the list of IDs or roles that is retrieved by this SQL code with the following lists:

- Lists of users from subsystems that connect to DB2 (such as IMS, CICS, and TSO)
- Lists of RACF groups
- Lists of users from other DBMSs that access your DB2 subsystem

If DB2 lists IDs or roles that do not exist elsewhere, you should revoke their privileges. **GUPI**

Retrieving multiple grants of the same privilege

You can query the catalog to find duplicate grants on objects. If multiple grants clutter your catalog, consider eliminating unnecessary grants.

GUPI You can use the following SQL statement to retrieve duplicate grants on plans:

```
SELECT GRANTEE, NAME, COUNT(*)
  FROM SYSIBM.SYSPLANAUTH
  GROUP BY GRANTEE, NAME
  HAVING COUNT(*) > 2
  ORDER BY 3 DESC;
```

This statement orders the duplicate grants by frequency, so that you can easily identify the most duplicated grants. Similar statements for other catalog tables can retrieve information about multiple grants on other types of objects.

If several grantors grant the same privilege to the same grantee, the DB2 catalog can become cluttered with similar data. This similar data is often unnecessary, and it might cause poor performance.

Example: Suppose that Judy, Kate, and Patti all grant the SELECT privilege on TABLE1 to Chris. If you care that Chris's ID has the privilege but not who granted the privilege, you might consider two of the SELECT grants to be redundant and unnecessary performance liabilities.

However, you might want to maintain information about authorities that are granted from several different IDs, especially when privileges are revoked.

Example: Suppose that the SELECT privilege from the previous example is revoked from Judy. If Chris has the SELECT privilege from only Judy, Chris loses the SELECT privilege. However, Chris retains the SELECT privilege because Kate and Patti also granted the SELECT privilege to Chris. In this case, the similar grants prove not to be redundant. **GUPI**

Retrieving all authorization IDs or roles with the DBADM authority

To retrieve all authorization IDs or roles that have the DBADM authority, issue the following statement:

GUPI

```
SELECT DISTINCT GRANTEE
  FROM SYSIBM.SYSDBAUTH
  WHERE DBADMAUTH <>' ' AND GRANTEETYPE IN (' ', 'L');
```

GUPI

Retrieving all IDs or roles with access to the same table

You can retrieve all IDs or roles (GRANTEETYPE="L") that are explicitly authorized to access the same object.

GUPI

For example, to retrieve all IDs or roles (GRANTEETYPE="L") that are explicitly authorized to access the sample employee table (DSN8910.EMP in database DSN8D91A), issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME='EMP' AND TCREATOR='DSN8910' AND
  GRANTEETYPE IN (' ', 'L');
```

To retrieve all IDs or roles (GRANTEETYPE="L") that can change the sample employee table (IDs with administrative authorities and IDs to which authority is explicitly granted), issue the following statement:

```

SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME='EMP' AND
         TCREATOR='DSN8910' AND
         GRANTEE IN (' ', 'L') AND
         (ALTERAUTH <> ' ' OR
          DELETEAUTH <> ' ' OR
          INSERTAUTH <> ' ' OR
          UPDATEAUTH <> ' ')
UNION
SELECT GRANTEE FROM SYSIBM.SYSUSERAUTH
  WHERE SYSADMAUTH <> ' '
UNION
SELECT GRANTEE FROM SYSIBM.SYSDBAUTH
  WHERE DBADMAUTH <> ' ' AND NAME='DSN8D91A';

```

To retrieve the columns of DSN8910.EMP for which update privileges have been granted on a specific set of columns, issue the following statement:

```

SELECT DISTINCT COLNAME, GRANTEE, GRANTEE IN (' ', 'L') AND
  WHERE CREATOR='DSN8910' AND TNAME='EMP'
ORDER BY COLNAME;

```

The character in the GRANTEE column shows whether the privileges have been granted to a primary or secondary authorization ID (blank), a role (L), or are used by an application plan or package (P).

To retrieve the IDs that have been granted the privilege of updating one or more columns of DSN8910.EMP, issue the following statement:

```

SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME='EMP' AND
         TCREATOR='DSN8910' AND
         GRANTEE IN (' ', 'L') AND
         UPDATEAUTH <> ' ';

```

The query returns only the IDs or roles (GRANTEETYPE="L") to which update privileges have been specifically granted. It does not return IDs or roles that have the privilege because of SYSADM or DBADM authority. You could include them by forming a union with additional queries, as shown in the following example:

```

SELECT DISTINCT GRANTEE GRANTEE IN (' ', 'L') AND
  WHERE TTNAME='EMP' AND
         TCREATOR='DSN8910' AND
         GRANTEE IN (' ', 'L') AND
         UPDATEAUTH <> ' '
UNION
SELECT GRANTEE FROM SYSIBM.SYSUSERAUTH
  WHERE SYSADMAUTH <> ' '
UNION
SELECT GRANTEE FROM SYSIBM.SYSDBAUTH
  WHERE DBADMAUTH <> ' ' AND NAME='DSN8D91A';

```



Retrieving all IDs or roles with access to the same routine

You can retrieve the IDs or roles (GRANTEETYPE="L") that are authorized to access the same routines.

GUIP For example, to retrieve the IDs or roles (GRANTEETYPE="L") that are authorized to access stored procedure PROC A in schema SCHEMA1, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSROUTINEAUTH
WHERE SPECIFICNAME='PROCA' AND
      SCHEMA='SCHEMA1' AND
      GRANTEE IN (' ', 'L') AND
      ROUTINETYPE='P';
```

You can write a similar statement to retrieve the IDs or roles (GRANTEETYPE="L") that are authorized to access a user-defined function. To retrieve the IDs or roles that are authorized to access user-defined function UDFA in schema SCHEMA1, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSROUTINEAUTH
WHERE SPECIFICNAME='UDFA' AND
      SCHEMA='SCHEMA1' AND
      GRANTEE IN (' ', 'L') AND
      ROUTINETYPE='F';
```

GUIP

Retrieving tables or views accessible by an ID

You can retrieve all tables or views that can be accessed by an ID.

GUIP For example, to retrieve the list of tables and views that PGM R001 can access, issue the following statement:

```
SELECT DISTINCT TCREATOR, TTNAME FROM SYSIBM.SYSTABAUTH
WHERE GRANTEE = 'PGMR001' AND GRANTEE = ' ';
```

To retrieve the tables, views, and aliases that PGM R001 owns, issue the following statement:

```
SELECT NAME FROM SYSIBM.SYSTABLES
WHERE CREATOR='PGMR001' OR
      OWNER='PGMR001'; <--- needs mods:if owner> else=creator
```

GUIP

Retrieving plans or packages with access to the same table

You can retrieve all the plans or packages that are granted access to the same table.

GUIP For example, to retrieve the names of application plans and packages that refer to table DSN8910.EMP directly, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
WHERE GRANTEE = 'P' AND
      TCREATOR = 'DSN8910' AND
      TTNAME = 'EMP';
```

The preceding query does not distinguish between plans and packages. To identify a package, use the COLLID column of table SYSTABAUTH, which names the collection in which a package resides and is blank for a plan.

A plan or package can refer to the table indirectly, through a view.

To find all views that refer to the table:

1. Issue the following query:

```
SELECT DISTINCT DNAME FROM SYSIBM.SYSVIEWDEP
WHERE BTYPE = 'T' AND
      BCREATOR = 'DSN8910' AND
      BNAME = 'EMP';
```

2. Write down the names of the views that satisfy the query. These values are instances of *DNAME_list*.
3. Find all plans and packages that refer to those views by issuing a series of SQL statements. For each instance of *DNAME_list*, issue the following statement:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
WHERE GRANTEEType = 'P' AND
      TCREATOR = 'DSN8910' AND
      TTNAME = DNAME_list;
```

GUIP

Retrieving privilege information through views

Only an ID with the SYSADM or SYSCTRL authority automatically has the privilege of retrieving data from catalog tables. If you do not want to grant the SELECT privilege on all catalog tables to PUBLIC, consider using views to let each ID retrieve information about its own privileges.

GUIP

For example, the following view includes the owner and the name of every table on which a user's primary authorization ID has the SELECT privilege:

```
CREATE VIEW MYSELECTS AS
SELECT TCREATOR, TTNAME FROM SYSIBM.SYSTABAUTH
WHERE SELECTAUTH <> ' ' AND
      GRANTEEType = ' ' AND
      GRANTEE IN (USER, 'PUBLIC', 'PUBLIC*', CURRENT SQLID);
```

The keyword USER in that statement is equal to the value of the primary authorization ID. To include tables that can be read by a secondary ID, set the current SQLID to that secondary ID before querying the view.

To make the view available to every ID, issue the following GRANT statement:
GRANT SELECT ON MYSELECTS TO PUBLIC;

Similar views can show other privileges. This view shows privileges over columns:

```
CREATE VIEW MYCOLS (OWNER, TNAME, CNAME, REMARKS, LABEL)
AS SELECT DISTINCT TCREATOR, TBNAME, NAME, REMARKS, LABEL
FROM SYSIBM.SYSCOLUMNS, SYSIBM.SYSTABAUTH
WHERE TCREATOR = TCREATOR AND
      TTNAME = TBNAME AND
      GRANTEEType = ' ' AND
      GRANTEE IN (USER, 'PUBLIC', CURRENT SQLID, 'PUBLIC*');
```

GUIP

Implementing multilevel security with DB2

Multilevel security allows you to classify objects and users with security labels that are based on hierarchical security levels and non-hierarchical security categories. Multilevel security prevents unauthorized users from accessing information at a higher classification than their authorization, and prevents users from declassifying information.

Using multilevel security with row-level granularity, you can define security for DB2 objects and perform security checks, including row-level security checks. Row-level security checks allow you to control which users have authorization to view, modify, or perform other actions on specific rows of data.

You can implement multilevel security with the following combinations:

DB2 authorization with multilevel security with row-level granularity

In this combination, DB2 grants are used for authorization at the DB2 object level (database, table, and so forth). Multilevel security is implemented only at the row level within DB2.

External access control and multilevel security with row-level granularity

In this combination, external access control (such as the RACF access control module) is used for authorization at the DB2 object level. External access control also uses security labels to perform mandatory access checking on DB2 objects as part of multilevel security. Multilevel security is also implemented on the row level within DB2.

Important: The following information about multilevel security is specific to DB2. It does not describe all aspects of multilevel security. However, this specific information assumes that you have general knowledge of multilevel security.

Multilevel security

Multilevel security is a security policy that allows you to classify objects and users based on a system of hierarchical security levels and a system of non-hierarchical security categories.

Multilevel security provides the capability to prevent unauthorized users from accessing information at a higher classification than their authorization, and prevents users from declassifying information.

Multilevel security offers the following advantages:

- Multilevel security enforcement is mandatory and automatic.
- Multilevel security can use methods that are difficult to express through traditional SQL views or queries.
- Multilevel security does not rely on special views or database variables to provide row-level security control.
- Multilevel security controls are consistent and integrated across the system, so that you can avoid defining users and authorizations more than once.
- Multilevel security does not allow users to declassify information.

Using multilevel security, you can define security for DB2 objects and perform other checks, including row-level security checks. Row-level security checks allow you to control which users have authorization to view, modify, or perform other actions on specific rows of data.

Security labels

Multilevel security restricts access to an object or a row based on the security label of the object or row and the security label of the user.

For local connections, the security label of the user is the security label that the user specified during sign-on. This security label is associated with the DB2

primary authorization ID and accessed from the RACF ACEE control block. If no security label is specified during sign-on, the security label is the user's default security label.

| For normal TCP/IP connections, the security label of the user can be defined by
| the security zone. IP addresses are grouped into security zones on the DB2 server.
| For trusted TCP/IP connections, the security label of the user is the security label
| established under the trusted context.

For SNA connections, the default security label for the user is used instead of the security label that the user signed on with.

| Security labels can be assigned to a user by establishing a trusted connection
| within a trusted context. The trusted context definition specifies the security label
| that is associated with a user on the trusted connection. You can define trusted
| contexts if you have the SYSADM authority.

Security labels are based on security levels and security categories. You can use the Common Criteria (COMCRIT) environment's subsystem parameter to require that all tables in the subsystem are defined with security labels.

| When defining security labels, do not include national characters, such as @, #, and
| \$. Use of these characters in security labels may cause CCSID conversions to
| terminate abnormally.

Determining the security label of a user

DB2 provides several built-in session variables that contain information about the server and application process. You can obtain the value of a built-in session variable by invoking the GETVARIABLE command with the name of the built-in session variable.

One of the built-in session variables is the user's security label. You can issue the GETVARIABLE('SYSIBM.SECLABEL') command to obtain the security label of a user.

Security levels

Along with security categories, hierarchical security levels are used as a basis for mandatory access checking decisions.

When you define the security level of an object, you define the degree of sensitivity of that object. Security levels ensure that an object of a certain security level is protected from access by a user of a lower security level.

Security categories

Security categories are the non-hierarchical basis for mandatory access checking decisions.

When making security decisions, mandatory access control checks whether one set of security categories includes the security categories that are defined in a second set of security categories.

Users and objects in multilevel security

In multilevel security, the relationship between users and objects is important. In the context of multilevel security, *user* is any entity that requires access to system resources; the entity can be a human user, a stored procedure, or a batch job. An

object is any system resource to which access must be controlled; the resource can be a data set, a table, a table row, or a command.

Global temporary tables with multilevel security

For a declared temporary table with a column definition, no syntax exists to specify a security label on a DECLARE GLOBAL TEMPORARY TABLE statement. An attempt to specify a security label results in an error.

If a DECLARE GLOBAL TEMPORARY TABLE statement uses a fullselect or a LIKE predicate or a CREATE GLOBAL TEMPORARY TABLE statement uses a LIKE predicate, the resulting temporary table can inherit the security label column from the referenced table or view. However, the temporary table does not inherit any security attributes on that column. That means that the inherited column in the temporary table is not defined AS SECURITY LABEL. The column in the temporary table is defined as NOT NULL, with no default. Therefore, any statements that insert data in the temporary table must provide a value for the inherited column.

Materialized query tables with multilevel security

Materialized query tables are tables that contain information that is derived and summarized from other tables.

If one or more of the source tables for a materialized query table has multilevel security with row-level granularity enabled, some additional rules apply to working with the materialized query table and the source tables.

Constraints in a multilevel-secure environment

Constraints operate in an multilevel-secure environment in the following ways:

- A unique constraint is allowed on a security label column.
- A referential constraint is not allowed on a security label column.
- A check constraint is not allowed on a security label column.

Multilevel security with row-level checking is not enforced when DB2 checks a referential constraint. Although a referential constraint is not allowed for the security label column, DB2 enforces referential constraints for other columns in the table that are not defined with a security label.

Field, edit, and validation procedures in a multilevel-secure environment

In an multilevel-secure environment, field, edit, and validation procedures operate in the following ways:

- Field procedures and edit procedures are not allowed on a security label column.
- Validation procedures are allowed on a table that is defined with a security label column. When an authorized user with write-down privilege makes an INSERT or UPDATE request for a row, the validation procedure passes the new row with the security label of the user. If the authorized user does not have write-down privilege, the security label of the row remains the same.

Triggers in a multilevel-secure environment

When a transition table is generated as the result of a trigger, the security label of the table or row from the original table is not inherited by the transition table. Therefore, multilevel security with row-level checking is not enforced for transition tables and transition values.

If an ALTER TABLE statement is used to add a security label column to a table with a trigger on it, the same rules apply to the new security label column that would apply to any column that is added to the table with the trigger on it.

When a BEFORE trigger is activated, the value of the NEW transition variable that corresponds to the security label column is set to the security label of the user if either of the following criteria are met:

- Write-down control is in effect and the user does not have the write-down privilege
- The value of the security label column is not specified

Mandatory access checking

Mandatory access checking evaluates the dominance relationships between user security labels and object security labels and determines whether to allow certain actions, based on the following rules:

- If the security label of the user dominates the security label of the object, the user can read from the object.
- If the security label of a user and the security label of the object are equivalent, the user can read from and write to the object.
- If the security label of the user dominates the security label of the object, the user cannot write to the object unless the user has the write-down RACF privilege.
- If the security label of the user is disjoint with the security label of the object, the user cannot read or write to that object.

Exception: IDs with the installation SYSADM authority bypass mandatory access checking at the DB2 object level because actions by Install SYSADM do not invoke the external access control exit routine (DSNX@XAC). However, multilevel security with row-level granularity is enforced for IDs with Install SYSADM authority.

After the user passes the mandatory access check, a discretionary check follows. The discretionary access check restricts access to objects based on the identity of a user, the user's role (if one exists), and the groups to which the user belongs. The discretionary access check ensures that the user is identified as having a "need to know" for the requested resource. The check is discretionary because a user with a certain access permission is capable of passing that permission to any other user.

Dominance relationships between security labels

Mandatory access checking is based on the *dominance* relationships between user security labels and object security labels. One security label dominates another security label when both of the following conditions are true:

- The security level that defines the first security label is greater than or equal to the security level that defines the second security label.
- The set of security categories that defines one security label includes the set of security categories that defines the other security label.

Comparisons between user security labels and object security labels can result in four types of relationships:

Dominant

One security label dominates another security label when both of the following conditions are true:

- The security level that defines the first security label is greater than or equal to the security level that defines the second security label.

- The set of security categories that defines the first security label includes the set of security categories that defines the other security label.

Reading data requires that the user security label dominates the data security label.

Reverse dominant

One security label reverse dominates another security label when both of the following conditions are true:

- The security level that defines the first security label is less than or equal to the security level that defines the second security label.
- The set of security categories that defines the first security label is a subset of the security categories that defines the other security label.

Equivalent

One security label is equivalent to another security label when they are the same or have the same level and set of categories. If both dominance and reverse dominance are true for two security labels, they are equivalent. The user security label must be equivalent to the data security label to be able to read and write data without being able to write down.

Disjoint

A security label is disjoint or incompatible with another security label if incompatible security categories cause neither security label to dominate the other security label. Two security labels are disjoint when each of them has at least one category that the other does not have. Disjoint access is not allowed, even when a user is allowed to write down. If a user security label that is disjoint to the data security label issues an INSERT, UPDATE, or LOAD command, DB2 issues an error.

Example: Suppose that the security level "secret" for the security label HIGH is greater than the security level "sensitive" for the security label MEDIUM. Also, suppose that the security label HIGH includes the security categories Project_A, Project_B, and Project_C, and that the security label MEDIUM includes the security categories Project_A and Project_B. The security label HIGH dominates the security label MEDIUM because both conditions for dominance are true.

Example: Suppose that the security label HIGH includes the security categories Project_A, Project_B, and Project_C, and that the security label MEDIUM includes the security categories Project_A and Project_Z. In this case, the security label HIGH does not dominate the security label MEDIUM because the set of security categories that define the security label HIGH does not contain the security category Project_Z.

Write-down control

Mandatory access checking prevents a user from declassifying information; it does not allow a user to write to an object unless the security label of the user is equivalent to or dominated by the security label of the object. DB2 requires either the equivalence of the security labels or the *write-down privilege* of the user to write to DB2 objects.

GUPI **Example:** Suppose that user1 has a security label of HIGH and that row_x has a security label of MEDIUM. Because the security label of the user and the security label of the row are not equivalent, user1 cannot write to row_x. Therefore, write-down control prevents user1 from declassifying the information that is in row_x.

Example: Suppose that user2 has a security label of MEDIUM and that row_x has a security label of MEDIUM. Because the security label of the user and the security label of the row are equivalent, user2 can read from and write to row_x. However, user2 cannot change the security label for row_x unless user2 has write-down privilege. Therefore write-down control prevents user2 from declassifying the information that is in row_x. 

Granting write-down privileges

To grant write-down privilege, you need to define a profile and then allow users to access the profile.

 To grant write-down privilege to users:

1. Issue the following RACF command to define an IRR.WRITEDOWN.BYUSER profile.

```
RDEFINE FACILITY IRR.WRITEDOWN.BYUSER UACC(NONE)
```

2. Issue the following RACF command to allow users to access the IRR.WRITEDOWN.BYUSER profile that you just created.

```
PERMIT IRR.WRITEDOWN.BYUSER ID(USRT051 USRT052 USRT054 USRT056 -  
USRT058 USRT060 USRT062 USRT064 USRT066 USRT068 USRT041) -  
ACCESS(UPDATE) CLASS(FACILITY)
```



Implementing multilevel security at the object level

Perform the following steps to implement multilevel security with DB2 at the object level:



1. Define security labels in RACF for all DB2 objects that require mandatory access checking by using the RDEFINE command.

Define security labels for the following RACF resource classes:

- DSNADM (administrative authorities)
- DSNR (access to DB2 subsystems)
- MDSNBP and GDSNBP (buffer pools)
- MDSNCL and GDSNCL (collections)
- MDSNJR and MDSNJR (JAR)
- MDSNPN and GDSNPN (plans)
- MDSNSC and GDSNSC (schema)
- MDSNSG and GDSNSG (storage groups)
- MDSNSM and GDSNSM (system privileges)
- MDSNSP and GDSNSP (stored procedures)
- MDSNSQ and GDSNSQ (sequences)
- MDSNTB and GDSNTB (tables, views, indexes)
- MDSNTS and GDSNTS (table spaces)
- MDSNUF and GDSNUF (user-defined functions)

Recommendation: Define the security label SYSMULTI for DB2 subsystems that are accessed by users with different security labels and tables that require row-level granularity.

2. Specify a proper hierarchy of security labels.

In general, the security label of an object that is higher in the object hierarchy should dominate the security labels of objects that are lower in the hierarchy. RACF and DB2 do not enforce the hierarchy; they merely enforce the dominance rules that you establish.

You can use RACF to define security labels for the DB2 objects in the following object hierarchy:

- Subsystem or data sharing group
 - Database
 - Table space
 - Table
 - Column
 - Row
 - View
 - Storage group
 - Buffer pool
 - Plan
 - Collection
 - Package
 - Schema
 - Stored procedure or user-defined function
 - Java Archive (JAR)
 - Distinct type
 - Sequence

The following examples suggest dominance relationships among objects in the DB2 object hierarchy.

Example: A collection should dominate a package.

Example: A subsystem should dominate a database. That database should dominate a table space. That table space should dominate a table. That table should dominate a column.

Example: If a view is based on a single table, the table should dominate the view. However, if a view is based on multiple tables, the view should dominate the tables.

3. Define security labels and associate users with the security labels in RACF. If you are using a TCP/IP connection, you need to define security labels in RACF for the security zones into which IP addresses are grouped. These IP addresses represent remote users. Give users with SYSADM, SYSCTRL, and SYSOPR authority the security label of SYSHIGH.
4. Activate the SECLABEL class in RACF. If you want to enforce write-down control, turn on write-down control in RACF.
5. Install the external security access control authorization exit routine (DSNX@XAC), such as the RACF access control module.



Implementing multilevel security with row-level granularity

Many applications need row-level security within the relational database so that access can be restricted to a specific set of rows. This security control often needs to be mandatory so that users, including application programmers and database administrators, are unable to bypass the row-level security mechanism. Using mandatory controls with z/OS and RACF provides consistency across the system.

GUPI **Requirement:** You must have z/OS Version 1 Release 5 or later to use DB2 authorization with multilevel security with row-level granularity.

You can implement multilevel security with row-level granularity with or without implementing multilevel security on the object level. If you implement multilevel security on the object level, you must define security labels in RACF for all DB2 objects and install the external security access control authorization exit routine. If you do not use the access control authorization exit routine or RACF access control, you can use DB2 native authorization control.

You can implement multilevel security with row-level granularity with or without implementing multilevel security on the object level. If you don't implement multilevel security on the object level,

Recommendation: Use multilevel security at the object level with multilevel security with row-level granularity. Using RACF with multilevel security provides an independent check at run time and always checks the authorization of a user to the data.

DB2 performs multilevel security with row-level granularity by comparing the security label of the user to the security label of the row that is accessed. Because security labels can be equivalent without being identical, DB2 uses the RACROUTE REQUEST=DIRAUTH macro to make this comparison when the two security labels are not the same. For read operations, such as SELECT, DB2 uses ACCESS=READ. For update operations, DB2 uses ACCESS=READWRITE.

The write-down privilege for multilevel security with row-level granularity has the following properties:

- A user with the write-down privilege can update the security label of a row to any valid value. The user can make this update independent of the user's dominance relationship with the row.
- DB2 requires that a user have the write-down privilege to perform certain utilities.
- If write-down control is not enabled, all users with valid security labels are equivalent to users with the write-down privilege. **GUPI**

Creating tables with multilevel security

You can use multilevel security with row-level checking to control table access by creating or altering a table to have a column with the AS SECURITY LABEL attribute.

GUPI Tables with multilevel security in effect can be dropped by using the DROP TABLE statement. Users must have a valid security label to execute CREATE TABLE, ALTER TABLE, and DROP TABLE statements on tables with multilevel security enabled.

The performance of tables that you create and alter can suffer if the security label is not included in indexes. The security label column is used whenever a table with multilevel security enabled is accessed. Therefore, the security label column should be included in indexes on the table. If you do not index the security label column, you cannot maintain index-only access.

When a user with a valid security label creates a table, the user can implement row-level security by including a security label column. The security label column

can have any name, but it must be defined as CHAR(8) and NOT NULL WITH DEFAULT. It also must be defined with the AS SECURITY LABEL clause.

Example: To create a table that is named TABLEMLS1 and that has row-level security enabled, issue the following statement:

```
CREATE TABLE TABLEMLS1
  (EMPNO      CHAR(6)          NOT NULL,
   EMPNAME    VARCHAR(20)     NOT NULL,
   DEPTNO     VARCHAR(5)
   SECURITY   CHAR(8)          NOT NULL WITH DEFAULT AS SECURITY LABEL,
   PRIMARY KEY (EMPNO)
 )
IN DSN8D71A.DSN8S71D;
```

After the user specifies the AS SECURITY LABEL clause on a column, users can indicate the security label for each row by entering values in that column. When a user creates a table and includes a security label column, SYSIBM.SYSTABLES indicates that the table has row-level security enabled. Once a user creates a table with a security label column, the security on the table cannot be disabled. The table must be dropped and recreated to remove this protection. 

Adding multilevel security to existing tables

With a valid security label, you can implement row-level security on an existing table by adding a security label column to the existing table.

 The security label column can have any name, but it must be defined as CHAR(8) and NOT NULL WITH DEFAULT. It also must be defined with the AS SECURITY LABEL clause.

Example: Suppose that the table EMP does not have row-level security enabled. To alter EMP so that it has row-level security enabled, issue the following statement:

```
ALTER TABLE EMP
  ADD SECURITY   CHAR(8)          NOT NULL WITH DEFAULT AS SECURITY LABEL;
```

After a user specifies the AS SECURITY LABEL clause on a column, row-level security is enabled on the table and cannot be disabled. The security label for existing rows in the table at the time of the alter is the same as the security label of the user that issued the ALTER TABLE statement.

Important: Plans, packages, and dynamic statements are invalidated when a table is altered to add a security label column. 

Removing tables with multilevel security

When a user with a valid security label drops a table that has row-level security in effect, the system generates an audit record. Row-level security does not affect the success of a DROP statement; the user's privilege on the table determines whether the statement succeeds.

Caching security labels

Caching is used with multilevel security with row-level granularity to improve performance.

DB2 caches all security labels that are checked (successfully and unsuccessfully) during processing. At commit or rollback, the security labels are removed from the cache. If a security policy that employs multilevel security with row-level granularity requires an immediate change and long-running applications have not committed or rolled back, you might need to cancel the application.

Restricting access to the security label column

If you do not want users to see a security label column, you can create views that do not include the column.

GUIP **Example:** Suppose that the ORDER table has the following columns: ORDERNO, PRODNO, CUSTNO, SECURITY. Suppose that SECURITY is the security label column, and that you do not want users to see the SECURITY column. Use the following statement to create a view that hides the security label column from users:

```
CREATE VIEW V1 AS
  SELECT ORDERNO, PRODNO, CUSTNO FROM ORDER;
```

Alternatively, you can create views that give each user access only to the rows that include that user's security label column. To do that, retrieve the value of the SYSIBM.SECLABEL session variable, and create a view that includes only the rows that match the session variable value.

Example: To allow access only to the rows that match the user's security label, use the following CREATE statement:

```
CREATE VIEW V2 AS SELECT * FROM ORDER
  WHERE SECURITY=GETVARIABLE('SYSIBM.SECLABEL');
```

GUIP

Managing data in a multilevel-secure environment

Multilevel security with row-level checking affects the results of the SELECT, INSERT, UPDATE, MERGE, DELETE, and TRUNCATE statements.

For example, row-level checking ensures that DB2 does not return rows that have a HIGH security label to a user that has a LOW security label. Users must have a valid security label to execute the SELECT, INSERT, UPDATE, MERGE, DELETE, and TRUNCATE statements.

This effect also applies to the results of the LOAD, UNLOAD, and REORG TABLESPACE utilities on tables that are enabled with multilevel security.

Using the SELECT statement with multilevel security

When a user with a valid security label selects data from a table or tables with row-level security enabled, DB2 compares the security label of the user to the security label of each row.

GUIP Results from the checking are returned according to the following rules:

- If the security label of the user dominates the security label of the row, DB2 returns the row.
- If the security label of the user does not dominate the security label of the row, DB2 does not return the data from that row, and DB2 does not generate an error report.

Example: Suppose that Alan has a security label of HIGH, Beth has a security label of MEDIUM, and Carlos has a security label of LOW. Suppose that DSN8910.EMP contains the data that is shown in the following table and that the SECURITY column has been declared with the AS SECURITY LABEL clause.

Table 56. Sample data from DSN8910.EMP

EMPNO	LASTNAME	WORKDEPT	SECURITY
000010	HAAS	A00	LOW
000190	BROWN	D11	HIGH
000200	JONES	D11	MEDIUM
000210	LUTZ	D11	LOW
000330	LEE	E21	MEDIUM

Now, suppose that Alan, Beth, and Carlos each submit the following SELECT statement:

```
SELECT LASTNAME
FROM EMP
ORDER BY LASTNAME;
```

Because Alan has the security label HIGH, he receives the following result:

```
BROWN
HAAS
JONES
LEE
LUTZ
```

Because Beth has the security label MEDIUM, she receives the following result:

```
HAAS
JONES
LEE
LUTZ
```

Beth does not see BROWN in her result set because the row with that information has a security label of HIGH.

Because Carlos has the security label LOW, he receives the following result:

```
HAAS
LUTZ
```

Carlos does not see BROWN, JONES, or LEE in his result set because the rows with that information have security labels that dominate Carlos's security label. Although Beth and Carlos do not receive the full result set for the query, DB2 does

not return an error code to Beth or Carlos. 

Using the INSERT statement with multilevel security

When a user with a valid security label inserts data into a table with row-level security enabled, the security label of the row is determined according to the following rules:

-  If the user has write-down privilege or write-down control is not enabled, the user can set the security label for the row to any valid security label. If the user does not specify a value for the security label, the security label of the row becomes the same as the security label of the user.
- If the user does not have write-down privilege and write-down control is enabled, the security label of the row becomes the same as the security label of the user.

Example: Suppose that Alan has a security label of HIGH, that Beth has a security label of MEDIUM and write-down privilege defined in RACF, and that Carlos has a security label of LOW. Write-down control is enabled.

Now, suppose that Alan, Beth, and Carlos each submit the following INSERT statement:

```
INSERT INTO DSN8910.EMP(EMPNO, LASTNAME, WORKDEPT, SECURITY)
VALUES('099990', 'SMITH', 'C01', 'MEDIUM');
```

Because Alan does not have write-down privilege, Alan cannot choose the security label of the row that he inserts. Therefore DB2 ignores the security label of MEDIUM that is specified in the statement. The security label of the row becomes HIGH because Alan's security label is HIGH.

Because Beth has write-down privilege on the table, she can specify the security label of the new row. In this case, the security label of the new row is MEDIUM. If Beth submits a similar INSERT statement that specifies a value of LOW for the security column, the security label for the row becomes LOW.

Because Carlos does not have write-down privilege, Carlos cannot choose the security label of the row that he inserts. Therefore DB2 ignores the security label of MEDIUM that is specified in the statement. The security label of the row becomes LOW because Carlos' security label is LOW.

Considerations for INSERT from a fullselect: For statements that insert the result of a fullselect, DB2 does not return an error code if the fullselect contains a table with a security label column. DB2 allows it if the target table does not contain a security label column while the source table contains one.

Considerations for SELECT...FROM...INSERT statements: If the user has write-down privilege or write-down control is not in effect, the security label of the user might not dominate the security label of the row. For statements that insert rows and select the inserted rows, the INSERT statement succeeds. However, the inserted row is not returned.

Considerations for INSERT with subselect: If you insert data into a table that does not have a security label column, but a subselect in the INSERT statement does include a table with a security label column, row-level checking is performed for the subselect. However, the inserted rows will not be stored with a security label column. 

Using the UPDATE statement with multilevel security

When a user with a valid security label updates a table with row-level security enabled, DB2 compares the security label of the user to the security label of the row.

 The update proceeds according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is updated and the value of the security label is determined by whether the user has write-down privilege:
 - If the user has write-down privilege or write-down control is not enabled, the user can set the security label of the row to any valid security label.

- If the user does not have write-down privilege and write-down control is enabled, the security label of the row is set to the value of the security label of the user.
- If the security label of the user dominates the security label of the row, the result of the UPDATE statement is determined by whether the user has write-down privilege:
 - If the user has write-down privilege or write-down control is not enabled, the row is updated and the user can set the security label of the row to any valid security label.
 - If the user does not have write-down privilege and write-down control is enabled, the row is not updated.
- If the security label of the row dominates the security label of the user, the row is not updated.

Example: Suppose that Alan has a security label of HIGH, that Beth has a security label of MEDIUM and write-down privilege defined in RACF, and that Carlos has a security label of LOW. Write-down control is enabled.

Suppose that DSN8910.EMP contains the data that is shown in the following table and that the SECURITY column has been declared with the AS SECURITY LABEL clause.

Table 57. Sample data from DSN8910.EMP

EMPNO	LASTNAME	WORKDEPT	SECURITY
000190	BROWN	D11	HIGH
000200	JONES	D11	MEDIUM
000210	LUTZ	D11	LOW

Now, suppose that Alan, Beth, and Carlos each submit the following UPDATE statement:

```
UPDATE DSN8910.EMP
  SET DEPTNO='X55', SECURITY='MEDIUM'
  WHERE DEPTNO='D11';
```

Because Alan has a security label that dominates the rows with security labels of MEDIUM and LOW, his write-down privilege determines whether these rows are updated. Alan does not have write-down privilege, so the update fails for these rows. Because Alan has a security label that is equivalent to the security label of the row with HIGH security, the update on that row succeeds. However, the security label for that row remains HIGH because Alan does not have the write-down privilege that is required to set the security label to any value. The results of Alan's update are shown in the following table:

Table 58. Sample data from DSN8910.EMP after Alan's update

EMPNO	EMPNAME	DEPTNO	SECURITY
000190	BROWN	X55	HIGH
000200	JONES	D11	MEDIUM
000210	LUTZ	D11	LOW

Because Beth has a security label that dominates the row with a security label of LOW, her write-down privilege determines whether this row is updated. Beth has write-down privilege, so the update succeeds for this row and the security label for

the row becomes MEDIUM. Because Beth has a security label that is equivalent to the security label of the row with MEDIUM security, the update succeeds for that row. Because the row with the security label of HIGH dominates Beth's security label, the update fails for that row. The results of Beth's update are shown in the following table:

Table 59. Sample data from DSN8910.EMP after Beth's update

EMPNO	EMPNAME	DEPTNO	SECURITY
000190	BROWN	D11	HIGH
000200	JONES	X55	MEDIUM
000210	LUTZ	X55	MEDIUM

Because Carlos's security label is LOW, the update fails for the rows with security labels of MEDIUM and HIGH. Because Carlos has a security label that is equivalent to the security label of the row with LOW security, the update on that row succeeds. However, the security label for that row remains LOW because Carlos does not have the write-down privilege, which is required to set the security label to any value. The results of Carlos's update are shown in the following table:

Table 60. Sample data from DSN8910.EMP after Carlos's update

EMPNO	EMPNAME	DEPTNO	SECURITY
000190	BROWN	D11	HIGH
000200	JONES	D11	MEDIUM
000210	LUTZ	X55	LOW

Recommendation: To avoid failed updates, qualify the rows that you want to update with the following predicate, for the security label column SECLABEL:
WHERE SECLABEL=GETVARIABLE('SYSIBM.SECLABEL')

Using this predicate avoids failed updates because it ensures that the user's security label is equivalent to the security label of the rows that DB2 attempts to update.

Considerations for SELECT...FROM...UPDATE statements: If the user has write-down privilege or if the write-down control is not in effect, the security label of the user might not dominate the security label of the row. For statements that update rows and select the updated rows, the UPDATE statement succeeds.

However, the updated row is not returned. 

Using the MERGE statement with multilevel security

MERGE is an SQL statement that combines the conditional INSERT and UPDATE operations on a target table. Data that are not already present in the target table are inserted with the INSERT part of the MERGE statement. Data that are already present in the target table are updated with the UPDATE part of the MERGE statement.

 Because the MERGE statement consists of the INSERT and UPDATE operations, the multilevel security rules for the INSERT operation apply to the INSERT part of the MERGE statement and the multilevel security rules for the UPDATE operation apply to the UPDATE part of the MERGE statement.

For the INSERT part of the MERGE statement, when a user with a valid security label inserts data into a table with row-level security enabled, the security label of the row is determined according to the following rules:

- If the user has write-down privilege or if the write-down control is not enabled, the user can set the security label for the row to any valid security label. If the user does not specify a value for the security label, the security label of the row becomes the same as the security label of the user.
- If the user does not have write-down privilege and if the write-down control is enabled, the security label of the row becomes the same as the security label of the user.

For the UPDATE part of the MERGE statement, when a user with a valid security label updates a table with row-level security enabled, DB2 compares the security label of the user to the security label of the row. The update proceeds according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is updated and the value of the security label is determined by whether the user has write-down privilege:
 - If the user has write-down privilege or if the write-down control is not enabled, the user can set the security label of the row to any valid security label.
 - If the user does not have write-down privilege and if the write-down control is enabled, the security label of the row is set to the value of the security label of the user.
- If the security label of the user dominates the security label of the row, the result of the UPDATE operation is determined by whether the user has write-down privilege:
 - If the user has write-down privilege or if the write-down control is not enabled, the row is updated and the user can set the security label of the row to any valid security label.
 - If the user does not have write-down privilege and if the write-down control is enabled, the row is not updated.
- If the security label of the row dominates the security label of the user, the row is not updated.

Considerations for SELECT...FROM...MERGE statements: If the user has write-down privilege or if the write-down control is not in effect, the security label of the user might not dominate the security label of the row. For statements that merge rows and select the merged rows, the MERGE statement succeeds. However, the merged row is not returned. 

Using the DELETE statement with multilevel security

When a user with a valid security label deletes data from a table with row-level security enabled, DB2 compares the security label of the user to the security label of the row.

 The delete proceeds according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is deleted.
- If the security label of the user dominates the security label of the row, the user's write-down privilege determines the result of the DELETE statement:

- If the user has write-down privilege or write-down control is not enabled, the row is deleted.
- If the user does not have write-down privilege and write-down control is enabled, the row is not deleted.
- If the security label of the row dominates the security label of the user, the row is not deleted.

Example: Suppose that Alan has a security label of HIGH, that Beth has a security label of MEDIUM and write-down privilege defined in RACF, and that Carlos has a security label of LOW. Write-down control is enabled.

Suppose that DSN8910.EMP contains the data that is shown in the following table and that the SECURITY column has been declared with the AS SECURITY LABEL clause.

Table 61. Sample data from DSN8910.EMP

EMPNO	LASTNAME	WORKDEPT	SECURITY
000190	BROWN	D11	HIGH
000200	JONES	D11	MEDIUM
000210	LUTZ	D11	LOW

Now, suppose that Alan, Beth, and Carlos each submit the following DELETE statement:

```
DELETE FROM DSN8910.EMP
WHERE DEPTNO='D11';
```

Because Alan has a security label that dominates the rows with security labels of MEDIUM and LOW, his write-down privilege determines whether these rows are deleted. Alan does not have write-down privilege, so the delete fails for these rows. Because Alan has a security label that is equivalent to the security label of the row with HIGH security, the delete on that row succeeds. The results of Alan's delete are shown in the following table:

Table 62. Sample data from DSN8910.EMP after Alan's delete

EMPNO	EMPNAME	DEPTNO	SECURITY
000200	JONES	D11	MEDIUM
000210	LUTZ	D11	LOW

Because Beth has a security label that dominates the row with a security label of LOW, her write-down privilege determines whether this row is deleted. Beth has write-down privilege, so the delete succeeds for this row. Because Beth has a security label that is equivalent to the security label of the row with MEDIUM security, the delete succeeds for that row. Because the row with the security label of HIGH dominates Beth's security label, the delete fails for that row. The results of Beth's delete are shown in the following table:

Table 63. Sample data from DSN8910.EMP after Beth's delete

EMPNO	EMPNAME	DEPTNO	SECURITY
000190	BROWN	D11	HIGH

Because Carlos's security label is LOW, the delete fails for the rows with security labels of MEDIUM and HIGH. Because Carlos has a security label that is equivalent to the security label of the row with LOW security, the delete on that row succeeds. The results of Carlos's delete are shown in the following table:

Table 64. Sample data from DSN8910.EMP after Carlos's delete

EMPNO	EMPNAME	DEPTNO	SECURITY
000190	BROWN	D11	HIGH
000200	JONES	D11	MEDIUM

Important: Do not omit the WHERE clause from DELETE statements. If you omit the WHERE clause from the DELETE statement, checking occurs for rows that have security labels. This checking behavior might have a negative impact on performance.

Considerations for SELECT...FROM...DELETE statements: If the user has write-down privilege or write-down control is not in effect, the security label of the user might not dominate the security label of the row. For statements that delete rows and select the deleted rows, the DELETE statement succeeds. However, the deleted row is not returned. 

Using the TRUNCATE statement with multilevel security

When a user with a valid security label uses a TRUNCATE statement to delete all data from a table with row-level security enabled, DB2 compares the security label of the user to the security label of each row.

 The delete proceeds according to the following rules:

- If the security label of the user and the security label of the row are equivalent, the row is deleted.
- If the security label of the user dominates the security label of the row, the user's write-down privilege determines the result of the DELETE statement:
 - If the user has write-down privilege or write-down control is not enabled, the row is deleted.
 - If the user does not have write-down privilege and write-down control is enabled, the row is not deleted.
- If the security label of the row dominates the security label of the user, the row is not deleted.
- If the row cannot be deleted as a result of the security label verification, the TRUNCATE statement fails. 

Using utilities with multilevel security

You need a valid security label and additional authorizations to run certain LOAD, UNLOAD, and REORG TABLESPACE jobs on tables that have multilevel security enabled. All other utilities check only for authorization to operate on the table space; they do not check for row-level authorization.

 **LOAD:** You must have the write-down privilege to run LOAD REPLACE on a table space that contains a table with multilevel security enabled. In this case, you can specify the values for the security label column.

When you run LOAD RESUME, you must have the write-down privilege to specify values for the security label column. If you run a LOAD RESUME job and do not have the write-down privilege, DB2 assigns your security label as the value for each row in the security label column.

UNLOAD: Additional restrictions apply to UNLOAD jobs on tables that have multilevel security enabled. Each row is unloaded only if the security label of the user dominates the security label of the row. If security label of the user does not dominate the security label of the row, the row is not unloaded and DB2 does not issue an error message.

REORG TABLESPACE: REORG TABLESPACE jobs on tables that have multilevel security enabled have the following restrictions:

- For jobs with the UNLOAD EXTERNAL option, each row is unloaded only if the security label of the user dominates the security label of the row. If the security label of the user does not dominate the security label of the row, the row is not unloaded and DB2 does not issue an error message.
- For jobs with the DISCARD option, a qualifying row is discarded only if the user has the write-down privilege and the security label of the user dominates the security label of the row. 

Implementing multilevel security in a distributed environment

SQL statements that originate from remote requesters can participate in a multilevel secure environment if all information on the requester has the same security label and all users of the requester are permitted to that security label.

Management of multilevel security in a distributed environment requires physical control of the participating systems and careful management of the network. Managed systems must be prevented from communicating with other systems that do not have equivalent security labels.

Configuring TCP/IP with multilevel security

A communications server IP stack that runs in a multilevel secure environment can be configured as either a restricted stack or an unrestricted stack.

Recommendation: Use an unrestricted stack for DB2. An unrestricted stack is configured with an ID that is defined with a security label of SYSMULTI. A single z/OS system can concurrently run a mix of restricted and unrestricted stacks. Unrestricted stacks allow DB2 to use any security label to open sockets.

All users on a TCP/IP connection have the security label that is associated with the IP address that is defined on the server. If a user requires a different security label, the user must enter through an IP address that has that security label associated with it. If you require multiple IP addresses on a remote z/OS server, a workstation, or a gateway, you can configure multiple virtual IP addresses. This strategy can increase the number of security labels that are available on a client.

Remote users that access DB2 by using a TCP/IP network connection use the security label that is associated with the RACF SERVAUTH class profile when the remote user is authenticated. Security labels are assigned to the database access thread when the DB2 server authenticates the remote server by using the RACROUTE REQUEST = VERIFY service.

If you use a trusted context for your TCP/IP connection, you can define a default security label for all users or specific security labels for individual users who use

| the trusted context. The security label that is defined in the trusted context
| overrides the one for the TCP/IP connection in RACF.

Configuring SNA with multilevel security

Security labels are assigned to the database access thread when the DB2 server authenticates the remote server by using the RACROUTE REQUEST = VERIFY service. The service establishes a security label to the authorization ID that is associated with the database access thread. For SNA connections, this security label is the default security label that is defined for the remote user.

Chapter 6. Managing access through RACF

You can control whether a local or remote application can gain access to a specific DB2 subsystem from different environments. You can set different levels of security depending on whether the requesting application uses SNA or Transmission Control Protocol/Internet Protocol (TCP/IP) protocols to access DB2.

After the local system authenticates the incoming ID, it treats the ID like a local connection request or a local sign-on request. You can process the ID with your connection or sign-on exit routine and associate secondary authorization IDs with the ID. If you are sending a request to a remote DB2 subsystem, that subsystem can subject your request to various security checks.

You can use an external security system, such as RACF, IMS, or CICS, to authorize and authenticate a remote request before it reaches your DB2 subsystem. The discussion in the following topics assumes that you use RACF, or an equivalent system, for external access control.

Establishing RACF protection for DB2

You can install and use RACF to protect your DB2 resources.

Suppose that your DB2-RACF environment is exactly as shown in the following diagram.

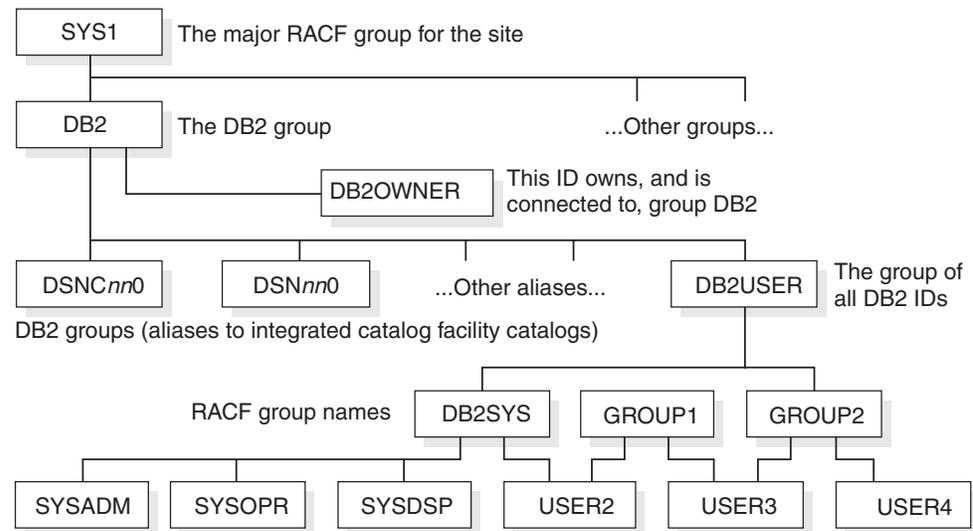


Figure 22. Sample DB2-RACF environment

Also, suppose that you use the system of RACF IDs, as listed in the following table, to control your DB2 usage:

Table 65. RACF relationships

RACF ID	Use
SYS1	Major RACF group ID
DB2	DB2 group

Table 65. RACF relationships (continued)

RACF ID	Use
DB2OWNER	Owner of the DB2 group
DSNC910	Group to control databases and recovery logs
DSN910	Group to control installation data sets
DB2USER	Group of all DB2 users
SYSADM	ID with DB2 installation SYSADM authority
SYSOPR	ID with DB2 installation SYSOPR authority
DB2SYS, GROUP1, GROUP2	RACF group names
SYSDSP	RACF user ID for DB2 started tasks
USER1, USER2, USER3	RACF user IDs.

You can perform the following tasks in any order to establish RACF protection for DB2:

- Define DB2 resources to RACF for protection.
- Grant RACF access to the protected DB2 resources.

Defining DB2 resources to RACF

To establish RACF protection for your DB2 subsystem, you must first define your DB2 resources to RACF and authorize RACF for authentication checking.

To define your DB2 resources to RACF:

- Define the names of protected access profiles.
- Enable RACF checking for the DSNR and SERVER classes.

You can also perform the following tasks:

- Control whether two DBMSs that use VTAM LU 6.2 can establish sessions with each other.
- Authorize IDs that are associated with stored procedures address spaces to run the appropriate attachment facility.
- Authorize the ID that is associated with the DDF address space to use z/OS UNIX[®] System Services if you use TCP/IP.

Naming protected access profiles

The RACF resource class for DB2 is DSNR, and the class is contained in the RACF class descriptor table. The profiles in that class help you control access to a DB2 subsystem from one of these environments: IMS, CICS, the distributed data facility (DDF), TSO, CAF, or batch.

Each profile has a name of the form *subsystem.environment*, where:

- *subsystem* is the name of a DB2 subsystem, of one to four characters; for example, DSN or DB2T.
- *environment* denotes the environment, by one of the following terms:
 - MASS for IMS (including MPP, BMP, Fast Path, and DL/I batch).
 - SASS for CICS.
 - DIST for DDF.
 - RRSAP for Resource Recovery Services attachment facility. Stored procedures use RRSAP in WLM-established address spaces.

- BATCH for all others, including TSO, CAF, batch, all utility jobs, and requests that come through the call attachment facility.

To control access, you need to define a profile, as a member of class DSNR, for every combination of subsystem and environment you want to use. For example, suppose that you want to access:

- Subsystem DSN from TSO and DDF
- Subsystem DB2P from TSO, DDF, IMS, and RRSF
- Subsystem DB2T from TSO, DDF, CICS, and RRSF

Then define the profiles with the following names:

```
DSN.BATCH  DSN.DIST
DB2P.BATCH DB2P.DIST DB2P.MASS DB2P.RRSF
DB2T.BATCH DB2T.DIST DB2T.SASS DB2T.RRSF
```

You can do that with a single RACF command, which also names an owner for the resources:

```
RDEFINE DSNR (DSN.BATCH DSN.DIST DB2P.BATCH DB2P.DIST DB2P.MASS DB2P.RRSF
              DB2T.BATCH DB2T.DIST DB2T.SASS DB2T.RRSF) OWNER(DB2OWNER)
```

In order to access a subsystem in a particular environment, a user must be on the access list of the corresponding profile. You add users to the access list by using the RACF PERMIT command. If you do not want to limit access to particular users or groups, you can give universal access to a profile with a command like this:

```
RDEFINE DSNR (DSN.BATCH) OWNER(DB2OWNER) UACC(READ)
```

Enabling RACF checking for the DSNR and SERVER classes

You need to allow RACF access to check for the DSNR and SERVER classes. Issue the following command to enable RACF access checking for resources in the DSNR resource class:

```
SETROPTS CLASSACT(DSNR)
```

If you are using stored procedures in a WLM-established address space, you might also need to enable RACF checking for the SERVER class.

Enabling partner LU verification

With RACF and VTAM, you can control whether two LUs that use LU 6.2 can connect to each other.

Each member of a connecting pair must establish a profile for the other member. For example, if LUAAA and LUBBB are to connect and know each other by those LUNAMES, issue RACF commands similar to these:

```
At LUAAA: RDEFINE APPCLU netid.LUAAA.LUBBB UACC(NONE) ...
At LUBBB: RDEFINE APPCLU netid.LUBBB.LUAAA UACC(NONE) ...
```

Here, *netid* is the network ID, given by the VTAM start option NETID.

When you create those profiles with RACF, use the SESSION operand to supply:

- The VTAM password as a session key (SESSKEY suboperand)
- The maximum number of days between changes of the session key (INTERVAL suboperand)
- An indication of whether the LU pair is locked (LOCK suboperand)

Finally, to enable RACF checking for the new APPCLU resources, issue this RACF command at both LUAAA and LUBBB:

Permitting RACF access

Perform the following tasks in the required order to enable a process to use the protected resources:

1. Define RACF user IDs for DB2-started tasks
2. Add RACF groups
3. Grant users and groups access

Defining RACF user IDs for DB2-started tasks

A DB2 subsystem has the following started-task address spaces:

- *ssnm*DBM1 for database services
- *ssnm*MSTR for system services
- *ssnm*DIST for the distributed data facility
- Names for your WLM-established address spaces for stored procedures

You must associate each of these address spaces with a RACF user ID. You can also assign each of them to a RACF group name. The RACF user IDs and group names that are associated with DB2 address spaces are listed in the following table:

Table 66. DB2 address spaces and associated RACF user IDs and group names

Address Space	RACF User ID	RACF Group Name
DSNMSTR	SYSDSP	DB2SYS
DSNDBM1	SYSDSP	DB2SYS
DSNDIST	SYSDSP	DB2SYS
DSNWLM	SYSDSP	DB2SYS
DB2TMSTR	SYSDSPT	DB2TEST
DB2TDBM1	SYSDSPT	DB2TEST
DB2TDIST	SYSDSPT	DB2TEST
DB2TSPAS	SYSDSPT	DB2TEST
DB2PMSTR	SYSDSPD	DB2PROD
DB2PDBM1	SYSDSPD	DB2PROD
DB2PDIST	SYSDSPD	DB2PROD
CICSSYS	CICS	CICSGRP
IMSCNTL	IMS	IMSGRP

You can use one of the two ways that RACF provides to associate user IDs and groups with started tasks: the STARTED class and the started procedures table (ICHRIN03). If you use the STARTED class, the changes take effect without a subsequent IPL. If you use ICHRIN03, you must perform another IPL for the changes to take effect. You cannot start the DB2 address spaces with batch jobs.

If you have IMS or CICS applications issuing DB2 SQL requests, you must associate RACF user IDs, and can associate group names, with:

- The IMS control region
- The CICS address space
- The four DB2 address spaces

If the IMS and CICS address spaces are started as batch jobs, provide their RACF IDs and group names with the USER and GROUP parameters on the JOB statement. If they are started as started-tasks, assign the IDs and group names as you do for the DB2 address spaces, by changing the RACF STARTED class or the RACF started procedures table.

The RACF user ID and group name do not need to match those that are used for the DB2 address spaces, but they must be authorized to run the Resource Recovery Services attachment facility (for WLM-established stored procedures address spaces). Note that the WLM-established stored procedures started tasks IDs require an OMVS segment.

If your installation has implemented the RACF STARTED class, you can use it to associate RACF user IDs and group names with the DB2 started procedures address spaces. If you have not previously set up the STARTED class, you first need to enable generic profile checking for the class:

```
SETROPTS GENERIC(STARTED)
```

Then, you need to define the RACF identities for the DB2 started tasks:

```
RDEFINE STARTED DSNMSTR.**  STDATA(USER(SYSDP)  GROUP(DB2SYS)  TRUSTED(NO))
RDEFINE STARTED DSNDBM1.**  STDATA(USER(SYSDP)  GROUP(DB2SYS)  TRUSTED(NO))
RDEFINE STARTED DSNDIST.**  STDATA(USER(SYSDP)  GROUP(DB2SYS)  TRUSTED(NO))
RDEFINE STARTED DSNWLM.**   STDATA(USER(SYSDP)  GROUP(DB2SYS)  TRUSTED(NO))
RDEFINE STARTED DB2TMSTR.** STDATA(USER(SYSDSPT) GROUP(DB2TEST) TRUSTED(NO))
...
```

Then, you need to activate the RACLIST processing to read the profiles into a data space:

```
SETROPTS CLASSACT(STARTED)
SETROPTS RACLIST(STARTED)
```

Lastly, you need to refresh the in-storage profiles:

```
SETROPTS RACLIST(STARTED) REFRESH
```

If you use the RACF-started procedures table (ICHRIN03) to associate RACF user IDs and group names with the DB2 started procedures address spaces, you need to change, reassemble, and link edit the resulting object code to z/OS. The following example shows a sample job that reassembles and link edits the RACF started-procedures table (ICHRIN03):

```
/**
/** REASSEMBLE AND LINKEDIT THE RACF STARTED-PROCEDURES
/** TABLE ICHRIN03 TO INCLUDE USERIDS AND GROUP NAMES
/** FOR EACH DB2 CATALOGED PROCEDURE. OPTIONALLY, ENTRIES
/** FOR AN IMS OR CICS SYSTEM MIGHT BE INCLUDED.
/**
/** AN IPL WITH A CLPA (OR AN MLPA SPECIFYING THE LOAD
/** MODULE) IS REQUIRED FOR THESE CHANGES TO TAKE EFFECT.
/**

ENTCOUNT DC    AL2(((ENDTABLE-BEGTABLE)/ENTLNTH)+32768)
*              NUMBER OF ENTRIES AND INDICATE RACF FORMAT
*
* PROVIDE FOUR ENTRIES FOR EACH DB2 SUBSYSTEM NAME.
*
BEGTABLE DS    0H
*              ENTRIES FOR SUBSYSTEM NAME "DSN"
              DC    CL8'DSNMSTR'          SYSTEM SERVICES PROCEDURE
              DC    CL8'SYSDSP'           USERID
              DC    CL8'DB2SYS'           GROUP NAME
```

	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
ENTLNGLTH	EQU	*-BEGTABLE	CALCULATE LENGTH OF EACH ENTRY
	DC	CL8'DSNDBM1'	DATABASE SERVICES PROCEDURE
	DC	CL8'SYSDSP'	USERID
	DC	CL8'DB2SYS'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
	DC	CL8'DSNDIST'	DDF PROCEDURE
	DC	CL8'SYSDSP'	USERID
	DC	CL8'DB2SYS'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
	DC	CL8'SYSDSP'	USERID
	DC	CL8'DB2SYS'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
	DC	CL8'DSNWLM'	WLM-ESTABLISHED S.P. ADDRESS SPACE
	DC	CL8'SYSDSP'	USERID
	DC	CL8'DB2SYS'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
*		ENTRIES FOR SUBSYSTEM NAME "DB2T"	
	DC	CL8'DB2TMSTR'	SYSTEM SERVICES PROCEDURE
	DC	CL8'SYSDSPT'	USERID
	DC	CL8'DB2TEST'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
	DC	CL8'DB2TDBM1'	DATABASE SERVICES PROCEDURE
	DC	CL8'SYSDSPT'	USERID
	DC	CL8'DB2TEST'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
	DC	CL8'DB2TDIST'	DDF PROCEDURE
	DC	CL8'SYSDSPT'	USERID
	DC	CL8'DB2TEST'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
	DC	CL8'SYSDSPT'	USERID
	DC	CL8'DB2TEST'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
*		ENTRIES FOR SUBSYSTEM NAME "DB2P"	
	DC	CL8'DB2PMSTR'	SYSTEM SERVICES PROCEDURE
	DC	CL8'SYSDSPD'	USERID
	DC	CL8'DB2PROD'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
	DC	CL8'DB2PDBM1'	DATABASE SERVICES PROCEDURE
	DC	CL8'SYSDSPD'	USERID
	DC	CL8'DB2PROD'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
	DC	CL8'DB2PDIST'	DDF PROCEDURE
	DC	CL8'SYSDSPD'	USERID
	DC	CL8'DB2PROD'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
	DC	CL8'SYSDSPD'	USERID
	DC	CL8'DB2PROD'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE
	DC	XL7'00'	RESERVED BYTES
*		OPTIONAL ENTRIES FOR CICS AND IMS CONTROL REGION	
	DC	CL8'CICSSYS'	CICS PROCEDURE NAME
	DC	CL8'CICS'	USERID
	DC	CL8'CICSGRP'	GROUP NAME
	DC	X'00'	NO PRIVILEGED ATTRIBUTE

```

DC    XL7'00'          RESERVED BYTES
DC    CL8'IMSCNTL'    IMS CONTROL REGION PROCEDURE
DC    CL8'IMS'        USERID
DC    CL8'IMSGRP'     GROUP NAME
DC    X'00'          NO PRIVILEGED ATTRIBUTE
DC    XL7'00'          RESERVED BYTES
ENDTABLE DS    0D
      END

```

The example shows the sample entries for three DB2 subsystems (DSN, DB2T, and DB2P), optional entries for CICS and IMS, and DB2 started tasks for the DB2 subsystems, CICS, the IMS control region.

Adding RACF groups

Issue the following RACF command to add the user DB2OWNER:

```
ADDUSER DB2OWNER CLAUTH(DSNR USER) UACC(NONE)
```

That gives class authorization to DB2OWNER for DSNR and USER. DB2OWNER can add users to RACF and issue the RDEFINE command to define resources in class DSNR. DB2OWNER has control over and responsibility for the entire DB2 security plan in RACF.

The RACF group SYS1 already exists. To add group DB2 and make DB2OWNER its owner, issue the following RACF command:

```
ADDGROUP DB2 SUPGROUP(SYS1) OWNER(DB2OWNER)
```

To connect DB2OWNER to group DB2 with the authority to create new subgroups, add users, and manipulate profiles, issue the following RACF command:

```
CONNECT DB2OWNER GROUP(DB2) AUTHORITY(JOIN) UACC(NONE)
```

To make DB2 the default group for commands issued by DB2OWNER, issue the following RACF command:

```
ALTUSER DB2OWNER DFLTGRP(DB2)
```

To create the group DB2USER and add five users to it, issue the following RACF commands:

```
ADDGROUP DB2USER SUPGROUP(DB2)
ADDUSER (USER1 USER2 USER3 USER4 USER5) DFLTGRP(DB2USER)
```

To define a user to RACF, use the RACF ADDUSER command. That invalidates the current password. You can then log on as a TSO user to change the password.

DB2 considerations when using RACF groups:

- When a user is newly connected to, or disconnected from, a RACF group, the change is not effective until the next logon. Therefore, before using a new group name as a secondary authorization ID, a TSO user must log off and log on, or a CICS or IMS user must sign on again.
- A user with the SPECIAL, JOIN, or GROUP-SPECIAL RACF attribute can define new groups with any name that RACF accepts and can connect any user to them. Because the group name can become a secondary authorization ID, you should control the use of those RACF attributes.
- Existing RACF group names can duplicate existing DB2 authorization IDs. That duplication is unlikely for the following reasons:
 - A group name cannot be the same as a user name.
 - Authorization IDs that are known to DB2 are usually known to RACF.

However, you can create a table with an owner name that is the same as a RACF group name and use the IBM-supplied sample connection exit routine. Then any TSO user with the group name as a secondary ID has ownership privileges on the table. You can prevent that situation by designing the connection exit routine to stop unwanted group names from being passed to DB2.

Granting users and groups access

Suppose that the DB2OWNER group in the following example is authorized for class DSNR, owns the profiles, and has the right to change them. You can issue the following commands to authorize the DB2USER members, the system administrators, and operators to be TSO users.

These users can run batch jobs and DB2 utilities on the three systems: DSN, DB2P, and DB2T. The ACCESS(READ) operand allows use of DB2 without the ability to manipulate profiles.

```
PERMIT DSN.BATCH CLASS(DSNR) ID(DB2USER) ACCESS(READ)
PERMIT DB2P.BATCH CLASS(DSNR) ID(DB2USER) ACCESS(READ)
PERMIT DB2T.BATCH CLASS(DSNR) ID(DB2USER) ACCESS(READ)
```

Defining profiles for IMS and CICS: You want the IDs for attaching systems to use the appropriate access profile. For example, to let the IMS user ID use the access profile for IMS on system DB2P, issue the following RACF command:

```
PERMIT DB2P.MASS CLASS(DSNR) ID(IMS) ACCESS(READ)
```

To let the CICS group ID use the access profile for CICS on system DB2T, issue the following RACF command:

```
PERMIT DB2T.SASS CLASS(DSNR) ID(CICSGRP) ACCESS(READ)
```

Providing installation authorities to default IDs: When DB2 is installed, IDs are named to have special authorities—one or two IDs for SYSADM and one or two IDs for SYSOPR. Those IDs can be connected to the group DB2USER; if they are not, you need to give them access. The next command permits the default IDs for the SYSADM and SYSOPR authorities to use subsystem DSN through TSO:

```
PERMIT DSN.BATCH CLASS(DSNR) ID(SYSADM,SYSOPR) ACCESS(READ)
```

Using secondary IDs: You can use secondary authorization IDs to define a RACF group. After you define the RACF group, you can assign privileges to it that are shared by multiple primary IDs. For example, suppose that DB2OWNER wants to create a group GROUP1 and to give the ID USER1 administrative authority over the group. USER1 should be able to connect other existing users to the group. To create the group, DB2OWNER issues this RACF command:

```
ADDGROUP GROUP1 OWNER(USER1) DATA('GROUP FOR DEPT. G1')
```

To let the group connect to the DSN system through TSO, DB2OWNER issues this RACF command:

```
PERMIT DSN.BATCH CLASS(DSNR) ID(GROUP1) ACCESS(READ)
```

USER1 can now connect other existing IDs to the group GROUP1 by using the RACF CONNECT command:

```
CONNECT (USER2 EPSILON1 EPSILON2) GROUP(GROUP1)
```

If you add or update secondary IDs for CICS transactions, you must start and stop the CICS attachment facility to ensure that all threads sign on and get the correct security information.

Allowing users to create data sets: You can use RACF to protect the data sets that store DB2 data. If you use the approach and when you create a new group of DB2 users, you might want to connect it to a group that can create data sets. To allow USER1 to create and control data sets, DB2OWNER creates a generic profile and permits complete control to USER1 and to the four administrators. The SYSDSP parameter also gives control to DB2.

```
ADDSO 'DSNC910.DSNDBC.ST*' UACC(NONE)
```

```
PERMIT 'DSNC910.DSNDBC.ST*'
      ID(USER1 SYSDSP SYSAD1 SYSAD2 SYSOP1 SYSOP2) ACCESS(ALTER)
```

Granting authorization on DB2 commands

IDs must be authorized to issue DB2 commands. If you authorize IDs by issuing DB2 GRANT statements, the GRANT statements must be made to a primary authorization ID, a secondary authorization ID, a role, or PUBLIC.

When RACF is used for access control, an ID must have appropriate RACF authorization on DB2 commands or must be granted authorization for DB2 commands to issue commands from a logged-on MVS console or from TSO SDSF.

You can ensure that an ID can issue DB2 commands from logged-on MVS consoles or TSO SDSF by using one of the following methods:

- Grant authorization for DB2 commands to the primary, secondary authorization ID, or role.
- Define RACF classes and permits for DB2 commands.
- Grant SYSOPR authority to appropriate IDs.

Permitting access from remote requesters

You can use the DSNR RACF class with a PERMIT command to access the distributed data address space, such as DSN.DIST, to control access from remote requesters.

The following RACF commands let the users in the group DB2USER access DDF on the DSN subsystem. These DDF requests can originate from any partner in the network.

Example: To permit READ access on profile DSN.DIST in the DSNR class to DB2USER, issue the following RACF command:

```
PERMIT DSN.DIST CLASS(DSNR) ID(DB2USER) ACCESS(READ)
```

If you want to ensure that a specific user can access only when the request originates from a specific LU name, you can use WHEN(APPCPORT) on the PERMIT command.

Example: To permit access to DB2 distributed processing on subsystem DSN when the request comes from USER5 at LUNAME equal to NEWYORK, issue the following RACF command:

```
PERMIT DSN.DIST CLASS(DSNR) ID(USER5) ACCESS(READ) +
      WHEN(APPCPORT(NEWYORK))
```

For connections that come through TCP/IP, use the RACF APPCPORT class or the RACF SERVAUTH class with TCP/IP Network Access Control to protect unauthorized access to DB2.

Example: To use the RACF APPCPORT class, perform the following steps:

1. Activate the ACCPORT class by issuing the following RACF command:

```
SETROPTS CLASSACT(APPCPORT) REFRESH
```
2. Define the general resource profile and name it TCPIP. Specify NONE for universal access and APPCPORT for class. Issue the following RACF command:

```
RDEFINE APPCPORT (TCPIP) UACC(NONE)
```
3. Permit READ access on profile TCPIP in the APPCPORT class. To permit READ access to USER5, issue the following RACF command:

```
PERMIT TCPIP ACCESS(READ) CLASS(APPCPORT) ID(USER5)
```
4. Permit READ access on profile DSN.DIST in the DSNR class. To permit READ access to USER5, issue the following RACF command:

```
PERMIT DSN.DIST CLASS(DSNR) ID(USER5) ACCESS(READ) +  
WHEN(APPCPORT(TCPIP))
```
5. Refresh the APPCPORT class by issuing the following RACF command:

```
SETROPTS CLASSACT(APPCPORT) REFRESH RACLIST(APPCPORT)
```

If the RACF APPCPORT class is active on your system, and a resource profile for the requesting LU name already exists, you must permit READ access to the APPCPORT resource profile for the user IDs that DB2 uses. You must permit READ access even when you are using the DSNR resource class. Similarly, if you are using the RACF APPL class and that class restricts access to the local DB2 LU name or generic LU name, you must permit READ access to the APPL resource for the user IDs that DB2 uses.

Recommendation: Use z/OS Communications Server IP Network Access Control and z/OS Security Server RACF SERVAUTH class if you want to use the port of entry (POE) for remote TCP/IP connections.

Requirement: To use the RACF SERVAUTH class and TCP/IP Network Access Control, you must have z/OS V1.5 (or later) installed.

Example: To use the RACF SERVAUTH class and TCP/IP Network Access Control, perform the following steps:

1. Set up and configure TCP/IP Network Access Control by using the NETACCESS statement that is in your TCP/IP profile.

For example, suppose that you need to allow z/OS system access only to IP addresses from 9.0.0.0 to 9.255.255.255. You want to define these IP addresses as a security zone, and you want to name the security zone IBM. Suppose also that you need to deny access to all IP addressed outside of the IBM security zone, and that you want to define these IP addresses as a separate security zone. You want to name this second security zone WORLD. To establish these security zones, use the following NETACCESS clause:

```
NETACCESS INBOUND OUTBOUND
; NETWORK/MASK      SAF
  9.0.0.0/8         IBM
  DEFAULT           WORLD
ENDNETACCESS
```

Now, suppose that USER5 has an IP address of 9.1.2.3. TCP/IP Network Access Control would determine that USER5 has an IP address that belongs to the IBM security zone. USER5 would be granted access to the system. Alternatively, suppose that USER6 has an IP address of 1.1.1.1. TCP/IP Network Access Control would determine that USER6 has an IP address that belongs to the WORLD security zone. USER6 would not be granted access to the system.

2. Activate the SERVAUTH class by issuing the following TSO command:

```
SETROPTS CLASSACT(SERVAUTH)
```

3. Activate RACLIST processing for the SERVAUTH class by issuing the following TSO command:

```
SETROPTS RACLIST(SERVAUTH)
```
4. Define the IBM and WORLD general resource profiles in RACF to protect the IBM and WORLD security zones by issuing the following commands:

```
RDEFINE SERVAUTH (EZB.NETACCESS.ZOSV1R5.TCPIP.IBM) UACC(NONE)
RDEFINE SERVAUTH (EZB.NETACCESS.ZOSV1R5.TCPIP.WORLD) UACC(NONE)
```
5. Permit USER5 and SYSDSP read access to the IBM profile by using the following commands.

```
PERMIT EZB.NETACCESS.ZOSV1R5.TCPIP.IBM ACCESS READ CLASS(SERVAUTH) ID(USER5)
PERMIT EZB.NETACCESS.ZOSV1R5.TCPIP.IBM ACCESS READ CLASS(SERVAUTH) ID(SYSDSP)
```
6. Permit SYSDSP read access to the WORLD profile by using the following command:

```
PERMIT EZB.NETACCESS.ZOSV1R5.TCPIP.WORLD ACCESS READ CLASS(SERVAUTH) ID(USER5)
```
7. For these permissions to take effect, refresh the RACF database by using the following command:

```
SETROPTS CLASSACT(SERVAUTH) REFRESH RACLIST(SERVAUTH)
```

Protecting stored procedures

You can set up RACF to protect your stored procedures that run on a DB2 subsystem.

Perform the following tasks in the recommended order to establish RACF protection for your stored procedures.

1. Control access through the RRSAF (Required)
2. Control access to WLM (Optional)
3. Control access to non-DB2 resources (Optional)

Managing access through RRSAF (Required)

You need to authorize the ID that is associated with the WLM-established stored procedures address space to run Resource Recovery Services attachment facility (RRSAF). Make sure that the ID is also associated with the *ssnm*.RRSAF profile that you create.

To control access to the DB2 subsystem through RRSAF:

1. Create a *ssnm*.RRSAF profile
 You can define *ssnm*.RRSAF in the DSNR resource class with a universal access authority of NONE by issuing the following command:

```
RDEFINE DSNR (DB2P.RRSAF DB2T.RRSAF) UACC(NONE)
```
2. Issue the following command to refresh the in-storage profiles with the profiles you just defined:

```
SETROPTS RACLIST(DSNR) REFRESH
```
3. Add user IDs that are associated with the stored procedures address spaces to the RACF-started procedures table, as shown in the following example:

```
RDEFINE STARTED DSNWLM.** STDATA(USER(SYSDP) GROUP(DB2SYS) TRUSTED(NO))
```
4. Refresh the in-storage profiles by issuing the following command:

```
SETROPTS RACLIST(STARTED) REFRESH
```
5. Issue the following command to grant read access to *ssnm*.RRSAF to the ID that is associated with the stored procedures address space:

```
PERMIT DB2P.RRSAF CLASS(DSNR) ID(SYSDSP) ACCESS(READ)
```

Managing access to WLM

You can use the SERVER resource class to manage the address spaces that can be WLM-established server address spaces and run stored procedures.

If you don't define or activate the SERVER resource class, you allow any address space to connect to WLM as a server address space and to identify itself as one that runs stored procedures.

To use the SERVER resource class:

1. Define a RACF profile for the SERVER resource class, as follows:

```
RDEFINE SERVER (DB2.ssnm.applenv)
```

applenv is the name of the application environment that is associated with the stored procedure.

Assume that you want to define the following profile names:

- DB2.DB2T.TESTPROC
- DB2.DB2P.PAYROLL
- DB2.DB2P.QUERY

Use the following RACF command:

```
RDEFINE SERVER (DB2.DB2T.TESTPROC DB2.DB2P.PAYROLL DB2.DB2P.QUERY)
```

2. Refresh the in-storage profiles with the ones you just defined:

```
SETROPTS RACLIST(SERVER) REFRESH
```

3. Grant read access to the server resource name to the IDs that are associated with the stored procedures address space as follows.

```
PERMIT DB2.DB2T.TESTPROC CLASS(SERVER) ID(SYSDSP) ACCESS(READ)
PERMIT DB2.DB2P.PAYROLL CLASS(SERVER) ID(SYSDSP) ACCESS(READ)
PERMIT DB2.DB2P.QUERY CLASS(SERVER) ID(SYSDSP) ACCESS(READ)
```

Managing stored procedures in a WLM environment

Programs can be grouped together and isolated in different WLM environments based on application security requirements. For example, payroll applications might be isolated in one WLM environment because they contain sensitive data, such as employee salaries.

To prevent users from creating a stored procedure in a sensitive WLM environment, DB2 invokes RACF to determine if the user is allowed to do so. The WLM ENVIRONMENT keyword on the CREATE PROCEDURE statement identifies the WLM environment to use for running a given stored procedure. Attempts to create a stored procedure fail if the user is not properly authorized.

DB2 performs a resource authorization check using the DSNR RACF class as follows:

- In a DB2 data sharing environment, DB2 uses the following RACF resource name:
`db2_groupname.WLMENV.wlm_environment`
- In a non-data sharing environment, DB2 checks the following RACF resource name:
`db2_subsystem_id.WLMENV.wlm_environment`

You can use the RACF RDEFINE command to create RACF profiles that prevent users from creating stored procedures and user-defined functions in sensitive WLM environments. For example, you can prevent all users on DB2 subsystem DB2A

(non-data sharing) from creating a stored procedure or user-defined function in the WLM environment named PAYROLL; to do this, use the following command:
RDEFINE DSNR (DB2A.WLMENV.PAYROLL) UACC(NONE)

The RACF PERMIT command authorizes a user to create a stored procedure or user-defined function in a WLM environment. For example, you can authorize a DB2 user (DB2USER1) to create stored procedures on DB2 subsystem DB2A (non-data sharing) in the WLM environment named PAYROLL:

```
PERMIT DB2A.WLMENV.PAYROLL CLASS(DSNR) ID(DB2USER1) ACCESS(READ)
```

Managing access to non-DB2 resources

With WLM-established address spaces, you can control access to non-DB2 resources by the authorization ID of the caller, instead of that of the stored procedure address space. You can do so by specifying EXTERNAL SECURITY USER on the CREATE PROCEDURE statement to control .

Use EXTERNAL SECURITY USER only when the caller must access resources outside of DB2. When you specify EXTERNAL SECURITY USER, a separate RACF environment is established for that stored procedure. Make sure to enable the RACF checking for the caller's ID.

To control access to non-DB2 resources:

1. Use the ALTER PROCEDURE statement with the EXTERNAL SECURITY USER clause.
2. Ensure that the ID of the stored procedure's caller has RACF authority to the resources.
3. For improved performance, specify the following keywords in the COFVLFxx member of library SYS1.PARMLIB to cache the RACF profiles in the virtual look-aside facility (VLF) of z/OS:

```
CLASS NAME(IRRACEE)  
EMAJ(ACEE)
```

Protecting connection requests that use the TCP/IP protocol

You can set your DB2 subsystem to send or receive connection requests that use the TCP/IP network protocol. You need to authorize the started task user ID (SYSDSP) that is associated with the DB2 distributed address space (ssnmDIST) to use the z/OS UNIX system services.

To secure connection requests over TCP/IP:

1. Create an OMVS segment in the RACF user profile for the started task user ID (SYSDSP)
2. Specify a z/OS UNIX user identifier of 0 and the maximum number of files of that the user is allowed to have concurrently active to 131702 in the following command:

```
ADDUSER ddfuid OMVS(UID(0) FILEPROCMAX(131702))
```

If the *ddfuid* ID already exists, use:

```
ALTUSER ddfuid OMVS(UID(0) FILEPROCMAX(131702))
```

The started task user ID of the DB2 distributed address space only needs a z/OS UNIX user identifier of 0 (UID(0)). A UID 0 is considered a superuser. If you don't want to grant the superuser authority to the started task user ID that is associated with the ssnmDIST address space during the DB2 installation, you can specify a value other than 0 for the UID parameter. Make sure that the value is a valid z/OS UNIX user identifier.

3. If you want to assign a z/OS group name to the address space, assign an OMVS segment to the z/OS group name by using one of the following RACF commands:

```
ADDGROUP ddfgmn OMVS(GID(nnn))...
```

```
ALTGROUP ddfgmn OMVS(GID(nnn))...
```

where *ddfgnm* is the z/OS group name and *nnn* is any valid, unique identifier.

The standard way to assign a z/OS userid and a z/OS group name to a started address space is to use the z/OS Security Server (RACF) STARTED resource class. This method enables you to dynamically assign a z/OS user ID by using commands instead of requiring an IPL to have the assignment take effect. The alternative method to assign a z/OS user ID and a z/OS group name to a started address space is to change the RACF started procedures table, ICHRIN03.

You can also manage TCP/IP requests in a trusted context. A trusted context allows you to use a trusted connection without needing additional authentication and to acquire additional privileges through the definition of roles.

The TCP/IP Already Verified (DSN6FAC TCPALVER) controls whether DB2 accepts TCP/IP connection requests that contain only a user ID. However, in the case of a trusted context, it is the definition of the trusted context, not the TCPALVER setting, handles the requirement for switching users of a trusted connection.

Do not set DSN6FAC TCPALVER to YES if you use a trusted context. If you set TCPALVER to YES in the definition of the trusted context, you need to define the authorization ID that establishes the trusted connection in the USER clause to enforce the authentication requirement.

Establishing Kerberos authentication through RACF

Kerberos security is a network security technology that was developed at the Massachusetts Institute of Technology. The Kerberos security technology does not require passwords to flow in readable text because it uses encrypted tickets that contain authentication information for the users.

DB2 can use Kerberos security services to authenticate remote users. With Kerberos security services, remote users need to issue their Kerberos name and password to access DB2. They can use the same name and password for access throughout the network, which makes a separate password to access DB2 unnecessary.

A remote user who is authenticated to DB2 by means of Kerberos authentication must be registered in RACF profiles. An organization that runs a Kerberos server establishes its own *realm*. The name of the realm in which a client is registered is part of the client's name and can be used by the application server to accept or reject a request.

To authenticate and register a remote user in RACF profiles:

1. Define the Kerberos realm to RACF by issuing the following command:

```
RDEFINE REALM KERBDFLT KERB(KERBNAME(localrealm) PASSWORD(mykerpw)
```

You must specify the name of the local realm in the definition. You must also specify a Kerberos password for RACF to grant Kerberos tickets.

2. Define local principals to RACF by issuing the following command:

```
AU RONTOMS KERB(KERBNAME(rontoms))  
ALU RONTOMS PASSWORD(new1pw) NOEXPIRE
```

Make sure to change RACF passwords before the principals become active Kerberos users.

3. Map foreign Kerberos principals by defining KERBLINK profiles to RACF with a command similar to the following:

```
RDEFINE KERBLINK /.../KERB390.ENDICOTT.IBM.COM/RWH APPLDATA('RONTOMS')
```

You must also define a principal name for the user ID that is used in the *ssnmDIST* started task address space, as shown in the following example:

```
ALU SYSDSP PASSWORD(pw) NOEXPIRE KERB(KERBNAME(SYSDSP))
```

The *ssnmDIST* address space must have the RACF authority to use its SAF ticket parsing service. The user ID that is used for the *ssnmDIST* started task address space is SYSDSP.

4. Define foreign Kerberos authentication servers to the local Kerberos authentication server by issuing the following command:

```
RDEFINE REALM /.../KERB390.ENDICOTT.IBM.COM/KRBTGT/KER2000.ENDICOTT.IBM.COM +  
KERB(PASSWORD(realmpw))
```

You must supply a password for the key to be generated. REALM profiles define the trust relationship between the local realm and the foreign Kerberos authentication servers. PASSWORD is a required keyword, so all REALM profiles have a KERB segment.

Data sharing environment: Data sharing Sysplex environments that use Kerberos security must have a Kerberos Security Server instance running on each system in the Sysplex. The instances must either be in the same realm and share the same RACF database, or have different RACF databases and be in different realms.

Implementing DB2 support for enterprise identity mapping

Enterprise identity mapping (EIM) enables the mapping of user identities across servers that are integrated but that do not share user registries. DB2 supports the EIM capability by implementing the SAF user mapping plug-in callable service, which is part of the z/OS V1.8 Security Server (RACF).

You can exploit the EIM support by using the IBM Websphere Application Server 6.0.1, the IBM DB2 Driver for JDBC and SQLJ, and the IBM DB2 Driver for ODBC and CLI.

You must install z/OS V1.8 or later to use the SAF user mapping plug-in service and implement the DB2 support for the EIM.

To implement the DB2 support for EIM:

1. Configure the z/OS LDAP server with a TDBM backend
2. Set up RACF for the LDAP server
3. Configure the z/OS EIM domain controller
4. Add the SAF user mapping data set to LNKLIST

If you enable DB2 support for EIM, DB2 can retrieve the mapped user ID from the SAF user mapping plug-in and specify the information in the ICTX structure. During the ENVIR=CREATE processing, DB2 passes the information to RACF through the RACROUTE REQUEST=VERIFY macro service. When RACF successfully authenticates the user, the ICTX structure is anchored in the ACEEICTX field.

Related information

- z/OS Integrated Security Services LDAP Server Administration and Use
- z/OS Integrated Security Services LDAP Client Programming
- z/OS Security Server RACF Command Language Reference
- z/OS Integrated Security Services Enterprise Identity Mapping (EIM) Guide and Reference

Configuring the z/OS LDAP server

When DB2 receives an authenticated user registry name, it invokes the SAF user mapping plug-in service. This service uses the EIM domain, which is an LDAP server, to retrieve the z/OS user ID that is used as the primary authorization ID.

You can use the LDAP configuration (ldapcnf) utility to configure and set up a z/OS LDAP server. The LDAP configuration utility requires the ldap.profile input file that is shipped in the /usr/lpp/ldap/etc directory. The ldap.profile file contains the settings that you need to set up the LDAP server.

To configure a z/OS LDAP server:

1. Copy and modify the ldap.profile file based on your own environment.
2. Issue the following command to run the LDAP configuration utility with the ldap.profile file that you modified:

```
ldapcnf -i ldap.profile
```

The LDAP configuration utility generates the following output files:

- SLAPDCNF member as the LDAP server configuration file
- SLAPDENV member as the LDAP server environment variable file
- PROG member for APF authorization
- GLDSRV procedure for starting the LDAP server
- DSNAOINI configuration file for DB2 CLI
- TDBSPUFI DB2 SQL DDL statements for creating the TDBM environment
- DBCLI DB2 SQL BIND statements for binding the CLI/ODBC packages and plan
- RACF member for creating the RACF profiles that protect the LDAP server service task and grant permissions for the user ID to run the LDAP server

These output files are stored in the OUTPUT_DATASET_NAME that you specified in the ldap.profile file.

3. Submit the following output JCL files after DB2 is started:
 - DBCLI member file
 - RACF member file
4. Submit the TDBSPUFI member file by using the DB2 SPUFI interactive tool.
5. Start the LDAP server from SDSF or the operator's console.

The name of the LDAP server procedure file is the same as the user ID that is specified on the LDAPUSRID statement. The pre-assigned value is GLDSRV.

To start the LDAP server from SDSF, enter:

```
/s GLDSRV
```

To start the LDAP server from the operator's console, enter:

```
s GLDSRV
```

6. Copy the schema.user.ldif file from the /usr/lpp/ldap/etc directory to a local directory

7. Use the following ldapmodify utility to modify the schema entry for the TDBM backend

```
ldapmodify -h ldaphost -p ldapport -D binddn -w passwd -f file
```

The following example shows how to use the ldapmodify utility:

```
ldapmodify -h v25ec099.svl.ibm.com -p 3389
-D "cn=LDAP Administrator"
-w secret -f schema.user.ldif
```

At the top of the schema.user.ldif file, find the following line, and supply the appropriate TDBM suffix in that line

```
dn: cn=schema, <suffix>
```

The suffix is the same value that is used in the TDBM_SUFFIX statement in the ldap.profile file, as in the following example:

```
dn: cn=schema, o=IBM, c=US
```

8. Use the ldapadd utility to load the suffix entry and to create a user ID that is used by the SAF user mapping plug-in for binding with the LDAP server. You can use the following ldapadd utility statement:

```
ldapadd -h ldaphost -p ldapport -D binddn -w passwd -f file
```

The following is an example of using the ldapadd utility:

```
ldapadd -h v25ec099.svl.ibm.com -p 3389
-D "cn=LDAP Administrator"
-w secret -f setup.ldap.ldif
```

Setting up RACF for the z/OS LDAP server

After you configure the z/OS LDAP server, you need to set up RACF to activate identity mapping and grant DB2 authority to use the SAF user mapping plug-in service.

To set up RACF for the z/OS LDAP server:

1. Enable identity mapping by activating the FACILITY class.

The FACILITY class must be active to enable identity mapping. Use the following SETROPTS command if it is not already active at your installation:

```
SETROPTS CLASSACT(FACILITY)
```

2. Define a KEYMSTR profile to store an encryption key.

Make sure to choose a key that is known only to the security administrator, and store it in the KEYMSTR profile that you defined, as shown in the following example:

```
RDEF KEYMSTR LDAP.BINDPW.KEY SSIGNON(KEYMASKED(0123456789ABCDEF))
```

The LDAP BIND passwords are encrypted with the key that is stored in the LDAP.BINDPW.KEY profile. The value of the key in this example is 0123456789ABCDEF.

3. Authorize DB2 to request lookup services by defining and granting READ access to the SYSDSP user in the following RACF profiles:

```
RDEF FACILITY IRR.RGETINFO.EIM UACC(NONE)
PE IRR.RGETINFO.EIM ACCESS(READ) ID(SYSDSP) CL(FACILITY)
```

```
RDEF FACILITY IRR.RDCEKEY UACC(NONE)
PE IRR.RDCEKEY ACCESS(READ) ID(SYSDSP) CL(FACILITY)
```

4. Define the IRR.PROXY.DEFAULTS profile in the FACILITY class, as follows:

```
RDEF FACILITY IRR.PROXY.DEFAULTS
PROXY(LDAPHOST('ldap://v25ec099.svl.ibm.com:3389'))
BINDDN('cn=eim user,o=IBM,c=US') BINDPW('secret')
```

```
EIM(DOMAINDN('ibm-eimDomainName=My Domain,o=IBM,c=US')
LOCALREG('My Target Registry'))
```

```
SETOPTS RACLIST(FACILITY) REFRESH
```

5. Grant DB2 the authority to use the SAF user mapping plug-in service by issuing the following commands:

```
RDEF PROGRAM IRRSPIM ADDMEM ('USER.PRIVATE.DLLLIB'//NOPADCHK)
PE IRRSPIM ACCESS(READ) ID(SYSDSP) CL(PROGRAM)
```

```
RDEF PROGRAM IRRSPIME ADDMEM ('USER.PRIVATE.DLLLIB'//NOPADCHK)
PE IRRSPIME ACCESS(READ) ID(SYSDSP) CL(PROGRAM)
```

```
SETOPTS WHEN(PROGRAM) REFRESH
```

Setting up the EIM domain controller

After you set up the LDAP server and RACF, you need to use the `eimadmin` utility to create and configure an EIM domain controller.

Suppose that you want to create an EIM domain controller on a z/OS system with two registries: `MySourceRegistry` and `MyTargetRegistry`. The `MyTargetRegistry` registry contains a mapping of a z/OS user ID, `Buffy`, which is returned to DB2 as the primary authorization ID.

To create an EIM domain controller in this situation:

1. Create an EIM domain by issuing the following command:

```
eimadmin -aD -d 'ibm-eimDomainName=My Domain,o=IBM,c=US'
-h ldap://v25ec099.svl.ibm.com:3389
-b "cn=LDAP Administrator" -w secret
```

The example shows that the new domain name is "My Domain." It also shows that the `TDBM_SUFFIX` statement in the `ldap.profile` file is defined as `o=IBM,c=US`.

2. Grant the EIM user access to the EIM domain for performing lookup services by issuing the following command:

```
eimadmin -aC -c MAPPING -q "cn=eim user, o=IBM, c=US" -f DN
-d 'ibm-eimDomainName=My Domain,o=IBM,c=US'
-h ldap://v25ec099.svl.ibm.com:3389
-b 'cn=LDAP Administrator' -w secret
```

3. Create the source registry in the EIM domain by issuing the following command:

```
eimadmin -aR -r "My Source Registry" -y KERBEROS
-d 'ibm-eimDomainName=My Domain,o=IBM,c=US'
-h ldap://v25ec099.svl.ibm.com:3389
-b 'cn=LDAP Administrator' -w secret
```

4. Create the target registry in the EIM domain by issuing the following command:

```
eimadmin -aR -r "My Target Registry" -y RACF
-d 'ibm-eimDomainName=My Domain,o=IBM,c=US'
-h ldap://v25ec099.svl.ibm.com:3389
-b 'cn=LDAP Administrator' -w secret
```

5. Add the enterprise identifier "Cat" to the EIM domain by issuing the following command:

```
eimadmin -aI -i "Cat" -d 'ibm-eimDomainName=My Domain,o=IBM,c=US'
-h ldap://v25ec099.svl.ibm.com:3389
-b 'cn=LDAP Administrator' -w secret
```

You can add multiple enterprise identifiers to the same EIM domain at any time.

6. Associate registry user IDs with the identifiers in the EIM domain by issuing the following commands:

```
eimadmin -aA -u "Kitty" -r "My Source Registry" -t SOURCE
-i "Cat" -d 'ibm-eimDomainName=My Domain,o=IBM,c=US'
-h ldap://v25ec099.svl.ibm.com:3389
-b 'cn=LDAP Administrator' -w secret
```

```
eimadmin -aA -u "Buffy" -r "My Target Registry" -t TARGET
-o "db2/stlec1/v12ec099.svl.ibm.com" -i "Cat"
-d 'ibm-eimDomainName=My Domain,o=IBM,c=US'
-h ldap://v25ec099.svl.ibm.com:3389
-b 'cn=LDAP Administrator' -w secret
```

Specify the "-o" flag with the "db2/location-name/domain-name" value when you define a user ID for DB2 to use as the primary authorization ID in your target registry. As the examples show, when DB2 calls the SAF user mapping plug-in service to retrieve the primary authorization ID, DB2 specifies the additional 'db2/location-name/domain-name' information for the plug-in service to look up.

If a target identity is found with the same information, the target identity "Buffy" is returned. If the target identity does not contain any additional information, user ID "Buffy" is also returned to DB2. However, if the target registry contains multiple user identities and if none of them contains the recommended additional information, no user identity is returned to DB2.

Adding the SAF user mapping plug-in data set to LNKLIST

The SAF user mapping plug-in IRRSPIME resides in a z/OS data set. This data set must be included in the LNKLIST. If the data set is not included, you need to add it to the LNKLIST.

To add the z/OS data set that contains the SAF user mapping plug-in to the LNKLIST:

1. Define a new LNKLIST by issuing the following command from the operator console:

```
SETPROG LNKLIST,DEFINE,NAME=MYLNKLIST,COPYFROM=CURRENT
```

2. Add the USER.PRIVATE.DLLLIB data set on the USER01 volume to the new MYLNKLIST by issuing the following command:

```
SETPROG LNKLIST,ADD,NAME=MYLNKLIST,DSNAME=USER.PRIVATE.DLLLIB,
VOLUME=USER01
```

3. Activate the new MYLNKLIST by issuing the following command:

```
SETPROG LNKLIST,ACTIVATE,NAME=MYLNKLIST
```

4. Use the MYLNKLIST to update the current tasks in the system by issuing the following command:

```
SETPROG LNKLIST,UPDATE,JOB=*
```

Managing connection requests from local applications

Different local processes enter the access control procedure at different points, depending on the environment in which they originate.

The following processes go through connection processing only:

- Requests originating in TSO foreground and background (including online utilities and requests through the call attachment facility)
- JES-initiated batch jobs

- Requests through started task control address spaces (from the z/OS START command)

The following processes go through connection processing and can later go through the sign-on processing:

- The IMS control region.
- The CICS recovery coordination task.
- DL/I batch.
- Applications that connect using the Resource Recovery Services attachment facility (RRSAF).

The following processes go through sign-on processing:

- Requests from IMS dependent regions (including MPP, BMP, and Fast Path)
- CICS transaction subtasks

IMS, CICS, RRSAF, and DDF-to-DDF connections can send a sign-on request, typically to execute an application plan. That request must provide a primary ID, and can also provide secondary IDs. After a plan is allocated, it need not be deallocated until a new plan is required. A different transaction can use the same plan by issuing a new sign-on request with a new primary ID.

Processing of connection requests

A connection request makes a new connection to DB2; it does not reuse an application plan that is already allocated. Therefore, an essential step in processing the request is to check that the ID is authorized to use DB2 resources.

DB2 completes the following steps to process a connection request:

1. DB2 obtains the initial primary authorization ID. As shown in the following table, the source of the ID depends on the type of address space from which the connection was made.

Table 67. Sources of initial primary authorization IDs

Source	Initial primary authorization ID
TSO	TSO logon ID.
BATCH	USER parameter on JOB statement.
IMS control region or CICS	USER parameter on JOB statement.
IMS or CICS started task	Entries in the started task control table.
Remote access requests	Depends on the security mechanism used.

2. RACF is called through the z/OS system authorization facility (SAF) to check whether the ID that is associated with the address space is authorized to use the following resources:

- The DB2 resource class (CLASS=DSNR)
- The DB2 subsystem (SUBSYS=*ssnm*)
- The requested connection type

The SAF return code (RC) from the invocation determines the next step, as follows:

- **If RC > 4**, RACF determined that the RACF user ID is not valid or does not have the necessary authorization to access the resource name. DB2 rejects the request for a connection.
- **If RC = 4**, the RACF return code is checked.

- If RACF return code value is **equal to 4**, the resource name is not defined to RACF and DB2 rejects the request with reason code X'00F30013'.
 - If RACF return code value is **not equal to 4**, RACF is not active. DB2 continues with the next step, but the connection request and the user are not verified.
 - **If RC = 0**, RACF is active and has verified the RACF user ID; DB2 continues with the next step.
3. If RACF is active and has verified the RACF user ID, DB2 runs the connection exit routine. To use DB2 secondary IDs, you must replace the exit routine. If you do not want to use secondary IDs, do nothing. The IBM-supplied default connection exit routine continues the connection processing. The process has the following effects:
- The DB2 primary authorization ID is set based on the following rules:
 - If a value for the initial primary authorization ID exists, the value becomes the DB2 primary ID.
 - If no value exists (the value is blank), the primary ID is set by default, as shown in the following table.

Table 68. Sources of default authorization identifiers

Source	Default primary authorization ID
TSO	TSO logon ID
BATCH	USER parameter on JOB statement
Started task, or batch job with no USER parameter	Default authorization ID set when DB2 was installed (UNKNOWN AUTHID on installation panel DSNTIIP)
Remote request	None. The user ID is required and is provided by the DRDA requester.

- The SQL ID is set equal to the primary ID.
 - No secondary IDs exist.
4. DB2 determines if TSO and BATCH connections that use DSN, RRSAP, and Utilities are trusted.
- For a TSO and BATCH connection that uses DSN, RRSAP, and Utilities, DB2 checks to see if a matching trusted context is defined for the primary authorization ID and the job name. If a matching trusted context is found, the connection is established as trusted.

Related concepts

“Connection routines and sign-on routines” on page 753

Related tasks

“Using sample connection and sign-on exit routines for CICS transactions” on page 273

“Specifying connection and sign-on routines” on page 754

“Debugging connection and sign-on routines” on page 762

Using secondary IDs for connection requests

If you want to use DB2 secondary authorization IDs, you must replace the default connection exit routine. If you want to use RACF group names as DB2 secondary IDs, the easiest method is to use the IBM-supplied sample routine.

The following table lists the difference between the default and sample connection exit routines.

Table 69. Differences between the default and sample connection exit routines

Default connection exit routine	Sample connection exit routine
Supplied as object code.	Supplied as source code. You can change the code.
Installed as part of the normal DB2 installation procedure.	Must be compiled and placed in the DB2 library.
Provides values for primary IDs and SQL IDs, but does not provide values for secondary IDs.	Provides values for primary IDs, secondary IDs, and SQL IDs.

The sample connection exit routine has the following effects:

- The sample connection exit routine sets the DB2 primary ID in the same way that the default routine sets the DB2 primary ID, and according to the following rules:
 - If the initial primary ID is not blank, the initial ID becomes the DB2 primary ID.
 - If the initial primary ID is blank, the sample routine provides the same default value as does the default routine.
 - If the sample routine cannot find a nonblank primary ID, DB2 uses the default ID (UNKNOWN AUTHID) from the DSNTIPP installation panel. In this case, no secondary IDs are supplied.
- The sample connection exit routine sets the SQL ID based on the following criteria:
 - The routine sets the SQL ID to the TSO data set name prefix in the TSO user profile table if the following conditions are true:
 - The connection request is from a TSO-managed address space, including the call attachment facility, the TSO foreground, and the TSO background.
 - The TSO data set name prefix is equal to the primary ID or one of the secondary IDs.
 - In all other cases, the routine sets the SQL ID equal to the primary ID.
- The secondary authorization IDs depend on RACF options:
 - If RACF is not active, no secondary IDs exist.
 - If RACF is active but its “list of groups” option is not active, one secondary ID exists (the default connected group name) if the attachment facility supplied the default connected group name.
 - If RACF is active and the “list of groups” option is active, the routine sets the list of DB2 secondary IDs to the list of group names to which the RACF user ID is connected. Those RACF user IDs that are in REVOKE status do not become DB2 secondary IDs. The maximum number of groups is 1012. The list of group names is obtained from RACF and includes the default connected group name.

If the default connection exit routine and the sample connection exit routine do not provide the flexibility and features that your subsystem requires, you can write your own exit routine.

Processing of sign-on requests

For requests from IMS-dependent regions, CICS transaction subtasks, or RRS connections, the initial primary ID is not obtained until just before allocating a

plan for a transaction. A new sign-on request can run the same plan without de-allocating the plan and reallocating it. Nevertheless, it can change the primary ID.

Unlike the connection processing, the sign-on processing does not check the RACF for the user ID of the address space. DB2 completes the following steps to process sign-on requests:

1. DB2 determines the initial primary ID as follows:

For IMS sign-ons from message-driven regions, if the user has signed on, the initial primary authorization ID is the user's sign-on ID. IMS passes to DB2 the IMS sign-on ID and the associated RACF connected group name, if one exists. If the user has not signed on, the primary ID is the LTERM name, or if that is not available, the PSB name. For a batch-oriented region, the primary ID is the value of the USER parameter on the job statement, if that is available. If that is not available, the primary ID is the program's PSB name.

For remote requests, the source of the initial primary ID is determined by entries in the SYSIBM.USERNAMES table. For connections using Resource Recovery Services attachment facility, the processing depends on the type of signon request:

- SIGNON
- AUTH SIGNON
- CONTEXT SIGNON

For SIGNON, the primary authorization ID is retrieved from ACEEUSRI if an ACEE is associated with the TCB (TCBSENV). This is the normal case. However, if an ACEE is not associated with the TCB, SIGNON uses the primary authorization ID that is associated with the address space, that is, from the ASXB. If the new primary authorization ID was retrieved from the ACEE that is associated with the TCB and ACEEGRPN is not null, DB2 uses ACEEGRPN to establish secondary authorization IDs.

With AUTH SIGNON, an APF-authorized program can pass a primary authorization ID for the connection. If a primary authorization ID is passed, AUTH SIGNON also uses the value that is passed in the secondary authorization ID parameter to establish secondary authorization IDs. If the primary authorization ID is not passed, but a valid ACEE is passed, AUTH SIGNON uses the value in ACEEUSRI for the primary authorization ID if ACEEUSRI is not 0. If ACEEUSRI is used for the primary authorization ID, AUTH SIGNON uses the value in ACEEGRPN as the secondary authorization ID if ACEEGRPL is not 0.

For CONTEXT SIGNON, the primary authorization ID is retrieved from data that is associated with the current RRS context using the context_key, which is supplied as input. CONTEXT SIGNON uses the CTXSDTA and CTXRDTA functions of RRS context services. An authorized function must use CTXSDTA to store a primary authorization ID prior to invoking CONTEXT SIGNON. Optionally, CTXSDTA can be used to store the address of an ACEE in the context data that has a context_key that was supplied as input to CONTEXT SIGNON. DB2 uses CTXRDTA to retrieve context data. If an ACEE address is passed, CONTEXT SIGNON uses the value in ACEEGRPN as the secondary authorization ID if ACEEGRPL is not 0.

2. DB2 runs the sign-on exit routine. **User action:** To use DB2 secondary IDs, you must replace the exit routine.

If you **do not** want to use secondary IDs, do nothing. Sign-on processing is then continued by the IBM-supplied default sign-on exit routine, which has the following effects:

- The initial primary authorization ID remains the primary ID.

- The SQL ID is set equal to the primary ID.
- No secondary IDs exist.

You can replace the exit routine with one of your own, even if it has nothing to do with secondary IDs. If you do, remember that IMS and CICS recovery coordinators, their dependent regions, and RRSAF take the exit routine only if they have provided a user ID in the sign-on parameter list.

3. DB2 determines if the user of a trusted RRSAF SIGNON connection is allowed to switch.

For a RRSAF SIGNON connection that is trusted, DB2 checks to see if the primary authorization ID is allowed to switch in the trusted connection. If the primary authorization ID is not allowed to switch, the connection is returned to the unconnected state.

Related concepts

“Connection routines and sign-on routines” on page 753

Related tasks

“Using sample connection and sign-on exit routines for CICS transactions” on page 273

“Specifying connection and sign-on routines” on page 754

“Debugging connection and sign-on routines” on page 762

Using secondary IDs for sign-on requests

If you want the primary authorization ID to be associated with DB2 secondary authorization IDs, you must replace the default sign-on exit routine.

The procedure is similar to that for connection processing. If you want to use RACF group names as DB2 secondary IDs, the easiest method is to use the IBM-supplied sample routine. An installation job can automatically replace the default routine with the sample routine.

Distinguish carefully between the two routines. The default sign-on routine provides no secondary IDs and has the following effects:

- The initial primary authorization ID remains the primary ID.
- The SQL ID is set equal to the primary ID.
- No secondary IDs exist.

Like the sample connection routine, the sample sign-on routine supports DB2 secondary IDs and has the following effects:

- The initial primary authorization ID is left unchanged as the DB2 primary ID.
- The SQL ID is made equal to the DB2 primary ID.
- The secondary authorization IDs depend on RACF options:
 - If RACF is not active, no secondary IDs exist.
 - If RACF is active but its “list of groups” option is not active, one secondary ID exists; it is the name passed by CICS or by IMS.
 - If RACF is active and you have selected the option for a list of groups, the routine sets the list of DB2 secondary IDs to the list of group names to which the RACF user ID is connected, up to a limit of 1012 groups. The list of group names includes the default connected groupname.

Using sample connection and sign-on exit routines for CICS transactions

For a CICS transaction to use the sample connection or sign-on exit routines, the external security system, such as RACF, must be defined to CICS with the following specifications:

- The CICS system initialization table must specify external security.
 - For CICS Version 4 or later, specify SEC=YES.
 - For earlier releases of CICS, specify EXTSEC=YES.

If you are using the CICS multiple region option (MRO), you must specify SEC=YES or EXTSEC=YES for every CICS system that is connected by interregion communication (IRC).

- If your version of CICS uses a sign-on table (SNT), the CICS sign-on table must specify EXTSEC=YES for each signed on user that uses the sign-on exit.
- When the user signs on to a CICS terminal-owning region, the terminal-owning region must propagate the authorization ID to the CICS application-owning region.

You must change the sample sign-on exit routine (DSN3SSGN) before using it if the following conditions are all true:

- You have the RACF list-of-groups option active.
- You have transactions whose initial primary authorization ID is not defined to RACF.

Related concepts

“Connection routines and sign-on routines” on page 753

Related reference

“Processing of connection requests” on page 268

“Processing of sign-on requests” on page 270

“Sample connection and sign-on routines” on page 754

“Exit parameter list for connection and sign-on routines” on page 756

Managing connection requests from remote applications

If you are controlling requests from remote applications, your DB2 subsystem might be accepting requests from applications that use SNA network protocols, TCP/IP network protocols, or both.

Security mechanisms for DRDA and SNA

SNA and DRDA have different security mechanisms. DRDA allows a user to be authenticated using SNA security mechanisms or DRDA mechanisms, which are independent of the underlying network protocol.

For an SNA network connection, a DRDA requester can send security tokens by using a SNA attach or a DRDA command. DB2 for z/OS as a requester uses SNA security mechanisms if it uses a SNA network connection (except for Kerberos) and DRDA security mechanisms for TCP/IP network connections (or when Kerberos authentication is chosen, regardless of the network type).

Security mechanisms for DB2 for z/OS as a requester

DB2 for z/OS as a requester chooses SNA or DRDA security mechanisms based on the network protocol and the authentication mechanisms that you use.

If you use SNA protocols, DB2 supports the following SNA authentication mechanisms:

- User ID only (already verified)
- User ID and password
- User ID and PassTicket

Authentication is performed based on SNA protocols, which means that the authentication tokens are sent in an SNA attach (FMH-5).

If you use TCP/IP protocols, DB2 supports the following DRDA authentication mechanisms:

- User ID only (already verified)
- User ID and password
- User ID and PassTicket

If you use TCP/IP protocols with the z/OS Integrated Cryptographic Service Facility, DB2 also supports the following DRDA authentication mechanisms:

- Encrypted user ID and encrypted password
- Encrypted user ID and encrypted security-sensitive data

Authentication is performed based on DRDA protocols, which means that the authentication tokens are sent in DRDA security flows.

Security mechanisms for DB2 for z/OS as a server

As a server, DB2 for z/OS can accept either SNA or DRDA authentication mechanisms. This means that DB2 can authenticate remote users from either the security tokens obtained from the SNA ATTACH (FMH-5) or from the DRDA security commands described by each of the protocols.

The following authentication methods are supported by DB2 for z/OS as a server:

- User ID only (already verified at the requester)
- User ID and password
- User ID and PassTicket
- Kerberos tickets
- Unencrypted user ID and encrypted password
- Encrypted user ID and encrypted password
- User ID, password, and new password

DB2 for z/OS as a server also supports the following authentication mechanisms if the z/OS Integrated Cryptographic Service Facility is installed and active:

- Encrypted user ID and encrypted security-sensitive data
- Encrypted user ID, encrypted password, and encrypted security-sensitive data
- Encrypted user ID, encrypted password, encrypted new password, and encrypted security-sensitive data
- Encrypted user ID, encrypted password, encrypted new password, and encrypted security-sensitive data

Communications database for the server

The *communications database* (CDB) is a set of DB2 catalog tables that let you control aspects of how requests leave this DB2 and how requests come in. Columns in the SYSIBM.LUNAMES and SYSIBM.USERNAMES tables pertain to security on the inbound side (the server).

SYSIBM.LUNAMES columns

The SYSIBM.LUNAMES table is used only for requests that use SNA protocols.

GUPI

LUNAME CHAR(8)

The LUNAME of the remote system. A blank value identifies a default row that serves requests from any system that is not specifically listed elsewhere in the column.

SECURITY_IN CHAR(1)

The *acceptance option* for a remote request from the corresponding LUNAME:

- V** The option is “verify.” An incoming request must include one of the following authentication entities:
- User ID and password
 - User ID and RACF PassTicket
 - User ID and RACF encrypted password (not recommended)
 - Kerberos security tickets
 - User ID and DRDA encrypted password
 - User ID, password, and new password
 - User ID and encrypted password, or encrypted user ID and encrypted password
- A** The option is “already verified.” This is the default. With A, a request does not need an authentication token, although the token is checked if it is sent.

With this option, an incoming connection request is accepted if it includes any of the following authentication tokens:

- User ID only
- All authentication methods that option V supports

If the USERNAMES column of SYSIBM.LUNAMES contains I or B, RACF is not invoked to validate incoming connection requests that contain only a user ID.

ENCRYPTPSWDS CHAR(1)

This column only applies to DB2 for z/OS or DB2 for z/OS partners when passwords are used as authentication tokens. It indicates whether passwords received from and sent to the corresponding LUNAME are encrypted:

- Y** Yes, passwords are encrypted. For outbound requests, the encrypted password is extracted from RACF and sent to the server. For inbound requests, the password is treated as if it is encrypted.
- N** No, passwords are not encrypted. This is the default; any character other than Y is treated as N. Specify N for CONNECT statements that contain a USER parameter.

Recommendation: When you connect to a DB2 for z/OS partner that is at Version 5 or a subsequent release, use RACF PassTickets (SECURITY_OUT='R') instead of using passwords.

USERNAMES CHAR(1)

This column indicates whether an ID accompanying a remote request, sent from or to the corresponding LUNAME, is subject to translation and “come

from” checking. When you specify I, O, or B, use the SYSIBM.USERNAMES table to perform the translation.

- I** An inbound ID is subject to translation.
- O** An outbound ID, sent to the corresponding LUNAME, is subject to translation.
- B** Both inbound and outbound IDs are subject to translation.
- blank** No IDs are translated.

GUPI

SYSIBM.USERNAMES columns

The SYSIBM.USERNAMES table is used by both SNA and TCP/IP connections.

GUPI

TYPE CHAR(1)

Indicates whether the row is used for inbound or outbound translation:

- S** The row is used to obtain the system authorization ID for establishing a trusted connection.
- I** The row applies to inbound IDs (not applicable for TCP/IP connections).
- O** The row applies to outbound IDs.

The field should contain only I or O. Any other character, including blank, causes the row to be ignored.

AUTHID VARCHAR(128)

An authorization ID that is permitted and perhaps translated. If blank, any authorization ID is permitted with the corresponding LINKNAME; all authorization IDs are translated in the same way. Outbound translation is not performed on CONNECT statements that contain an authorization ID for the value of the USER parameter.

LINKNAME CHAR(8)

Identifies the VTAM or TCP/IP network locations that are associated with this row. A blank value in this column indicates that this name translation rule applies to any TCP/IP or SNA partner.

If you specify a nonblank value for this column, one or both of the following situations must be true:

- A row exists in table SYSIBM.LUNAMES that has an LUNAME value that matches the LINKNAME value that appears in this column.
- A row exists in table SYSIBM.IPNAMES that has a LINKNAME value that matches the LINKNAME value that appears in this column.

NEWAUTHID VARCHAR(128)

The translated authorization ID. If blank, no translation occurs.

GUPI

Enabling change of user passwords

You can specify YES in the EXTENDED SECURITY field of the DSNTIPR installation panel so that DB2 can return to the DRDA requester information about errors and expired passwords.

When the DRDA requester is notified that the RACF password has expired, and the requester has implemented function to allow passwords to be changed, the requester can prompt the end user for the old password and a new password. The requester sends the old and new passwords to the DB2 server. This function is supported through DB2[®] Connect[™].

With the extended security option, DB2 passes the old and new passwords to RACF. If the old password is correct, and the new password meets the installation's password requirements, the end user's password is changed and the DRDA connection request is honored.

When a user changes a password, the user ID, the old password, and the new password are sent to DB2 by the client system. The client system can optionally encrypt these three tokens before they are sent.

Authorization failure code

If the DB2 server is installed with YES for the EXTENDED SECURITY field of the DSNTIPR installation panel, detailed reason codes are returned to a DRDA client when a DDF connection request fails because of security errors.

When using SNA protocols, the requester must have included support for extended security sense codes. One such product is DB2 Connect.

If the proper requester support is present, the requester generates SQLCODE -30082 (SQLSTATE '08001') with a specific indication for the failure. Otherwise, a generic security failure code is returned.

Managing inbound SNA-based connection requests

Requests from a remote LU are subject to two security checks before they come into contact with DB2. Those checks control what LUs can attach to the network and verify the identity of a partner LU.

In addition, DB2 itself imposes several checks before accepting an attachment request.

If using private protocols, the LOCATIONS table controls the locations that can access DB2. To allow a remote location to access DB2, the remote location name must be specified in the SYSIBM.LOCATIONS table. This check is only supported for connections using private protocols.

Processing of remote attachment requests

The DB2 server completes the following sequence of authentication process before accepting a remote attachment request that uses the SNA protocol.

1. As the following diagram shows, if the remote request has no authentication token, DB2 checks the security acceptance option in the SECURITY_IN column of table SYSIBM.LUNAMES. No password is sent or checked for the plan or package owner that is sent from a DB2 subsystem.

Activity at the DB2 server

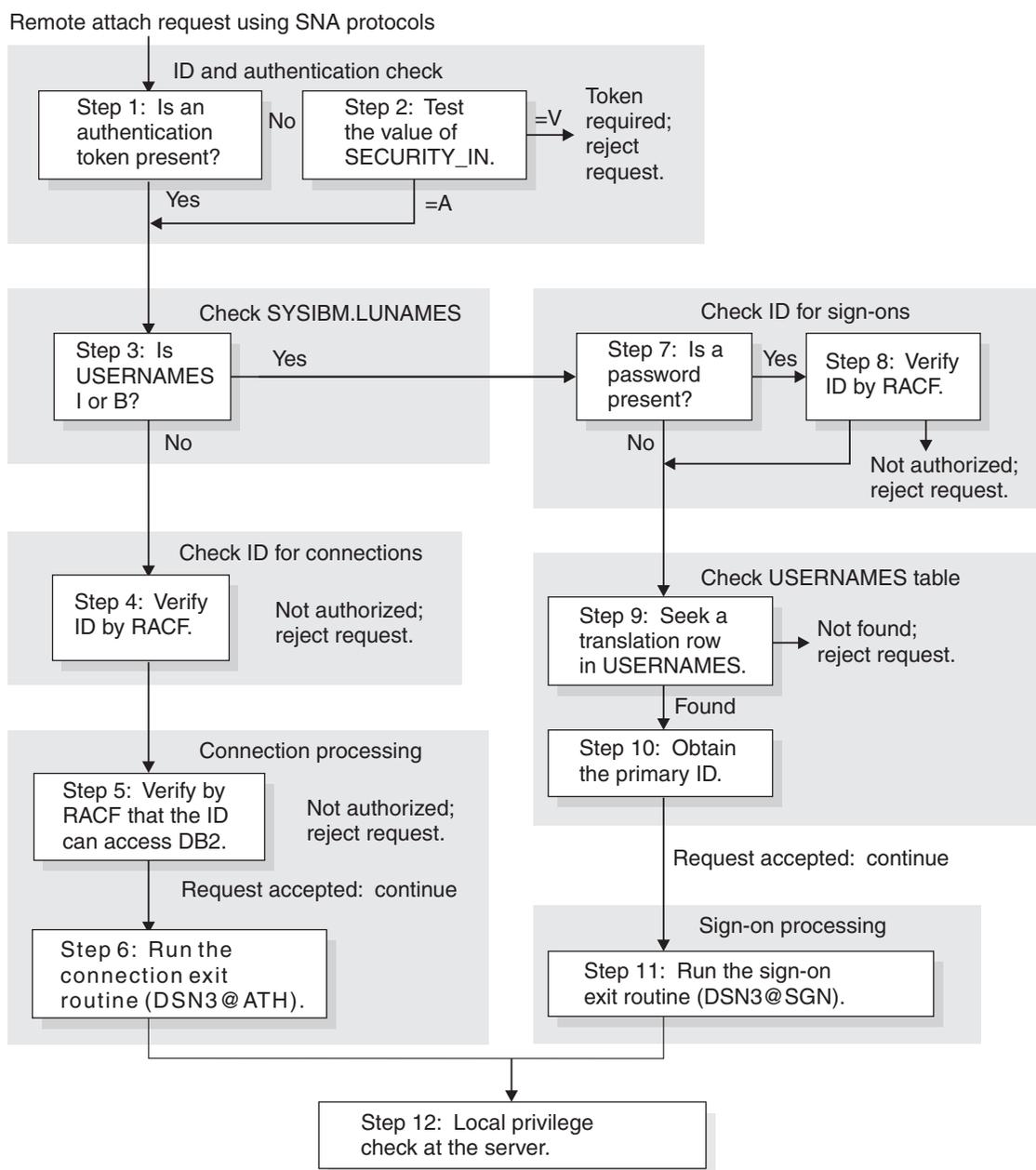


Figure 23. DB2 processing of remote attachment requests

2. If the acceptance option is “verify” (SECURITY_IN = V), a security token is required to authenticate the user. DB2 rejects the request if the token missing.
3. If the USERNAMES column of SYSIBM.LUNAMES contains I or B, the authorization ID, and the plan or package owner that is sent by a DB2 subsystem, are subject to translation under control of the SYSIBM.USERNAMES table. If the request is allowed, it eventually goes through sign-on processing. If USERNAMES does not contain I or B, the authorization ID is not translated.
4. DB2 calls RACF by the RACROUTE macro with REQUEST=VERIFY to check the ID. DB2 uses the PASSCHK=NO option if no password is specified and

ENCRYPT=YES if the ENCRYPTSWDS column of SYSIBM.LUNAMES contains Y. If the ID, password, or PassTicket cannot be verified, DB2 rejects the request.

In addition, depending on your RACF environment, the following RACF checks may also be performed:

- If the RACF APPL class is active, RACF verifies that the ID has been given access to the DB2 APPL. The APPL resource that is checked is the LU name that the requester used when the attachment request was issued. This is either the local DB2 LU name or the generic LU name.
 - If the RACF APPCPORT class is active, RACF verifies that the ID is authorized to access z/OS from the Port of Entry (POE). The POE that RACF uses in the verify call is the requesting LU name.
5. The remote request is now treated like a local connection request with a DIST environment for the DSNR resource class. DB2 calls RACF by the RACROUTE macro with REQUEST=AUTH, to check whether the authorization ID is allowed to use DB2 resources that are defined to RACF.
The RACROUTE macro call also verifies that the user is authorized to use DB2 resources from the requesting system, known as the port of entry (POE).
 6. DB2 invokes the connection exit routine. The parameter list that is passed to the routine describes where a remote request originated.
 7. If no password exists, RACF is not called. The ID is checked in SYSIBM.USERNAMES.
 8. If a password exists, DB2 calls RACF through the RACROUTE macro with REQUEST=VERIFY to verify that the ID is known with the password. ENCRYPT=YES is used if the ENCRYPTSWDS column of SYSIBM.LUNAMES contains Y. If DB2 cannot verify the ID or password, the request is rejected.
 9. DB2 searches SYSIBM.USERNAMES for a row that indicates how to translate the ID. The need for a row that applies to a particular ID and sending location imposes a “come-from” check on the ID: If no such row exists, DB2 rejects the request.
 10. If an appropriate row is found, DB2 translates the ID as follows:
 - If a nonblank value of NEWAUTHID exists in the row, that value becomes the primary authorization ID.
 - If NEWAUTHID is blank, the primary authorization ID remains unchanged.
 11. The remote request is now treated like a local sign-on request. DB2 invokes the sign-on exit routine. The parameter list that is passed to the routine describes where a remote request originated.
 12. The remote request now has a primary authorization ID, possibly one or more secondary IDs, and an SQL ID. A request from a remote DB2 is also known by a plan or package owner. Privileges and authorities that are granted to those IDs at the DB2 server govern the actions that the request can take.

Controlling LU attachments to the network

VTAM checks to prevent an unauthorized LU from attaching to the network and presenting itself to other LUs as an acceptable partner in communication. It requires each LU that attaches to the network to identify itself by a password.

If that requirement is in effect for your network, your DB2 subsystem, like every other LU on the network, must:

1. Choose a VTAM password.

2. Code the password with the PRTCT parameter of the VTAM APPL statement, when you define your DB2 to VTAM.

Verifying partner LUs

RACF and VTAM check the identity of an LU sending a request to your DB2.

Perform the following steps to specify partner-LU verification:

1. Code VERIFY=REQUIRED on the VTAM APPL statement, when you define your DB2 to VTAM.
2. Establish a RACF profile for each LU from which you permit a request.

Accepting remote attachment requests

When VTAM has established a conversation for a remote application, that application sends a remote request, which is a request to attach to your local DB2.

Make sure that you do not confuse the remote request with a local attachment request that comes through one of the DB2 attachment facilities—IMS, CICS, TSO, and so on. A remote attachment request is defined by Systems Network Architecture and LU 6.2 protocols; specifically, it is an SNA Function Management Header 5.

In order to accept remote attachment requests, you must first define your DB2 to VTAM with the conversation-level security set to “already verified”. That is, you need to code SECACPT=ALREADYV on the VTAM APPL statement. The SECACPT=ALREADYV setting provides more options than does SECACPT=CONV or “conversation”, which is not recommended.

The primary tools for controlling remote attachment requests are entries in tables SYSIBM.LUNAMES and SYSIBM.USERNAMES in the communications database. You need a row in SYSIBM.LUNAMES for each system that sends attachment requests, a dummy row that allows any system to send attachment requests, or both. You might need rows in SYSIBM.USERNAMES to permit requests from specific IDs or specific LUNAMES, or to provide translations for permitted IDs.

Managing inbound IDs through DB2

If you manage incoming IDs through DB2, you can avoid calls to RACF and specify acceptance of many IDs by a single row in the SYSIBM.USERNAMES table.

To manage incoming IDs through DB2, put an I in the USERNAMES column of SYSIBM.LUNAMES for the particular LU. If an O is already specified because you are also sending requests to that LU, change O to B. Attachment requests from that LU now go through the sign-on processing, and its IDs are subject to translation.

Managing inbound IDs through RACF

If you manage incoming IDs through RACF, you must register every acceptable ID with RACF, and DB2 must call RACF to process every request.

You can use RACF or Kerberos can authenticate the user. Kerberos cannot be used if you do not have RACF on the system.

To manage incoming IDs through RACF, leave USERNAMES blank for that LU, or leave the O unchanged, if already specified. Requests from that LU now go through the connection processing, and its IDs are not subject to translation.

Authenticating partner LUs

Presumably, RACF has already validated the identity of the other LU. If you trust incoming IDs from that LU, you do not need to validate them by an authentication token.

Put an A in the SECURITY_IN column of the row in SYSIBM.LUNAMES that corresponds to the other LU; your acceptance level for requests from that LU is now “already verified”. Requests from that LU are accepted without an authentication token. (In order to use this option, you must have defined DB2 to VTAM with SECACPT=ALREADYV.

If an authentication token does accompany a request, DB2 calls RACF to check the authorization ID against it. To require an authentication token from a particular LU, put a V in the SECURITY_IN column in SYSIBM.LUNAMES; your acceptance level for requests from that LU is now “verify”. You must also register every acceptable incoming ID and its password with RACF.

Performance considerations: Each request to RACF to validate authentication tokens results in an I/O operation, which has a high performance cost.

Recommendation: To eliminate the I/O, allow RACF to cache security information in VLF. To activate this option, add the IRRACEE class to the end of z/OS VLF member COFVLFxx in SYS1.PARMLIB, as follows:

```
CLASS NAME(IRRACEE)
EMAJ (ACEE)
```

Encrypting passwords

You can encrypt passwords by using one of the following methods:

- RACF using PassTickets.
- DRDA password encryption support. DB2 for z/OS as a server supports DRDA encrypted passwords and encrypted user IDs with encrypted passwords.
- The SET ENCRYPTION PASSWORD statement. This encryption method should not be used for distributed access because the unencrypted passwords in the SET ENCRYPTION PASSWORD statement flow from the client to the server.

Authenticating users through Kerberos

If your distributed environment uses Kerberos to manage users and perform user authentication, DB2 for z/OS can use Kerberos security services to authenticate remote users.

Translating inbound IDs

Ideally, each of your authorization IDs has the same meaning throughout your entire network. In practice, that might not be so, and the duplication of IDs on different LUs is a security exposure.

Example: Suppose that the ID DBADM1 is known to the local DB2 and has DBADM authority over certain databases there; suppose also that the same ID exists in some remote LU. If an attachment request comes in from DBADM1, and if nothing is done to alter the ID, the wrong user can exercise privileges of DBADM1 in the local DB2. The way to protect against that exposure is to translate the remote ID into a different ID before the attachment request is accepted.

You must be prepared to translate the IDs of plan owners, package owners, and the primary IDs of processes that make remote requests. Do not plan to translate all IDs in the connection exit routine—the routine does not receive plan and package owner IDs.

If you have decided to manage inbound IDs through DB2, you can translate an inbound ID to some other value. Within DB2, you grant privileges and authorities only to the translated value. The “translation” is not affected by anything you do in your connection or sign-on exit routine. The *output* of the translation becomes the *input* to your sign-on exit routine.

Recommendation: Do not translate inbound IDs in an exit routine; translate them only through the SYSIBM.USERNAMES table.

The examples in the following table shows the possibilities for translation and how to control translation by SYSIBM.USERNAMES. You can use entries to allow requests only from particular LUs or particular IDs, or from combinations of an ID and an LU. You can also translate any incoming ID to another value.

Table 70. Your SYSIBM.USERNAMES table. (Row numbers are added for reference.)

Row	TYPE	AUTHID	LINKNAME	NEWAUTHID
1	I	blank	LUSNFRAN	blank
2	I	BETTY	LUSNFRAN	ELIZA
3	I	CHARLES	blank	CHUCK
4	I	ALBERT	LUDALLAS	blank
5	I	BETTY	blank	blank

The following table shows the search order of the SYSIBM.USERNAMES table.

Table 71. Precedence search order for SYSIBM.USERNAMES table

AUTHID	LINKNAME	Result
Name	Name	If NEWAUTHID is specified, AUTHID is translated to NEWAUTHID for the specified LINKNAME.
Name	Blank	If NEWAUTHID is specified, AUTHID is translated to NEWAUTHID for all LINKNAMEs.
Blank	Name	If NEWAUTHID is specified, it is substituted for AUTHID for the specified LINKNAME.
Blank	Blank	Unavailable resource message (SQLCODE -904) is returned.

DB2 searches SYSIBM.USERNAMES to determine how to translate for each of the requests that are listed in the following table.

Table 72. How DB2 translates inbound authorization ids

Request	How DB2 translates request
ALBERT requests from LUDALLAS	DB2 searches for an entry for AUTHID=ALBERT and LINKNAME=LUDALLAS. DB2 finds one in row 4, so the request is accepted. The value of NEWAUTHID in that row is blank, so ALBERT is left unchanged.

Table 72. How DB2 translates inbound authorization ids (continued)

Request	How DB2 translates request
BETTY requests from LUDALLAS	DB2 searches for an entry for AUTHID=BETTY and LINKNAME=LUDALLAS; none exists. DB2 then searches for AUTHID=BETTY and LINKNAME=blank. It finds that entry in row 5, so the request is accepted. The value of NEWAUTHID in that row is blank, so BETTY is left unchanged.
CHARLES requests from LUDALLAS	DB2 searches for AUTHID=CHARLES and LINKNAME=LUDALLAS; no such entry exists. DB2 then searches for AUTHID=CHARLES and LINKNAME=blank. The search ends at row 3; the request is accepted. The value of NEWAUTHID in that row is CHUCK, so CHARLES is translated to CHUCK.
ALBERT requests from LUSNFRAN	DB2 searches for AUTHID=ALBERT and LINKNAME=LUSNFRAN; no such entry exists. DB2 then searches for AUTHID=ALBERT and LINKNAME=blank; again no entry exists. Finally, DB2 searches for AUTHID=blank and LINKNAME=LUSNFRAN, finds that entry in row 1, and the request is accepted. The value of NEWAUTHID in that row is blank, so ALBERT is left unchanged.
BETTY requests from LUSNFRAN	DB2 finds row 2, and BETTY is translated to ELIZA.
CHARLES requests from LUSNFRAN	DB2 finds row 3 before row 1; CHARLES is translated to CHUCK.
WILBUR requests from LUSNFRAN	No provision is made for WILBUR, but row 1 of the SYSIBM.USERNAMES table allows any ID to make a request from LUSNFRAN and to pass without translation. The acceptance level for LUSNFRAN is "already verified", so WILBUR can pass without a password check by RACF. After accessing DB2, WILBUR can use only the privileges that are granted to WILBUR and to PUBLIC (for DRDA access) or to PUBLIC AT ALL LOCATIONS (for DB2 private-protocol access).
WILBUR requests from LUDALLAS	Because the acceptance level for LUDALLAS is "verify" as recorded in the SYSIBM.LUNAMES table, WILBUR must be known to the local RACF. DB2 searches in succession for one of the combinations WILBUR/LUDALLAS, WILBUR/blank, or blank/LUDALLAS. None of those is in the table, so the request is rejected. The absence of a row permitting WILBUR to request from LUDALLAS imposes a "come-from" check: WILBUR can attach from some locations (LUSNFRAN), and some IDs (ALBERT, BETTY, and CHARLES) can attach from LUDALLAS, but WILBUR cannot attach if coming from LUDALLAS.

In the process of accepting remote attachment requests, any step that calls RACF is likely to have a relatively high performance cost. To trade some of that cost for a somewhat greater security exposure, have RACF check the identity of the other LU just once. Then trust the partner LU, translating the inbound IDs and not requiring or using passwords. In this case, no calls are made to RACF from within DB2; the penalty is only that you make the partner LU responsible for verifying IDs.

If you update tables in the CDB while the distributed data facility is running, the changes might not take effect immediately. If incoming authorization IDs are managed through DB2 and if the ICSF is installed and properly configured, you can use the DSNLEUSR stored procedure to encrypt translated authorization IDs

and store them in the NEWAUTHID column of the SYSIBM.USERNAMES table. DB2 decrypts the translated authorization IDs during connection processing.

Associating inbound IDs with secondary IDs

Your decisions on password encryption, ID translation, and so on determine the value you use for the primary authorization ID on an attachment request.

They also determine whether those requests are next treated as connection requests or as sign-on requests. That means that the remote request next goes through the same processing as a local request, and that you have the opportunity to associate the primary ID with a list of secondary IDs in the same way you do for local requests.

Managing inbound TCP/IP-based connection requests

DRDA connections that use TCP/IP have fewer security controls than do connections that use SNA protocols. When planning to control inbound TCP/IP connection requests, you must first decide whether you want the requests to have authentication information, such as RACF passwords, RACF PassTickets, and Kerberos tickets, passed along with the authorization ID.

If you require authentication, specify NO on the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5, which is the default option, to indicate that you require this authentication information. Also, ensure that the security subsystem at your server is properly configured to handle the authentication information that is passed to it. If you do not specify NO, all incoming TCP/IP requests can connect to DB2 without any authentication.

For requests that use RACF passwords or PassTickets, enter the following RACF command to indicate which user IDs that use TCP/IP are authorized to access DDF (the distributed data facility address space):

```
PERMIT ssnm.DIST CLASS(DSNR) ID(yyy) ACCESS(READ)
  WHEN(APPCPORT(TCPIP))
```

Consider the following questions:

Do you permit access by TCP/IP? If the serving DB2 for z/OS subsystem has a DRDA port and resynchronization port specified in the BSDS, DB2 is enabled for TCP/IP connections.

Do you manage inbound IDs through DB2 or RACF? All IDs must be passed to RACF or Kerberos for processing. No option exists to handle incoming IDs through DB2.

Do you trust the partner? TCP/IP does not verify partner LUs as SNA does. If your requesters support mutual authentication, use Kerberos to handle this on the requester side.

If you use passwords, are they encrypted? Passwords can be encrypted through:

- RACF using PassTickets
- DRDA password encryption support. DB2 for z/OS as a server supports DRDA encrypted passwords and encrypted user IDs with encrypted passwords.

If you use Kerberos, are users authenticated? If your distributed environment uses Kerberos to manage users and perform user authentication, DB2 for z/OS can use Kerberos security services to authenticate remote users.

Do you translate inbound IDs? Inbound IDs are not translated when you use TCP/IP.

How do you associate inbound IDs with secondary IDs? To associate an inbound ID with secondary IDs, modify the default connection exit routine (DSN3@ATH). TCP/IP requests do not use the sign-on exit routine.

Processing of TCP/IP-based connection requests

The DB2 server completes the following sequence of authentication process when handling a remote connection request that uses the TCP/IP protocol.

1. As the following diagram shows, DB2 checks to see if an authentication token (RACF encrypted password, RACF PassTicket, DRDA encrypted password, or Kerberos ticket) accompanies the remote request.

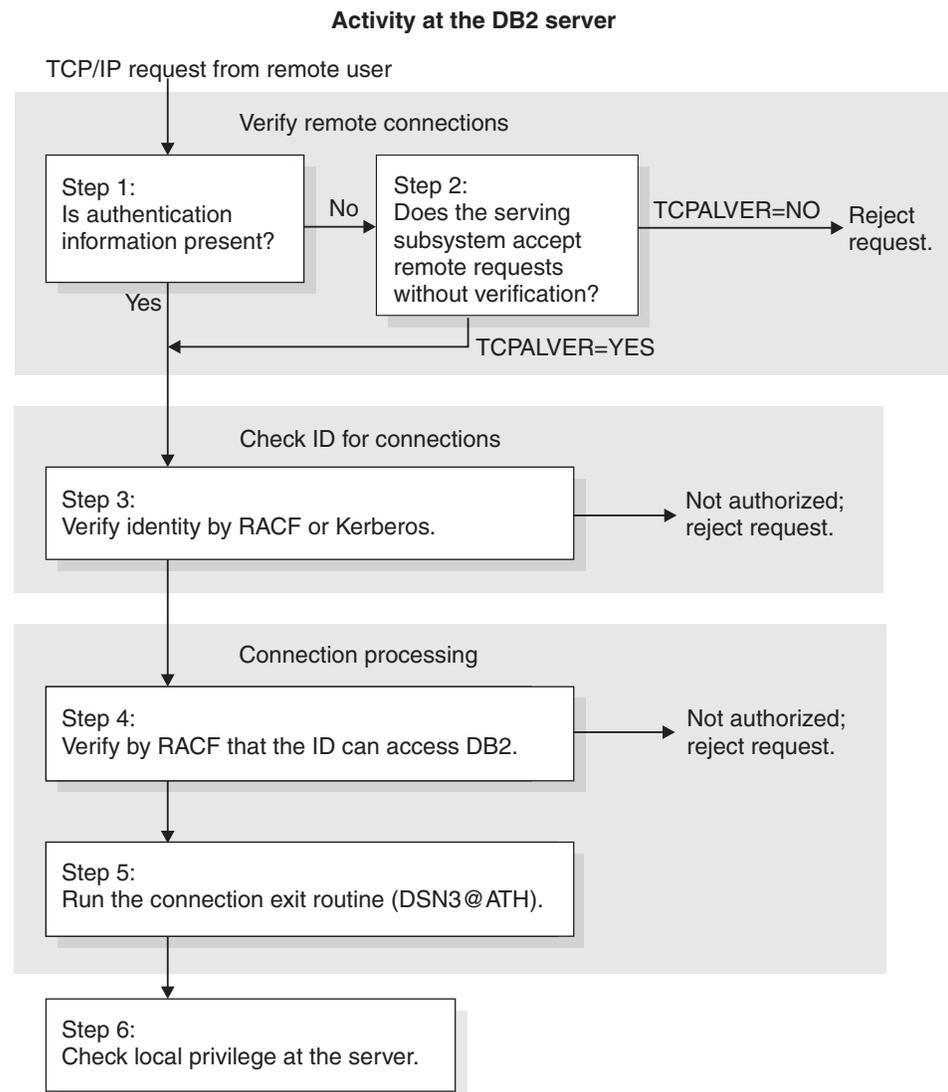


Figure 24. DB2 processing of TCP/IP-based connection requests

2. If no authentication token is supplied, DB2 checks the TCPALVER subsystem parameter to see if DB2 accepts IDs without authentication information. If

TCPALVER=NO, authentication information must accompany all requests, and DB2 rejects the request. If TCPALVER=YES, DB2 accepts the request without authentication.

3. The identity is a RACF ID that is authenticated by RACF if a password or PassTicket is provided, or the identity is a Kerberos principal that is validated by Kerberos Security Server, if a Kerberos ticket is provided. Ensure that the ID is defined to RACF in all cases. When Kerberos tickets are used, the RACF ID is derived from the Kerberos principal identity. To use Kerberos tickets, ensure that you map Kerberos principal names with RACF IDs.

In addition, depending on your RACF environment, the following RACF checks may also be performed:

- If the RACF APPL class is active, RACF verifies that the ID has access to the DB2 APPL. The APPL resource that is checked is the LU name that the requester used when the attachment request was issued. This is either the local DB2 LU name or the generic LU name.
- If the RACF APPCPORT class is active, RACF verifies that the ID is authorized to access z/OS from the port of entry (POE). The POE that RACF uses in the RACROUTE VERIFY call depends on whether all the following conditions are true:
 - The current operating system is z/OS V1.5 or later
 - The TCP/IP Network Access Control is configured
 - The RACF SERVAUTH class is active

If all these conditions are true, RACF uses the remote client's POE security zone name that is defined in the TCP/IP Network Access Control file. If one or more of these conditions is not true, RACF uses the literal string 'TCPIP'.

If this is a request to change a password, the password is changed.

4. The remote request is now treated like a local connection request (using the DIST environment for the DSNR resource class). DB2 calls RACF to check the ID's authorization against the *ssnm.DIST* resource.
5. DB2 invokes the connection exit routine. The parameter list that is passed to the routine describes where the remote request originated.
6. The remote request has a primary authorization ID, possibly one or more secondary IDs, and an SQL ID. (The SQL ID cannot be translated.) The plan or package owner ID also accompanies the request. Privileges and authorities that are granted to those IDs at the DB2 server govern the actions that the request can take.

Managing denial-of-service attacks

With DB2, you can manage denial-of-service attacks in the network connections to a DB2 server.

The most common type of denial-of-service attack occurs when an attacker "floods" a network with connection requests to a DB2 server. If this occurs, the attacker quickly exhausts the threshold for the maximum number of remote connections that are defined to the DB2 server system. As a result, no additional remote connections can be accepted by the DB2 server, including those from legitimate client systems.

To prevent the typical denial-of-service attacks, DB2 monitors the traffic of inbound connections and terminates those that don't contain data for establishing a valid connection.

Managing outbound connection requests

If you are planning to send requests to another DB2 subsystem, consider that the security administrator of that subsystem might have chosen any of the options. You need to know what those choices are and make entries in your CDB to correspond to them. You can also choose some things independently of what the other subsystem requires.

If you are planning to send remote requests to a DBMS that is not DB2 for z/OS, you need to satisfy the requirements of that system.

DB2 chooses how to send authentication tokens based on the network protocols that are used (SNA or TCP/IP). If the request is sent using SNA, the authentication tokens are sent in the SNA attachment request (FMH5), unless you are using Kerberos. If you use Kerberos, authentication tokens are sent with DRDA security commands. If the request uses TCP/IP, the authentication tokens are always sent using DRDA security commands.

At least one authorization ID is always sent to the server to be used for authentication. That ID is one of the following values:

- The primary authorization ID of the process.
- If you connect to the server using a CONNECT statement with the USER keyword, the ID that you specify as the USER ID. The CONNECT statement allows non-RACF user IDs on the USER keyword. If connecting to a remote location, the user ID is not authenticated by RACF.

However, other IDs can accompany some requests. You need to understand what other IDs are sent because they are subject to translation. You must include these other IDs in table SYSIBM.USERNAMES to avoid an error when you use outbound translation. The following table shows the IDs that you send in the different situations:

Table 73. IDs that accompany the primary ID on a remote request

In this situation:	You send this ID also:
An SQL query, using DB2 private-protocol or DRDA-protocol access	The plan owner
A remote BIND, COPY, or REBIND PACKAGE command	The package owner

If the connection is to a remote non-DB2 for z/OS server using DRDA protocol and if the outbound translation is specified, a row for the plan owner in the USERNAMES table is optional.

Communications database for the requester

The *communications database* (CDB) is a set of DB2 catalog tables that let you control aspects of remote requests. Columns in the SYSIBM.LUNAMES, SYSIBM.IPNAMES, SYSIBM.USERNAMES, and SYSIBM.LOCATIONS tables pertain to security issues related to the requesting system.

SYSIBM.LUNAMES columns:

The SYSIBM.LUNAMES table is used only for outbound requests that use SNA protocols.

LUNAME CHAR(8)

The LUNAME of the remote system. A blank value identifies a default row that serves requests from any system that is not specifically listed elsewhere in the column.

SECURITY_OUT (CHAR 1)

Indicates the security option that is used when local DB2 SQL applications connect to any remote server that is associated with the corresponding LUNAME.

A The letter A signifies the security option of already verified, and it is the default. With A, outbound connection requests contain an authorization ID and no authentication token. The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column.

R The letter R signifies the RACF PassTicket security option. Outbound connection requests contain a user ID and a RACF PassTicket. The LUNAME column is used as the RACF PassTicket application name.

The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column. The translated ID is used to build the RACF PassTicket. Do not specify R for CONNECT statements with a USER parameter.

P The letter P signifies the password security option. Outbound connection requests contain an authorization ID and a password. The password is obtained from RACF if ENCRYPTPSWDS=Y, or from SYSIBM.USERNAMES if ENCRYPTPSWDS=N. If you get the password from SYSIBM.USERNAMES, the USERNAMES column of SYSIBM.LUNAMES must contain B or O. The value that is used for an outbound request is the translated ID.

ENCRYPTPSWDS CHAR(1)

Indicates whether passwords received from and sent to the corresponding LUNAME are encrypted. This column only applies to DB2 for z/OS and DB2 for z/OS partners when passwords are used as authentication tokens.

Y Yes, passwords are encrypted. For outbound requests, the encrypted password is extracted from RACF and sent to the server. For inbound requests, the password is treated as encrypted.

N No, passwords are not encrypted. This is the default; any character but Y is treated as N.

Recommendation: When you connect to a DB2 for z/OS partner that is at Version 5 or a subsequent release, use RACF PassTickets (SECURITY_OUT='R') instead of encrypting passwords.

USERNAMES CHAR(1)

Indicates whether an ID accompanying a remote attachment request, which is received from or sent to the corresponding LUNAME, is subject to translation and "come from" checking. When you specify I, O, or B, use the SYSIBM.USERNAMES table to perform the translation.

I An inbound ID is subject to translation.

- O An outbound ID, sent to the corresponding LUNAME, is subject to translation.
- B Both inbound and outbound IDs are subject to translation.
- blank No IDs are translated.

◀ GUPI

SYSIBM.IPNAMES columns:

The SYSIBM.IPNAMES table is used only for outbound requests that use TCP/IP protocols.

▶ GUPI

LINKNAME CHAR(8)

The name used in the LINKNAME column of SYSIBM.LOCATIONS to identify the remote system.

IPADDR

Specifies an IP address or domain name of a remote TCP/IP host.

SECURITY_OUT

Indicates the DRDA security option that is used when local DB2 SQL applications connect to any remote server that is associated with this TCP/IP host.

- A The letter A signifies the security option of already verified, and it is the default. Outbound connection requests contain an authorization ID and no password. The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column.

The authorization ID is not encrypted when it is sent to the partner. For encryption, see option D.

- R The letter R signifies the RACF PassTicket security option. Outbound connection requests contain a user ID and a RACF PassTicket. When a RACF PassTicket is generated, the LINKNAME column value is used as the RACF PassTicket application name and must match the following at the target server

- LUNAME - if the remote site is a DB2 subsystem that is defined with only an LUNAME value and no GENERIC LU name value or IPNAME value
- GENERIC - if the remote site is a DB2 subsystem that is defined with a GENERIC LU name value in addition to an LUNAME value but no IPNAME value
- IPNAME - if the remote site is a DB2 subsystem that is defined with an IPNAME value that triggers the remote DB2 subsystem's DDF to activate only its TCP/IP communications support.

The value that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the value in the USERNAMES column. The translated ID is used to build the RACF PassTicket. Do not specify R for CONNECT statements with a USER parameter.

The authorization ID is not encrypted when it is sent to the partner.

- D** The letter D signifies the security option of user ID and security-sensitive data encryption. Outbound connection requests contain an authorization ID and no password. The authorization ID that is used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending on the USERNAMES column.

This option indicates that the user ID and the security-sensitive data are to be encrypted. If you do not require encryption, see option A.

- E** The letter E signifies the security option of user ID, password, and security-sensitive data encryption. Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table. The USERNAMES column must specify "O".

This option indicates that the user ID, password, and security-sensitive data are to be encrypted. If you do not require security-sensitive data encryption, see option P.

- P** The letter P signifies the password security option. Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table. If you specify P, the USERNAMES column must specify "O".

If you specify P and the server supports encryption, the user ID and the password are encrypted. If the server does not support encryption, the user ID and the password are sent to the partner in clear text. If you also need to encrypt security-sensitive data, see option E.

USERNAMES CHAR(1)

This column indicates whether an outbound request translates the authorization ID. When you specify O, use the SYSIBM.USERNAMES table to perform the translation.

- O** The letter O signifies an outbound ID that is subject to translation. Rows in the SYSIBM.USERNAMES table are used to perform ID translation. If a connection to any remote server is to be established as trusted, a row in the SYSIBM.USERNAMES table is used to obtain the system authorization ID.

- S** The letter S signifies the system authorization ID, within a trusted context, obtained from the SYSIBM.USERNAMES table. If the system authorization ID that is specified in the AUTHID column is different from the primary authorization ID, DB2 sends the user switch request on behalf of the primary authorization ID after successfully establishing the trusted connection.

blank No translation is done.



SYSIBM.USERNAMES columns:

The SYSIBM.USERNAMES table is used by outbound connection requests that use SNA and TCP/IP protocols.

GUPI**TYPE CHAR(1)**

Indicates whether the row is used for inbound or outbound translation:

- S** The row is used to obtain the outbound system authorization ID for establishing a trusted connection.
- I** The row applies to inbound IDs.
- O** The row applies to outbound IDs.

The field should contain only I, O, or S. Any other character, including blank, causes the row to be ignored.

AUTHID VARCHAR(128)

An authorization ID that is permitted and perhaps translated. If blank, any authorization ID is permitted with the corresponding LINKNAME, and all authorization IDs are translated in the same way.

LINKNAME CHAR(8)

Identifies the VTAM or TCP/IP network locations that are associated with this row. A blank value in this column indicates that this name translation rule applies to any TCP/IP or SNA partner.

If you specify a nonblank value for this column, one or both of the following situations must be true:

- A row exists in table SYSIBM.LUNAMES that has an LUNAME value that matches the LINKNAME value that appears in this column.
- A row exists in table SYSIBM.IPNAMES that has a LINKNAME value that matches the LINKNAME value that appears in this column.

NEWAUTHID VARCHAR(128)

The translated authorization ID. If blank, no translation occurs.

PASSWORD CHAR(8)

A password that is sent with outbound requests. This password is not provided by RACF and cannot be encrypted.

GUPI**SYSIBM.LOCATIONS columns:**

The SYSIBM.LOCATIONS table contains a row for every accessible remote server. Each row associates a LOCATION name with the TCP/IP or SNA network attributes for the remote server. Requesters are not defined in this table.

GUPI**LOCATION CHAR(16)**

Indicates the unique location name by which the the remote server is known to local DB2 SQL applications.

LINKNAME CHAR(8)

Identifies the VTAM or TCP/IP network locations that are associated with this row. A blank value in this column indicates that this name translation rule applies to any TCP/IP or SNA partner.

If you specify a nonblank value for this column, one or both of the following situations must be true:

- A row exists in table SYSIBM.LUNAMES that has a LUNAME value that matches the LINKNAME value that appears in this column.
- A row exists in table SYSIBM.IPNAMES that has a LINKNAME value that matches the LINKNAME value that appears in this column.

PORT CHAR(32)

Indicates that TCP/IP is used for outbound DRDA connections when the following statement is true:

- A row exists in SYSIBM.IPNAMES, where the LINKNAME column matches the value that is specified in the SYSIBM.LOCATIONS LINKNAME column.

If the previously mentioned row is found, and the SECURE column has a value of 'N', the value of the PORT column is interpreted as follows:

- If PORT is blank, the default DRDA port (446) is used.
- If PORT is nonblank, the value that is specified for PORT can take one of two forms:
 - If the value in PORT is left-justified with one to five numeric characters, the value is assumed to be the TCP/IP port number of the remote database server.
 - Any other value is assumed to be a TCP/IP service name, which you can convert to a TCP/IP port number by using the TCP/IP getservbyname socket call. TCP/IP service names are not case-sensitive.

If the previously mentioned row is found, and the SECURE column has a value of 'Y', the value of the PORT column is interpreted as follows:

- If PORT is blank, the default secure DRDA port (448) is used.
- If PORT is nonblank, the value that is specified for PORT takes the value of the configured secure DRDA port at the remote server.

TPN VARCHAR(64)

Used only when the local DB2 begins an SNA conversation with another server. When used, TPN indicates the SNA LU 6.2 transaction program name (TPN) that will allocate the conversation. A length of zero for the column indicates the default TPN. For DRDA conversations, this is the DRDA default, which is X'07F6C4C2'.

For DB2 private protocol conversations, this column is not used. For an SQL/DS™ server, TPN should contain the resource ID of the SQL/DS machine.

DBALIAS(128)

Used to access a remote database server. If DBALIAS is blank, the location name is used to access the remote database server. This column does not change the name of any database objects sent to the remote site that contains the location qualifier.

TRUSTED

Indicates whether the connection to the remote server can be trusted. This is restricted to TCP/IP only. This column is ignored for connections that use SNA.

- Y** The location is trusted. Access to the remote location requires a trusted context that is defined at the remote location.
- N** The location is not trusted.

SECURE

Indicates the use of the Secure Socket Layer (SSL) protocol for outbound DRDA connections when local DB2 applications connect to the remote database server by using TCP/IP.

Y A secure SSL connection is required for the outbound DRDA connection.

N A secure connection is not required for the outbound DRDA connection.

GUPI

Processing of outbound connection requests

The DB2 subsystem completes the following steps when sending out a connection request:

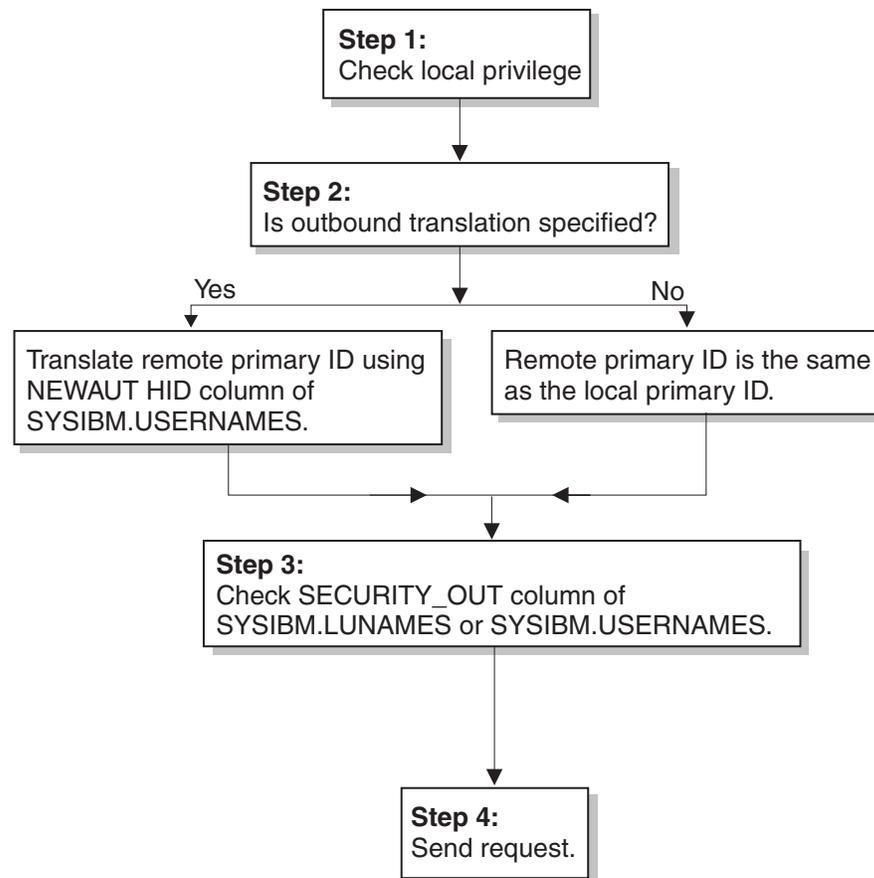


Figure 25. Steps in sending a request from a DB2 subsystem

1. The DB2 subsystem that sends the request checks whether the primary authorization ID has the privilege to execute the plan or package.
DB2 determines which value in the LINKNAME column of the SYSIBM.LOCATIONS table matches either the LUNAME column in the SYSIBM.LUNAMES table or the LINKNAME column in the SYSIBM.IPNAMES table. This check determines whether SNA or TCP/IP protocols are used to carry the DRDA request. (Statements that use DB2 private protocol, not DRDA, always use SNA.)

2. When a plan is executed, the authorization ID of the plan owner is sent with the primary authorization ID. When a package is bound, the authorization ID of the package owner is sent with the primary authorization ID. If the USERNAMES column of the SYSIBM.LUNAMES table contains O or B, or if the USERNAMES column of the SYSIBM.IPNAMES table contains O, both IDs are subject to translation under control of the SYSIBM.USERNAMES table. Ensure that these IDs are included in SYSIBM.USERNAMES, or SQLCODE -904 is issued. DB2 translates the ID as follows:

- If a nonblank value of NEWAUTHID is in the row, that value becomes the new ID.
- If NEWAUTHID is blank, the ID is not changed.

If the SYSIBM.USERNAMES table does not contain a new authorization ID to which the primary authorization ID is translated, the request is rejected with SQLCODE -904.

If the USERNAMES column does not contain O or B, the IDs are not translated.

3. SECURITY_OUT is checked for outbound security options as shown in the following diagram.

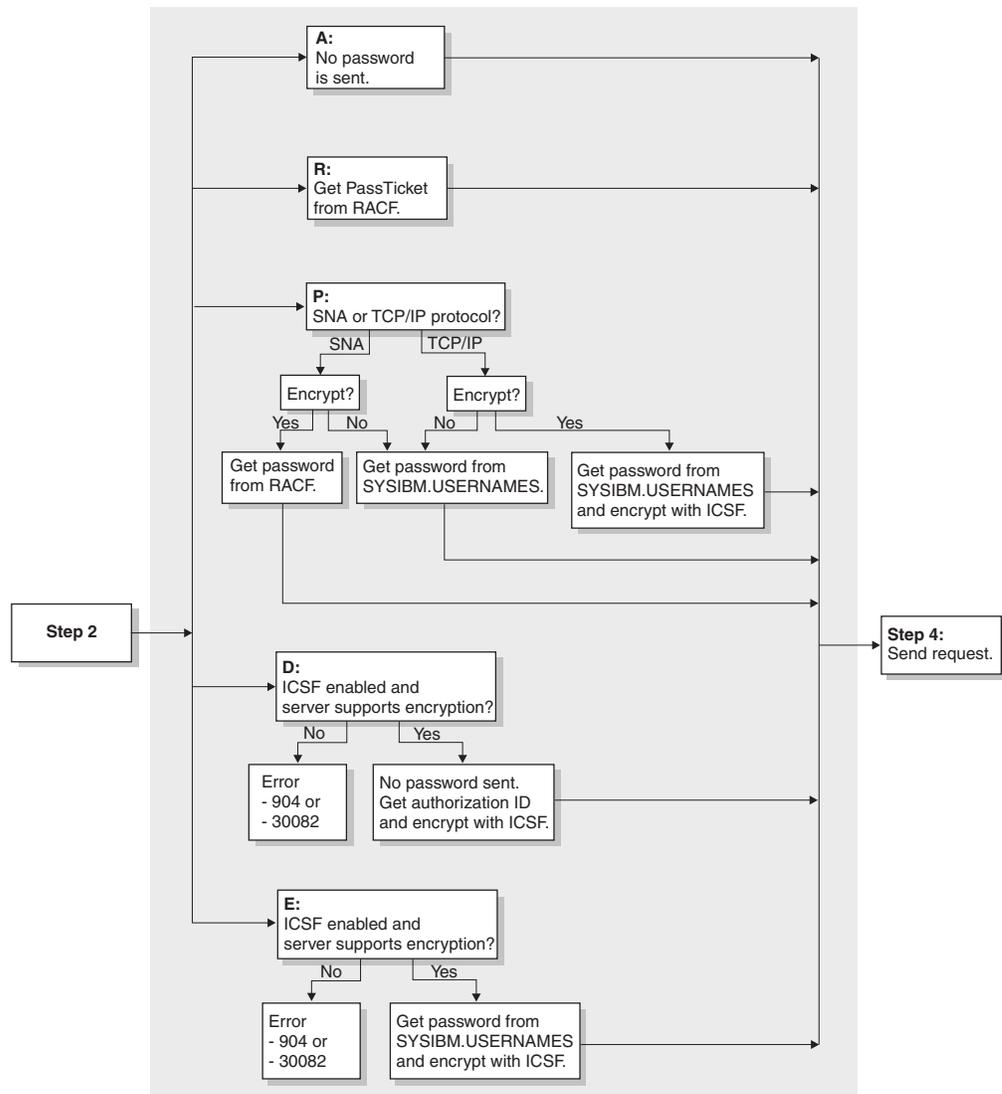


Figure 26. Details of Step 3

- A** Already verified. No password is sent with the authorization ID. This option is valid only if the server accepts already verified requests.
- For SNA, the server must have specified A in the SECURITY_IN column of SYSIBM.LUNAMES.
 - For TCP/IP, the server must have specified YES in the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5.
- R** RACF PassTicket. If the primary authorization ID was translated, that translated ID is sent with the PassTicket.
- P** Password. The outbound request must be accompanied by a password:
- If the requester is DB2 for z/OS and uses SNA protocols, passwords can be encrypted if you specify Y in the ENCRYPTPSWDS column of SYSIBM.LUNAMES. If passwords are encrypted, the password is obtained from RACF. If passwords are not encrypted, the password is obtained from the PASSWORD column of SYSIBM.USERNAMES.
 - If the requester uses TCP/IP protocols, the password is obtained from the PASSWORD column of SYSIBM.USERNAMES. If the

Integrated Cryptographic Service Facility is enabled and properly configured and the server supports encryption, the password is encrypted.

Recommendation: Use RACF PassTickets to avoid sending unencrypted passwords over the network.

- D User ID and security-sensitive data encryption. No password is sent with the authorization ID. If the Integrated Cryptographic Service Facility (ICSF) is enabled and properly configured and the server supports encryption, the authorization ID is encrypted before it is sent. If the ICSF is not enabled or properly configured, SQL return code -904 is returned. If the server does not support encryption, SQL return code -30082 is returned.
- E User ID, password, and security-sensitive data encryption. If the ICSF is enabled and properly configured and the server supports encryption, the password is encrypted before it is sent. If the ICSF is not enabled or properly configured, SQL return code -904 is returned. If the server does not support encryption, SQL return code -30082 is returned.

4. Send the request.

Translating outbound IDs

If an ID on your system is duplicated on a remote system, you can translate outbound IDs to avoid confusion. You can also translate IDs to ensure that they are accepted by the remote system.

To indicate that you want to translate outbound user IDs, perform the following steps:

1. Specify an O in the USERNAMES column of table SYSIBM.IPNAMES or SYSIBM.LUNAMES.
2. Use the NEWAUTHID column of SYSIBM.USERNAMES to specify the ID to which the outbound ID is translated.

Example 1: Suppose that the remote system accepts from you only the IDs XXGALE, GROUP1, and HOMER.

1. Specify that outbound translation is in effect for the remote system LUXXX by specifying in SYSIBM.LUNAMES the values that are shown in the following table.

Table 74. SYSIBM.LUNAMES to specify that outbound translation is in effect for the remote system LUXXX

LUNAME	USERNAMES
LUXXX	O

If your row for LUXXX already has I for the USERNAMES column (because you translate inbound IDs that come from LUXXX), change I to B for both inbound and outbound translation.

2. Translate the ID GALE to XXGALE on all outbound requests to LUXXX by specifying in SYSIBM.USERNAMES the values that are shown in the following table.

Table 75. Values in SYSIBM.USERNAMES to translate GALE to XXGALE on outbound requests to LUXXX

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	GALE	LUXXX	XXGALE	GALEPASS

- Translate EVAN and FRED to GROUP1 on all outbound requests to LUXXX by specifying in SYSIBM.USERNAMES the values that are shown in the following table.

Table 76. Values in SYSIBM.USERNAMES to translate EVAN and FRED to GROUP1 on outbound requests to LUXXX

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	EVAN	LUXXX	GROUP1	GRP1PASS
O	FRED	LUXXX	GROUP1	GRP1PASS

- Do not translate the ID HOMER on outbound requests to LUXXX. (HOMER is assumed to be an ID on your DB2, and on LUXXX.) Specify in SYSIBM.USERNAMES the values that are shown in the following table.

Table 77. Values in SYSIBM.USERNAMES to not translate HOMER on outbound requests to LUXXX

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	HOMER	LUXXX	(blank)	HOMERSPW

- Reject any requests from BASIL to LUXXX before they are sent. To do that, leave SYSIBM.USERNAMES empty. If no row indicates what to do with the ID BASIL on an outbound request to LUXXX, the request is rejected.

Example 2: If you send requests to another LU, such as LUYYY, you generally need another set of rows to indicate how your IDs are to be translated on outbound requests to LUYYY.

However, you can use a single row to specify a translation that is to be in effect on requests to all other LUs. For example, if HOMER is to be sent untranslated everywhere, and DOROTHY is to be translated to GROUP1 everywhere, specify in SYSIBM.USERNAMES the values that are shown in the following table.

Table 78. Values in SYSIBM.USERNAMES to not translate HOMER and to translate DOROTHY to GROUP1

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	HOMER	(blank)	(blank)	HOMERSPW
O	DOROTHY	(blank)	GROUP1	GRP1PASS

You can also use a single row to specify that all IDs that accompany requests to a single remote system must be translated. For example, if every one of your IDs is to be translated to THEIRS on requests to LUYYY, specify in SYSIBM.USERNAMES the values that are shown in the following table.

Table 79. Values in SYSIBM.USERNAMES to translate every ID to THEIRS

TYPE	AUTHID	LINKNAME	NEWAUTHID	PASSWORD
O	(blank)	LUYYY	THEIR	THEPASS

If the ICSF is installed and properly configured, you can use the DSNLEUSR stored procedure to encrypt the translated outbound IDs that are specified in the NEWAUTHID column of SYSIBM.USERNAMES. DB2 decrypts the translated outbound IDs during connection processing.

Sending passwords

For the tightest security, do not send passwords through the network. Instead, use one of the following security mechanisms:

- RACF encrypted passwords
- RACF PassTickets
- Kerberos tickets
- DRDA-encrypted passwords or DRDA-encrypted user IDs with encrypted passwords

If you have to send passwords through the network, you can put the password for an ID in the PASSWORD column of the SYSIBM.USERNAMES table.

Recommendation: Use the DSNLEUSR stored procedure to encrypt passwords in SYSIBM.USERNAMES. If the ICSF is installed and properly configured, you can use the DSNLEUSR stored procedure to encrypt passwords in the SYSIBM.USERNAMES table. DB2 decrypts the password during connection processing.

DB2 for z/OS allows the use of RACF encrypted passwords or RACF PassTickets. However, workstations, such as Windows® workstations, do not support these security mechanisms. RACF encrypted passwords are not a secure mechanism because they can be replayed.

Recommendation: Do not use RACF encrypted passwords unless you are connecting to a previous release of DB2 for z/OS.

Sending RACF-encrypted passwords

For DB2 subsystems that use the SNA protocols to communicate with each other, you can specify password encryption in the SYSIBM.LUNAMES table.

Table 80. Specifying password encryption in SYSIBM.LUNAMES

LUNAME	USERNAMES	ENCRYPTPSWDS
LUXXX	O	Y

The partner DB2 must also specify password encryption in its SYSIBM.LUNAMES table. Both partners must register each ID and its password with RACF. Then, for every request to LUXXX, your DB2 calls RACF to supply an encrypted password to accompany the ID. With password encryption, you do not use the PASSWORD column of SYSIBM.USERNAMES, so the security of that table becomes less critical.

Sending RACF PassTickets

To send RACF PassTickets with your remote requests to a particular remote system, you can specify 'R' in the SECURITY_OUT column of the SYSIBM.IPNAMES or SYSIBM.LUNAMES table for that system.

Perform the following steps to set up RACF to generate PassTickets:

1. Activate the RACF PTKTDATA class by issuing the following RACF commands:

```
SETROPTS CLASSACT(PTKTDATA)
SETROPTS RACLIST(PTKTDATA)
```

2. Define a RACF profiles for each remote system by entering the system name as it appears in the LINKNAME column in the SYSIBM.LOCATIONS table.

For example, issue the following command defines a profile for a remote system, DB2A, in the RACF PTKTDATA class:

```
RDEFINE PTKTDATA DB2A SSIGNON(KEYMASKED(E001193519561977))
```

3. Refresh the RACF PTKTDATA definition with the new profiles by issuing the following command:

```
SETROPTS RACLIST(PTKTDATA) REFRESH
```

Sending encrypted passwords from workstations

Depending on the DRDA level, clients can encrypt passwords, user IDs and passwords, or user IDs, passwords, and security-sensitive data when they send them to a DB2 for z/OS server.

This support uses the Diffie-Hellman key distribution algorithm.⁴ To enable DB2 Connect to flow encrypted passwords, database connection services (DCS) authentication must be set to DCS_ENCRYPT in the DCS directory entry.

When the workstation application issues an SQL CONNECT, the workstation negotiates this support with the database server. If supported, a shared private key is generated by the client and server using the Diffie-Hellman public key technology and the password is encrypted using 56-bit DES with the shared private key. The encrypted password is non-replayable, and the shared private key is generated on every connection. If the server does not support password encryption, the application receives SQLCODE -30073 (DRDA security manager level 6 is not supported).

4. Diffie-Hellman is one of the first standard public key algorithms. It results in exchanging a connection key which is used by client and server to generate a shared private key. The 56-bit Data Encryption Standards (DES) algorithm is used for encrypting and decrypting of the password using the shared private key.

Chapter 7. Managing access through trusted contexts

DB2 helps you satisfy the need for data security and accountability by enabling you to create and use trusted contexts as another method to manage access to your DB2 servers.

You can use trusted connections within a trusted context to reuse the authorization and switch users of the connection without the database server needing to authenticate the IDs.

Trusted contexts

A *trusted context* is an independent database entity that you can define based on a system authorization ID and connection trust attributes.

The trust attributes specify a set of characteristics about a specific connection. These attributes include the IP address, domain name, or SERVAUTH security zone name of a remote client and the job or task name of a local client.

A trusted context allows for the definition of a unique set of interactions between DB2 and the external entity, including the following abilities:

- The ability for the external entity to use an established database connection with a different user without the need to authenticate that user at the DB2 server. This ability eliminates the need to manage end-user passwords by the external entity. Also, a database administrator can assume the identity of other users and perform actions on their behalf.
- The ability for a DB2 authorization ID to acquire one or more privileges within a trusted context that are not available to it outside of that trusted context. This is accomplished by associating a role with the trusted context.

The following client applications provide support for the trusted context:

- The DB2 Universal Java Driver introduces new APIs for establishing trusted connections and switching users of a trusted connection.
- The DB2 CLI/ODBC Driver introduces new keywords for connecting APIs to establish trusted connections and switch users of a trusted connection.
- The Websphere Application Server 6.0 exploits the trusted context support through its "propagate client identity" property.

Trusted connections

A *trusted connection* is a database connection that is established when the connection attributes match the attributes of a unique trusted context that is defined at the server. It can be established locally or at a remote location.

A trusted context allows you to establish a trusted relationship between DB2 and an external entity, such as a middleware server. DB2 evaluates a series of trust attributes to determine if a specific context is to be trusted. Currently, the only attribute that DB2 considers is the database connection. The relationship between a connection and a trusted context is established when the connection to the server is first created, and that relationship remains as long as that connection exists.

Defining trusted contexts

Before you can create a trusted connection, you must define a trusted context by using a system authorization ID and connection trust attributes.

A system authorization ID is the DB2 primary authorization ID that is used to establish the trusted connection. For local connections, the system authorization ID is derived as shown in the following table.

Table 81. System authorization ID for local connections

Source	System authorization ID
Started task (RRSAF)	USER parameter on JOB statement or RACF USER
TSO	TSO logon ID
BATCH	USER parameter on JOB statement

For remote connections, the system authorization ID is derived from the system user ID that is provided by an external entity, such as a middleware server.

The connection trust attributes identify a set of characteristics about the specific connection. The connection trust attributes are required for the connection to be considered a trusted connection. For a local connection, the connection trust attribute is the job or started task name. For a remote connection, the connection trust attribute is the client's IP address, domain name, or SERVAUTH security zone name. The connection trust attributes are as follows:

ADDRESS

Specifies the client's IP address or domain name, used by the connection to communicate with DB2. The protocol must be TCP/IP.

SERVAUTH

Specifies the name of a resource in the RACF SERVAUTH class. This resource is the network access security zone name that contains the IP address of the connection to communicate with DB2.

ENCRYPTION

Specifies the minimum level of encryption of the data stream (network encryption) for the connection. Supported values are as follows:

- NONE - No encryption. This is the default.
- LOW - DRDA data stream encryption.
- HIGH - Secure Socket Layer (SSL) encryption.

JOBNAME

Specifies the local z/OS started task or job name. The value of JOBNAME depends on the source of the address space, as shown in the following table.

Table 82. JOBNAME for local connections

Source	JOBNAME
Started task (RRSAF)	Job or started task name
TSO	TSO logon ID
BATCH	Job name on JOB statement

The JOBNAME attribute cannot be specified with the ADDRESS, SERVAUTH, or ENCRYPTION attributes.

Creating local trusted connections

You can establish a trusted connection to a local DB2 subsystem by using RRSAP or the DSN command processor under TSO and DB2I.

When you attempt to create a local trusted connection, DB2 searches for a trusted context that matches the primary authorization ID and the job or started task name that you supply. If DB2 finds a matching trusted context, DB2 checks if the DEFAULT SECURITY LABEL attribute is defined in the trusted context.

If the DEFAULT SECURITY LABEL attribute is defined with a security label, DB2 verifies the security label with RACF. This security label is used for multilevel security verification for the system authorization ID. If verification is successful, the connection is established as trusted. If the verification is not successful, the connection is established as a normal connection without any additional privileges.

In addition, the DB2 online utilities can run in a trusted connection if a matching trusted context is defined, if the primary authorization ID matches the SYSTEM AUTHID value of the trusted context, and if the job name matches the JOBNAME attribute defined for the trusted context.

Establishing remote trusted connections by DB2 for z/OS requesters

A DB2 for z/OS requester can establish a trusted connection to a remote location by setting up the new TRUSTED column in the SYSIBM.LOCATIONS table.

How DB2 obtains the system authorization ID to establish the trusted connection depends on the value of the SECURITY_OUT option in the SYSIBM.IPNAMES table. The SECURITY_OUT option in the SYSIBM.IPNAMES table must be 'E', 'P', or 'R'.

When the z/OS requester receives an SQL CONNECT with or without the USER and USING clauses to a remote location or if an application references a remote table or procedure, DB2 looks at the SYSIBM.LOCATIONS table to find a matching row. If DB2 finds a matching row, it checks the TRUSTED column. If the value of TRUSTED column is set to 'Y', DB2 looks at the SYSIBM.IPNAMES table. The values in the SECURITY_OUT column and USERNAMES column are used to determine the system authorization ID as follows:

SECURITY_OUT = 'P' or 'E' and USERNAMES = 'S'

The system authorization ID credentials that are used to establish the trusted connection are obtained from the row in the SYSIBM.USERNAMES table with TYPE 'S'.

DB2 sends the user switch request on behalf of the primary authorization ID without authentication under two conditions. First, the system authorization ID value in the AUTHID column is different from the primary authorization ID. Second, a trusted connection is successfully established.

SECURITY_OUT='P' or 'E' and USERNAMES = 'O'

If a row with TYPE 'S' is defined in the SYSIBM.USERNAMES table, the system authorization ID credentials that are used to establish the trusted connection are obtained from the row.

After successfully establishing the trusted connection, DB2 obtains the translated authorization ID information for the primary authorization ID

from the row in the SYSIBM.USERNAMES table with TYPE 'O'. DB2 sends the user switch request on behalf of the primary authorization ID with authentication.

If a row with TYPE 'S' is not defined in the SYSIBM.USERNAMES table, DB2 obtains the system authorization ID information that is used to establish the trusted connection from the row in the SYSIBM.USERNAMES table with TYPE 'O'.

SECURITY_OUT = 'R' and USERNAMES = ''

The primary authorization ID is used as the system authorization ID to establish the trusted connection.

SECURITY_OUT = 'R' and USERNAMES = 'S'

The system authorization ID that is used to establish the trusted connection is obtained from the row in the SYSIBM.USERNAMES table with TYPE='S'.

After establishing the trusted connection successfully, DB2 sends the user switch request on behalf of the primary authorization ID without authentication.

SECURITY_OUT = 'R' and USERNAMES = 'O'

The system authorization ID that is used to establish the trusted connection is obtained from the row in the SYSIBM.USERNAMES table with TYPE 'S'.

After successfully establishing the trusted connection, DB2 obtains the translated authorization ID for the primary authorization ID from the row in the SYSIBM.USERNAMES table with TYPE 'O'. DB2 sends the user switch request on behalf of the primary authorization ID with RACF passticket authentication.

If the SECURITY_OUT option is not correctly set up, DB2 returns an error.

Establishing remote trusted connections to DB2 for z/OS servers

When the DB2 for z/OS server receives a remote request to establish a trusted connection, DB2 checks to see if an authentication token accompanies the request.

The authentication token can be a password, a RACF passticket, or a Kerberos ticket. The requester goes through the standard authorization processing at the server. If the authorization is successful, DB2 invokes the connection exit routine, which associates the primary authorization ID, possibly one or more secondary authorization IDs, and an SQL ID with the remote request. DB2 searches for a matching trusted context. If DB2 finds a matching trusted context, it validates the following attributes:

- If the SERVAUTH attribute is defined for the identified trusted context and TCP/IP provides a RACF SERVAUTH profile name to DB2 during the establishment of the connection, DB2 matches the SERVAUTH profile name with the SERVAUTH attribute value.
- If the SERVAUTH attribute is not defined or the SERVAUTH name does not match the SERVAUTH that is defined for the identified trusted context, DB2 matches the remote client's TCP/IP address with the ADDRESS attribute that is defined for the identified trusted context.
- If the ENCRYPTION attribute is defined, DB2 validates whether the connection is using the proper encryption as specified in the value of the ENCRYPTION attribute.

- If the DEFAULT SECURITY LABEL attribute is defined for the system authorization ID, DB2 verifies the security label with RACF. This security label is used for verifying multilevel security for the system authorization ID. However, if the system authorization ID is also in the ALLOW USER clause with SECURITY LABEL, then that one is used.

If the validation is successful, DB2 establishes the connection as trusted. If the validation is not successful, the connection is established as a normal connection without any additional privileges, DB2 returns a warning, and SQLWARN8 is set.

Switching users of a trusted connection

When a trusted connection is established, DB2 enables the trusted connection to be reused under a different user on a transaction boundary. A trusted connection can be reused at a local DB2 subsystem by using RRSF, the DSN command processor under TSO, DB2I, and the SQL CONNECT statement with the USER and USING clauses.

To reuse a trusted connection, you must add the specific user to the trusted context. If you specify 'PUBLIC' as the user, DB2 allows the trusted connection to be used by any authorization ID; the trusted connection can be used by a different user with or without authentication. However, you can require authentication by specifying the WITH AUTHENTICATION clause.

You can use RRSF, the DSN command processor under TSO, and DB2I to switch to a new user on a trusted connection without authentication. If authentication is required, you can use the SQL CONNECT statement with the USER and USING clauses. The SQL CONNECT semantics prevent the use of CONNECT TO with the USER and USING clauses to switch authorization IDs on a remote connection.

Reusing a local trusted connection through the DSN command processor and DB2I

You can use the DSN command processor and DB2I to switch the user on a trusted connection if the DSN ASUSER option is specified.

DB2 establishes a trusted connection if the primary authorization ID and job name match a trusted context that is defined in DB2. The user ID that is specified for the ASUSER option goes through the standard authorization processing. If the user ID is authorized, DB2 runs the connection exit routine to associate the primary and secondary IDs.

DB2 then searches to see if the primary authorization ID is allowed to use the trusted connection without authentication. If the primary authorization ID is allowed to use the trusted connection without authentication, DB2 determines if the SECURITY LABEL attribute is defined in the trusted context for the user either explicitly or implicitly. If the SECURITY LABEL attribute is defined with a security label, DB2 verifies the security label with RACF. If the verification of the security label is successful, the trusted connection is established and used by the user ID that is specified for the ASUSER option. DB2 uses the security label for multilevel security verification for the user.

If the primary authorization ID that is associated with the user ID that is specified for the ASUSER option is not allowed or requires authentication information, the connection request fails. If the security label verification is not successful, the connection request fails.

Reusing a remote trusted connection by DB2 for z/OS requesters

After establishing a trusted connection with a system authorization ID, the DB2 for z/OS requester automatically switches the user on the connection to the primary authorization ID on the remote trusted connection in the following scenarios:

- The system authorization ID is different from the primary authorization ID that is associated with the application user.
- The system authorization ID is different from the authorization ID that is specified in the SQL CONNECT statement with the USER and USING clauses.
- Outbound translation is required for the primary authorization ID.

Reusing a remote trusted connection through DB2 for z/OS servers

The DB2 for z/OS server performs the following steps when it receives a request to switch users on a trusted connection:

1. DB2, on successful authorization, invokes the connection exit routine. The invocation associates the primary authorization ID, possibly one or more secondary authorization IDs, and an SQL ID with the remote request. This new set of IDs replaces the previous set of IDs that was associated with the request.
2. DB2 determines if the primary authorization ID is allowed to use the trusted connection. If the WITH AUTHENTICATION clause is specified for the user, DB2 requires an authentication token for the user. The authentication token can be a password, a RACF passticket, or a Kerberos ticket.
3. Assuming that the primary authorization ID is allowed, DB2 determines the trusted context for any SECURITY LABEL definition. If a specific SECURITY LABEL is defined for this user, it becomes the SECURITY LABEL for this user. If no specific SECURITY LABEL is defined for this user but a DEFAULT SECURITY LABEL is defined for the trusted context, DB2 verifies the validity of this SECURITY LABEL for this user through RACF by issuing the RACROUTE VERIFY request.

If the primary authorization ID is allowed, DB2 performs a connection initialization. This results in an application environment that truly mimics the environment that is initialized if the new user establishes the connection in the normal DB2 manner. For example, any open cursor is closed, and temporary table information is dropped.

4. If the primary authorization ID is not allowed to use the trusted connection or if SECURITY LABEL verification fails, the connection is returned to an unconnected state. The only operation that is allowed is to establish a valid authorization ID to be associated with the trusted connection. Until a valid authorization is established, if any SQL statement is issued, an error (SQLCODE -900) is returned.

Reusing a local trusted connection through RRSAP

If you use Resource Recovery Services Attachment Facility (RRSAF) to switch to a new user on a trusted connection, DB2 obtains the primary authorization ID and runs the sign-on exit routine.

DB2 then searches to determine if the primary authorization ID is allowed to use the trusted connection without authentication. If the primary authorization ID is allowed, DB2 determines if SECURITY LABEL is explicitly or implicitly defined in the trusted context for the user. If SECURITY LABEL is defined, DB2 verifies the

SECURITY LABEL with RACF by using the RACROUTE VERIFY request. If the SECURITY LABEL verification is successful, the trusted connection is used by the new user.

If the primary authorization ID is not allowed to use the trusted connection without authentication, DB2 returns the connection to an unconnected state. The only action that you can take is to try running the sign-on exit routine again. Until a valid authorization is established, any SQL statement that you issue causes DB2 to return an error.

Reusing a local trusted connection through the SQL CONNECT statement

You can switch users on a trusted connection by using the SQL CONNECT statement with the USER and USING clauses.

DB2, on successful authorization, invokes the connection exit routine if it is defined. The connection then has a primary authorization ID, zero or more secondary IDs, and an SQL ID.

DB2 searches to determine if the primary authorization ID is allowed to use the trusted connection. If the primary authorization ID is allowed, DB2 determines if the SECURITY LABEL attribute is defined in the trusted context for the user either explicitly or implicitly. If the SECURITY LABEL attribute is defined with a security label, DB2 verifies the security label with RACF. If the security label verification is successful, DB2 switches the user on the trusted connection. DB2 uses the security label for multilevel security verification for the user.

If the primary authorization ID is not allowed to use the trusted connection or if the security label verification is not successful, DB2 returns the connection to an unconnected state. The only action you can take is to establish a valid authorization ID to be associated with the trusted connection. Until a valid authorization is established, any SQL statement that you issue causes DB2 to return an error.

Defining external security profiles

You can control the users who can be switched in a trusted connection by defining an external security profile in RACF and authorizing users to use the profile.

To define an external security profile in RACF:

1. Create a general resource profile in RACF for the DSNR class by issuing the following command:

```
RDEFINE DSNR (TRUSTEDCTX.PROFILE1) UACC(NONE)
```
2. Add users to the TRUSTEDCTX.PROFILE1 profile and define their level of access authority by issuing the following command:

```
PERMIT TRUSTEDCTX.PROFILE1 CLASS(DSNR) ID(USER1 USER2) ACCESS(READ)
```
3. Associate the profile with the trusted context definition by using the EXTERNAL SECURITY PROFILE keyword in the trusted context user clause definition.

You can remove users who can be switched in a trusted connection individually from the TRUSTEDCTX.PROFILE1 profile in RACF. You can also remove all users by simply dissociating the profile from the trusted context definition.

Enabling users to perform actions on behalf of others

Within a trusted context, you can allow users to perform actions on objects on behalf of others.

You can specify the DSN ASUSER option with the authorization ID of the object owner. During the connection processing, the authorization ID is used to determine if a trusted context exists for this authorization ID. If a trusted context exists, a trusted connection is established. The primary authorization ID that is associated with the user ID and specified in the ASUSER option is used to determine if the user can be switched on the trusted connection.

If the user ID that is specified in the ASUSER option is allowed to use the trusted connection, the user runs under the authorization ID of the object owner and can perform actions on behalf of the object owner. The authorization ID of the original user is traced for audit purposes.

Performing tasks on objects for other users

If you have DBADM authority, you can assume the identity of other users within a trusted context and perform tasks on their behalf.

After you successfully assume the identity of a view owner, you inherit all the privileges from the ID that owns the view and can therefore perform the CREATE, DROP, and GRANT actions on the view.

To perform tasks on behalf of another user:

1. Define a trusted context. Make sure that the SYSTEM AUTH ID is the primary authorization ID that you use in SPUFI.
2. Specify the primary authorization ID as the JOBNAME for the trusted connection
3. Specify the primary authorization ID of the user whose identity you want to assume
4. Log onto TSO with your primary authorization ID
5. Set the ASUSER option on the DB2I DEFAULTS panel to the primary authorization ID of the user whose identity you want to assume
6. Perform the desired actions by using privileges of the specified user.

Assume that you have DBADM authority, your primary authorization ID is BOB, and you want to drop a view that is owned by user SALLY. You can issue the following statement to create and enable a trusted context called CTXLOCAL in which BOB can drop the selected view on SALLY's behalf:

```
CREATE TRUSTED CONTEXT CTXLOCAL  
  BASED UPON CONNECTION USING SYSTEM AUTHID BOB  
  ATTRIBUTES (JOBNAME 'BOB')  
  ENABLE  
  ALLOW USE FOR SALLY;
```

After logging onto TSO, you can set the ASUSER option to SALLY in the DB2I DEFAULTS panel and invoke SPUFI to process SQL statements. DB2 obtains the primary authorization ID BOB and JOBNAME BOB from the TSO log-on session, authenticates BOB, searches for the matching trusted context (CTXLOCAL), and establishes a trusted connection. DB2 then authenticates the primary authorization ID SALLY and validates all privileges that are assigned to SALLY. After successful authentication and validation, you, BOB, can drop the view that is owned by

I

SALLY.

Chapter 8. Managing access through data definition control

You can use the DB2 *data definition control* to manage access to your DB2 data. Data definition control is a security measure that provides additional constraints to existing authorization checks. With data definition control, you can regulate how specific plans or collections of packages can use data definition statements.

Data definition statements

Data definition statements are a subset of statements that are referred to as *data definition language*.

The following data definition statements are controlled through the DB2 data definition control support.

Table 83. Data definition Statements

Object	CREATE statement	ALTER statement	DROP statement
Alias	CREATE ALIAS		DROP ALIAS
Database	CREATE DATABASE	ALTER DATABASE	DROP DATABASE
Index	CREATE INDEX	ALTER INDEX	DROP INDEX
Storage group	CREATE STOGROUP	ALTER STOGROUP	DROP STOGROUP
Synonym	CREATE SYNONYM		DROP SYNONYM
Table	CREATE TABLE	ALTER TABLE	DROP TABLE
Table space	CREATE TABLESPACE	ALTER TABLESPACE	DROP TABLESPACE
View	CREATE VIEW		DROP VIEW

The data definition control support also controls the COMMENT and LABEL statements.

Data definition control support

If you want to use data definition statements for your plans and packages, you must install the *data definition control support* on the DB2 DSNTIPZ installation panel.

As shown in the following example, you can specify appropriate values for several installation options to install the data definition control support and to control data definition behaviors.

```

DSNTIPZ      INSTALL DB2 - DATA DEFINITION CONTROL SUPPORT
===>

Enter data below:

 1 INSTALL DD CONTROL SUPT. ===> NO          YES - activate the support
                                         NO - omit DD control support
 2 CONTROL ALL APPLICATIONS ===> NO          YES or NO
 3 REQUIRE FULL NAMES      ===> YES          YES or NO
 4 UNREGISTERED DDL DEFAULT ===> ACCEPT      Action for unregistered DDL:
                                         ACCEPT - allow it
                                         REJECT - prohibit it
                                         APPL - consult ART
 5 ART/ORT ESCAPE CHARACTER ===>            Used in ART/ORT Searches
 6 REGISTRATION OWNER      ===> DSNRGCOL      Qualifier for ART and ORT
 7 REGISTRATION DATABASE   ===> DSNRGFDB      Database name
 8 APPL REGISTRATION TABLE ===> DSN_REGISTER_APPL Table name
 9 OBJT REGISTRATIION TABLE ===> DSN_REGISTER_OBJT Table name

Note: ART = Application Registration Table
      ORT = Object      Registration Table

PRESS: ENTER to continue  RETURN to exit  HELP for more information

```

Figure 27. DSNTIPZ installation panel with default values

Registration tables

If you use data definition control support, you must create and maintain a *application registration table (ART)* and a *object registration table (ORT)*.

You can register the plans and package collections in the ART, and you can register the objects that are associated with the plans and collections in the ORT. DB2 consults these two registration tables before accepting a data definition statement from a process. It denies a request to create, alter, or drop a particular object if the registration tables indicate that the process is not allowed to do so.

ART columns

The columns of the ART are described in the following table. The CREATOR and CHANGER columns are CHAR(26) and large enough for a three-part authorization ID. You need to separate each 8-byte part of the ID with a period in byte 9 and in byte 18. If you enter only the primary authorization ID, consider entering it right-justified in the field (that is, preceded by 18 blanks). Some columns are optional and reserved for administrator use; DB2 does not use these columns.

Table 84. Columns of the ART

Column name	Description
APPLIDENT	Indicates the collection-ID of the package that executes the data definition language. If no package exists, it indicates the name of the plan that executes the data definition language.
APPLIDENTTYPE	Indicates the type of application identifier.
APPLICATIONDESC ¹	Optional data. Provides a more meaningful description of each application than the eight-byte APPLIDENT column can contain.
DEFAULTAPPL	Indicates whether all data definition language should be accepted from this application.

Table 84. Columns of the ART (continued)

Column name	Description
QUALIFIEROK	Indicates whether the application can supply a missing name part for objects that are named in the ORT. Applies only if REQUIRE FULL NAMES = NO.
CREATOR ^{1,2}	Optional data. Indicates the authorization ID that created the row.
CREATETIMESTAMP ¹	Optional data. Indicates when a row was created. If you use CURRENT TIMESTAMP, DB2 automatically enters the value of CURRENT TIMESTAMP When you load or insert a row.
CHANGER ^{1,2}	Optional data. Indicates the authorization ID that last changed the row.
CHANGETIMESTAMP ¹	Optional data. Indicates when a row was changed. If you use CURRENT TIMESTAMP, DB2 automatically enters the value of CURRENT TIMESTAMP When you update a row.

ORT columns

The columns of the ORT are described in the following table. The CREATOR and CHANGER columns are CHAR(26) which is large enough for a three-part authorization ID. You need to separate each 8-byte part of the ID with a period in byte 9 and in byte 18. If you enter only the primary authorization ID, consider entering it right-justified in the field (that is, preceded by 18 blanks). Some columns are optional and reserved for administrator use; DB2 does not use these columns.

Table 85. Columns of the ORT

Column name	Description
QUALIFIER	Indicates the object name qualifier.
NAME	Indicates the unqualified object name.
TYPE	Indicates the type of object.
APPLMATCHREQ	Indicates whether an application that names this object must match the one that is named in the APPLIDENT column.
APPLIDENT	Collection-ID of the plan or package that executes the data definition language.
APPLIDENTTYPE	Indicates the type of application identifier.
APPLICATIONDESC ¹	Optional data. Provides a more meaningful description of each application than the eight-byte APPLIDENT column can contain.
CREATOR ^{1,2}	Optional data. Indicates the authorization ID that created the row.
CREATETIMESTAMP ¹	Optional data. Indicates when a row was created. If you use CURRENT TIMESTAMP, DB2 automatically enters the value of CURRENT TIMESTAMP When you load or insert a row.
CHANGER ^{1,2}	Optional data. Indicates the authorization ID that last changed the row.
CHANGETIMESTAMP ¹	Optional data. Indicates when a row was changed. If you use CURRENT TIMESTAMP, DB2 automatically enters the value of CURRENT TIMESTAMP When you update a row.

Installing data definition control support

To install data definition control support:

1. Enter YES for option 1 on the DSNTIPZ installation panel, as shown in the following example.
1 INSTALL DD CONTROL SUPT. ====> YES
2. Enter the names and owners of the registration tables in your DB2 subsystem and the databases in which these tables reside for options 6, 7, 8, and 9 on the DSNTIPZ installation panel.

The default values for these options are as follows:

```
6 REGISTRATION OWNER      ====> DSNRGCOL
7 REGISTRATION DATABASE   ====> DSNRGFDB
8 APPL REGISTRATION TABLE ====> DSN_REGISTER_APPL
9 OBJT REGISTRATION TABLE ====> DSN_REGISTER_OBJT
```

You can accept the default names or assign names of your own. If you specify your own table names, each name can have a maximum of 17 characters.

3. Enter an escape character for option 5 on the DSNTIPZ installation panel if you want to use the percent character (%) or the underscore character (_) as a regular character in the ART or ORT.

You can use any special character other than underscore or percent as the escape character. For example, you can use the pound sign (#) as an escape character. If you do, the value for option looks like this:

```
5 ART/ORT ESCAPE CHARACTER ====> #
```

After you specify the pound sign as an escape character, the pound sign can be used in names in the same way that an escape character is used in an SQL LIKE predicate.

4. Register plans, packages, and objects in the ART and ORT.
Choose the plans, packages, and objects to register based on whether you want to control data definition by application name or object name.
5. Enter the values for the three other options on the DSNTIPZ installation panel as follows:

```
2 CONTROL ALL APPLICATIONS ====>
3 REQUIRE FULL NAMES          ====>
4 UNREGISTERED DDL DEFAULT    ====>
```

Enabling data definition control

You can use data definition control after you install the DB2 data definition control support and create the ART and ORT.

You can use data definition control in the following four ways:

- Controlling data definition by application name
- Controlling data definition by application name with exceptions
- Controlling data definition by object name
- Controlling data definition by object name with exceptions

Controlling data definition by application name

The simplest way to implement data definition control is to give one or more applications total control over the use of data definition statements in the subsystem.

To control data definition by application name, perform the following steps:

1. Enter YES for the first option on the DSNTIPZ installation panel, as shown:
2 CONTROL ALL APPLICATIONS ==> YES

When you specify YES, only package collections or plans that are registered in the ART are allowed to use data definition statements.

2. In the ART, register all package collections and plans that you will allow to issue DDL statements, and enter the value Y in the DEFAULTAPPL column for these package collections. You must supply values for the APPLIDENT, APPLIDENTTYPE, and DEFAULTAPPL columns of the ART. You can enter information in other columns for your own use.

Example: Suppose that you want all data definition language in your subsystem to be issued only through certain applications. The applications are identified by the following application plan names, collection-IDs, and patterns:

PLANA

The name of an application plan

PACKB

The collection-ID of a package

TRULY%

A pattern name for any plan name beginning with TRULY

TR%

A pattern name for any plan name beginning with TR

The following table shows the entries that you need in your ART.

Table 86. Table DSN_REGISTER_APPL for total subsystem control

APPLIDENT	APPLIDENTTYPE	DEFAULTAPPL
PLANA	P	Y
PACKB	C	Y
TRULY%	P	Y
TR%	P	Y

If the row with TR% for APPLIDENT contains the value Y for DEFAULTAPPL, any plan with a name beginning with TR can execute data definition language. If DEFAULTAPPL is later changed to N to disallow that use, the changed row does not prevent plans beginning with TR from using data definition language; the row merely fails to allow that specific use. In this case, the plan TRXYZ is not allowed to use data definition language. However, the plan TRULYXYZ is allowed to use data definition language, by the row with TRULY% specified for APPLIDENT.

Controlling data definition by application name with exceptions

Registering application names with some exceptions is one of four methods for controlling data definition. If you want to give one or more applications almost total control over data definition language on all objects, you can reserve a few objects that can be created, altered, or dropped by applications that are not registered in the ART.

To control data definition by application name with exceptions, perform the following steps:

1. Choose not to control all applications. On the DSNTIPZ installation panel, specify the following value for option 2:

2 CONTROL ALL APPLICATIONS ====> NO

When you specify NO, you allow unregistered applications to use data definition statements on some objects.

2. On the DSNTIPZ installation panel, specify the following for option 4:

4 UNREGISTERED DDL DEFAULT ====> APPL

When you specify APPL, you restrict the use of data definition statements for objects that are **not** registered in the ORT. If an object is registered in the ORT, any applications that are not registered in the ART can use data definition language on the object. However, if an object is not registered in the ORT, only applications that are registered in the ART can use data definition language on the object.

3. In the ART, register package collections and plans that you will allow to issue data definition statements on any object. Enter the value Y in the DEFAULTAPPL column for these package collections. Applications that are registered in the ART retain almost total control over data definition. Objects that are registered in the ORT are the only exceptions.
4. In the ORT, register all objects that are exceptions to the subsystem data definition control that you defined in the ART. You must supply values for the QUALIFIER, NAME, TYPE, APPLMATCHREQ, APPLIDENT, and APPLIDENTTYPE columns of the ORT. You can enter information in other columns of the ORT for your own use.

Example: Suppose that you want almost all of the data definition language in your subsystem to be issued only through an application plan (PLANA) and a package collection (PACKB).

The following table shows the entries that you need in your ART.

Table 87. Table DSN_REGISTER_APPL for total subsystem control with exceptions

APPLIDENT	APPLIDENTTYPE	DEFAULTAPPL
PLANA	P	Y
PACKB	C	Y

However, suppose that you also want the following specific exceptions:

- Object KIM.VIEW1 can be created, altered, or dropped by the application plan PLANC.
- Object BOB.ALIAS can be created, altered, or dropped only by the package collection PACKD.
- Object FENG.TABLE2 can be created, altered, or dropped by **any** plan or package collection.
- Objects with names that begin with SPIFFY.MSTR and exactly one following character can be created, altered, or dropped by any plan that matches the name pattern TRULY%. For example, the plan TRULYJKL can create, alter, or drop the object SPIFFY.MSTRA.

The following table shows the entries that are needed to register these exceptions in the ORT.

Table 88. Table DSN_REGISTER_OBJT for subsystem control with exceptions

QUALIFIER	NAME	TYPE	APPLMATCHREQ	APPLIDENT	APPLIDENTTYPE
KIM	VIEW1	C	Y	PLANC	P
BOB	ALIAS	C	Y	PACKD	C
FENG	TABLE2	C	N		
SPIFFY	MSTR_	C	Y	TRULY%	P

You can register objects in the ORT individually, or you can register sets of objects.

Controlling data definition by object name

Registering object names is one of four methods for controlling data definition. If you want all objects in the subsystem to be registered and you want several applications to control specific sets of objects, you need to control by object name.

When you control by object name, all objects are registered regardless of whether they are controlled by specific applications. To control data definition by object name, perform the following steps:

1. Choose not to control all applications. On the DSNTIPZ installation panel, specify the following value for option 2:
2 CONTROL ALL APPLICATIONS ==> NO

When you specify NO, you allow unregistered applications to use data definition statements on some objects.

2. On the DSNTIPZ installation panel, fill in option 4 as follows:
4 UNREGISTERED DDL DEFAULT ==> REJECT

When you specify REJECT for option 4, you totally restrict the use of data definition statements for objects that are not registered in the ORT. Therefore, no application can use data definition statements for any unregistered object.

3. In the ORT, register all of the objects in the subsystem, and enter Y in the APPLMATCHREQ column. You must supply values for the QUALIFIER, NAME, TYPE, APPLMATCHREQ, APPLIDENT, and APPLIDENTTYPE columns of the ORT. You can enter information in other columns of the ORT for your own use.
4. In the ART, register any plan or package collection that can use a set of objects that you register in the ORT with an incomplete name. Enter the value Y in the QUALIFIEROK column. These plans or package collections can use data definition language on sets of objects regardless of whether a set of objects has a value of Y in the APPLMATCHREQ column.

Example: The following table shows entries in the ORT for a DB2 subsystem that contains the following objects that are controlled by object name:

- Two storage groups (STOG1 and STOG2) and a database (DATB1) that are not controlled by a specific application. These objects can be created, altered, or dropped by a user with the appropriate authority by using any application, such as SPUFI or QMF.
- Two table spaces (TBSP1 and TBSP2) that are not controlled by a specific application. Their names are qualified by the name of the database in which they reside (DATB1).
- Three objects (OBJ1, OBJ2, and OBJ3) whose names are qualified by the authorization IDs of their owners. Those objects might be tables, views, indexes,

synonyms, or aliases. Data definition statements for OBJ1 and OBJ2 can be issued only through the application plan named PLANX. Data definition statements for OBJ3 can be issued only through the package collection named PACKX.

- Objects that match the qualifier pattern E%D and the name OBJ4 can be created, altered, or deleted by application plan SPUFI. For example, the objects EDWARD.OBJ4, ED.OBJ4, and EBHARD.OBJ4, can be created, altered, or deleted by application plan SPUFI. Entry E%D in the QUALIFIER column represents all three objects.
- Objects with names that begin with TRULY.MY_, where the underscore character is actually part of the name. Assuming that you specify # as the escape character, all of the objects with this name pattern can be created, altered, or dropped only by plans with names that begin with TRULY.

Table 89. Table DSN_REGISTER_OBJT for total control by object

QUALIFIER	NAME	TYPE	APPLMATCHREQ	APPLIDENT	APPLIDENTTYPE
	STOG1	S	N		
	STOG2	S	N		
	DATB1	D	N		
DATB1	TBSP1	T	N		
DATB1	TBSP2	T	N		
KIM	OBJ1	C	Y	PLANX	P
FENG	OBJ2	C	Y	PLANX	P
QUENTIN	OBJ3	C	Y	PACKX	C
E%D	OBJ4	C	Y	SPUFI	P
TRULY	MY#_%	C	Y	TRULY%	P

Assume the following installation option:

```
3 REQUIRE FULL NAMES      ==> YES
```

The entries do not specify incomplete names. Hence, objects that are not represented in the table cannot be created in the subsystem, except by an ID with installation SYSADM authority.

Controlling data definition by object name with exceptions

Registering object names with some exceptions is one of four methods for controlling data definition.

If you want several applications to control specific sets of registered objects and to allow other applications to use data definition statements for unregistered objects, perform the following steps:

1. Choose not to control all applications. On the DSNTIPZ installation panel, specify the following value for option 2:

```
2 CONTROL ALL APPLICATIONS ==> NO
```

When you specify NO, you allow unregistered applications to use data definition statements on some objects.

2. On the DSNTIPZ installation panel, fill in option 4 as follows:

```
4 UNREGISTERED DDL DEFAULT ==> ACCEPT
```

This option **does not** restrict the use of data definition statements for objects that are not registered in the ORT. Therefore, **any** application can use data definition language for any unregistered object.

3. Register all controlled objects in the ORT. Use a name and qualifier to identify a single object. Use only one part of a two-part name to identify a set of objects that share just that part of the name. For each controlled object, use APPLMATCHREQ = Y. Enter the name of the plan or package collection that controls the object in the APPLIDENT column.
4. For each set of controlled objects (identified by only a simple name in the ORT), register the controlling application in the ART. You must supply values for the APPLIDENT, APPLIDENTTYPE, and QUALIFIEROK columns of the ART.

Example: The following two tables assume that the installation option REQUIRE FULL NAMES is set to NO. The following table shows entries in the ORT for the following controlled objects:

Table 90. Table DSN_REGISTER_OBJT for object control with exceptions

QUALIFIER	NAME	TYPE	APPLMATCHREQ	APPLIDENT	APPLIDENTTYPE
KIM	OBJ1	C	Y	PLANX	P
FENG	OBJ2	C	Y	PLANX	P
QUENTIN	OBJ3	C	Y	PACKX	C
EDWARD	OBJ4	C	Y	PACKX	C
	TABA	C	Y	PLANA	P
	TABB	C	Y	PACKB	C

- The objects KIM.OBJ1, FENG.OBJ2, QUENTIN.OBJ3, and EDWARD.OBJ4, all of which are controlled by PLANX or PACKX. DB2 cannot interpret the object names as incomplete names because the objects that control them, PLANX and PACKX, are registered, with QUALIFIEROK=N, in the corresponding ART as shown in the following table:

Table 91. Table DSN_REGISTER_APPL for object control with exceptions

APPLIDENT	APPLIDENTTYPE	DEFAULTAPPL	QUALIFIEROK
PLANX	P	N	N
PACKX	C	N	N
PLANA	P	N	Y
PACKB	C	N	Y

In this situation, with the combination of installation options shown previously, any application can use data definition language for objects that are not covered by entries in the ORT. For example, if HOWARD has the CREATETAB privilege, HOWARD can create the table HOWARD.TABLE10 through any application.

- Two sets of objects, *.TABA and *.TABB, are controlled by PLANA and PACKB, respectively.

Registering object sets

Registering object sets enables you to save time and to simplify object registration.

Registering object sets is not a data definition control method; you must install of the data definition control methods before you can register any object sets.

Because complete two-part names are not required for every object that is registered in the ORT, you can use incomplete names to register sets of objects. To use incomplete names and register sets of objects, fill in option 3 on the DSNTIPZ installation panel as follows:

```
3 REQUIRE FULL NAMES      ==> NO
```

The default value YES requires you to use both parts of the name for each registered object. If you specify the value NO, an incomplete name in the ORT represents a set of objects that all share the same value for one part of a two-part name. Objects that are represented by incomplete names in the ORT require an authorizing entry in the ART.

Example: If you specify NO for option 3, you can include entries with incomplete names in the ORT. The following table shows entries in the ORT for the following objects:

Table 92. Table DSN_REGISTER_OBJT for objects with incomplete names

QUALIFIER	NAME	TYPE	APPLMATCHREQ	APPLIDENT	APPLIDENTTYPE
	TABA	C	Y	PLANX	P
	TABB	C	Y	PACKY	C
SYSADM		C	N		
DBSYSADM		T	N		
USER1	TABLEX	C	N		

- Two sets of objects, *.TABA and *.TABB, which are controlled by PLANX and PACKY, respectively. Only PLANX can create, alter, or drop any object whose name is *.TABA. Only PACKY can create, alter, or drop any object whose name is *.TABB. PLANX and PACKY must also be registered in the ART with QUALIFIEROK set to Y, as shown in the following table: That setting allows the applications to use sets of objects that are registered in the ORT with an incomplete name.

Table 93. Table DSN_REGISTER_APPL for plans that use sets of objects

APPLIDENT	APPLIDENTTYPE	DEFAULTAPPL	QUALIFIEROK
PLANA	P	N	Y
PACKB	C	N	Y

- Tables, views, indexes, or aliases with names like SYSADM.*.
- Table spaces with names like DBSYSADM.*; that is, table spaces in database DBSYSADM.
- Tables with names like USER1.* **and** tables with names like *.TABLEX.

ART entries for objects with incomplete names in the ORT: APPLMATCHREQ=N and objects SYSADM.*, DBSYSADM.*, USER1.*, and *.TABLEX can be created, altered, or dropped by any package collection or application plan. However, the collection or plan that creates, alters, or drops such an object must be registered in the ART with QUALIFIEROK=Y to allow it to use incomplete object names.

Disabling data definition control

When data definition control is active, only IDs with the installation SYSADM or installation SYSOPR authority are able to stop a database, a table space, or an index space that contains a registration table or index.

When the object is stopped, only an ID with one of those authorities can start it again.

An ID with the installation SYSADM authority can execute data definition statements regardless of whether data definition control is active and whether the ART or ORT is available. To bypass data definition control, an ID with the installation SYSADM authority can use the following methods:

- If the ID is the owner of the plan or package that contains the statement, the ID can bypass data definition control by using a static SQL statement.
- If the ID is the current SQL ID, the ID can bypass data definition control through a dynamic CREATE statement.
- If the ID is the current SQL ID, the primary ID, or any secondary ID of the executing process, the ID can bypass data definition control through a dynamic ALTER or DROP statement.

Managing registration tables and indexes

You can create, update, and drop registration tables and indexes. You can also create table spaces for or add columns to registration tables.

Creating registration tables and indexes

The ART, the ORT, and the required unique indexes on them are created when you install the data definition control support. If you drop any of these objects, you can re-create them.

You can use the following CREATE statements to recreate ART, the ORT, or the required unique indexes:

CREATE statements for the ART and its index:

```
CREATE TABLE DSNRGCOL.DSN_REGISTER_APPL
  (APPLIDENT      VARCHAR(128) NOT NULL WITH DEFAULT,
   APPLIDENTTYPE  CHAR(1)      NOT NULL WITH DEFAULT,
   APPLICATIONDESC VARCHAR(30)  NOT NULL WITH DEFAULT,
   DEFAULTAPPL    CHAR(1)      NOT NULL WITH DEFAULT,
   QUALIFIEROK    CHAR(1)      NOT NULL WITH DEFAULT,
   CREATOR        CHAR(26)     NOT NULL WITH DEFAULT,
   CREATETIMESTAMP  TIMESTAMP  NOT NULL WITH DEFAULT,
   CHANGER        CHAR(26)     NOT NULL WITH DEFAULT,
   CHANGETIMESTAMP  TIMESTAMP  NOT NULL WITH DEFAULT)
IN DSNRGFDB.DSNRGFTS;

CREATE UNIQUE INDEX DSNRGCOL.DSN_REGISTER_APPLI
ON DSNRGCOL.DSN_REGISTER_APPL
  (APPLIDENT, APPLIDENTTYPE, DEFAULTAPPL DESC, QUALIFIEROK DESC)
CLUSTER;
```

CREATE statements for the ORT and its index:

```
CREATE TABLE DSNRGCOL.DSN_REGISTER_OBJT
  (QUALIFIER      CHAR(8)      NOT NULL WITH DEFAULT,
   NAME           CHAR(18)     NOT NULL WITH DEFAULT,
   TYPE           CHAR(1)      NOT NULL WITH DEFAULT,
   APPLMATCHREQ   CHAR(1)      NOT NULL WITH DEFAULT,
   APPLIDENT      VARCHAR(128) NOT NULL WITH DEFAULT,
   APPLIDENTTYPE  CHAR(1)      NOT NULL WITH DEFAULT,
   APPLICATIONDESC VARCHAR(30)  NOT NULL WITH DEFAULT,
   CREATOR        CHAR(26)     NOT NULL WITH DEFAULT,
```

```

    CREATETIMESTAMP    TIMESTAMP    NOT NULL WITH DEFAULT,
    CHANGER            CHAR(26)      NOT NULL WITH DEFAULT,
    CHANGETIMESTAMP    TIMESTAMP    NOT NULL WITH DEFAULT)
IN DSNRGFDB.DSNRGFTS;

CREATE UNIQUE INDEX DSNRGCOL.DSN_REGISTER_OBJTI
ON DSNRGCOL.DSN_REGISTER_OBJT
(QUALIFIER, NAME, TYPE) CLUSTER;

```

You can alter these CREATE statements in the following ways:

- Add columns to the ends of the tables
- Assign an auditing status
- Choose buffer pool or storage options for indexes
- Declare table check constraints to limit the types of entries that are allowed

Naming registration tables and indexes

Every member of a data sharing group must have the same names for the ART and ORT tables. Avoid changing the names of the ART and ORT tables.

If you change the names, owners, or residing database of your ART and ORT, you must reinstall DB2 in update mode and make the corresponding changes on the DSNTIPZ installation panel.

Name the required index by adding the letter I to the corresponding table name. For example, suppose that you are naming a required index for the ART named ABC. You should name the required index ABCI.

Dropping registration tables and indexes

If you drop any of the registration tables or their indexes, most data definition statements are rejected until the dropped objects are re-created.

The only data definition statements that are allowed under such circumstances are those that create the following objects:

- Registration tables that are defined during installation
- Indexes of the registration tables that are defined during installation
- Table spaces that contain the registration tables that are defined during installation
- The database that contains the registration tables that are defined during installation

Creating table spaces for registration tables

The DSNTIJSJG installation job creates a segmented table space to hold the ART and ORT when you issue this statement:

```
CREATE TABLESPACE DSNRGFTS IN DSNRGFDB SEGSIZE 4 CLOSE NO;
```

If you want to use a table space with a different name or different attributes, you can modify the DSNTIJSJG job before installing DB2. Alternatively, you can drop the table space and re-create it, the ART and ORT tables, and their indexes.

Adding columns to registration tables

You can use the ALTER TABLE statement to add columns to the ART or ORT for your own use. If you add columns, the additional columns must come at the end of the table, after existing columns.

Use a special character, such as the plus sign (+), in your column names to avoid possible conflict. If IBM adds columns to the ART or the ORT in future releases, the column names will contain only letters and numbers.

Updating registration tables

You can use either the LOAD utility or with the INSERT, UPDATE, or DELETE SQL statements to update the ART or ORT.

Because security provisions are important, allow only a restricted set of authorization IDs, or perhaps only those with the SYSADM authority, to update the ART. Consider assigning a validation exit routine to the ORT, to allow applications to change only those rows that have the same application identifier in the APPLIDENT column.

A registration table cannot be updated until all jobs whose data definition statements are controlled by the table have completed.

Chapter 9. Protecting data through encryption and RACF

You can use the DB2 built-in data encryption functions or Secure Socket Layer (SSL) support to protect your sensitive data. You can also use the security features of RACF or an equivalent system to protect your data sets.

Encrypting your data through DB2 built-in functions

DB2 provides built-in data encryption and decryption functions that you can use to encrypt sensitive data, such as credit card numbers and medical record numbers.

You can encrypt data at the column or value level. You must install the Integrated Cryptographic Service Facility to use the built-in functions for data encryption.

When you use data encryption, DB2 requires the correct password to retrieve the data in a decrypted format. If an incorrect password is provided, DB2 does not decrypt the data.

The ENCRYPT keyword encrypts data. The DECRYPT_BIT, DECRYPT_CHAR, and DECRYPT_DB keywords decrypt data. These functions work like other built-in functions. To use these functions on data, the column that holds the data must be properly defined.

Built-in encryption functions work for data that is stored within DB2 subsystem and is retrieved from within that same DB2 subsystem. The encryption functions do not work for data that is passed into and out of a DB2 subsystem. This task is handled by DRDA data encryption, and it is separate from built-in data encryption functions.

Attention: DB2 cannot decrypt data without the encryption password, and DB2 does not store encryption passwords in an accessible format. If you forget the encryption password, you cannot decrypt the data, and the data might become unusable.

Defining columns for encrypted data

When data is encrypted, it is stored as a binary data string. Therefore, encrypted data should be stored in columns that are defined as VARCHAR FOR BIT DATA.

Columns that hold encrypted data also require additional bytes to hold a header and to reach a multiple of 8 bytes.

Suppose that you have non-encrypted data in a column that is defined as VARCHAR(6). Use the following calculation to determine the column definition for storing the data in encrypted format:

Maximum length of non-encrypted data	6 bytes
Number of bytes to the next multiple of 8	2 bytes
24 bytes for encryption key	24 bytes

Encrypted data column length	32 bytes

Therefore, define the column for encrypted data as VARCHAR(32) FOR BIT DATA.

If you use a password hint, DB2 requires an additional 32 bytes to store the hint. Suppose that you have non-encrypted data in a column that is defined as VARCHAR(10). Use the following calculation to determine the column definition for storing the data in encrypted format with a password hint:

Maximum length of non-encrypted data	10 bytes
Number of bytes to the next multiple of 8	6 bytes
24 bytes for encryption key	24 bytes
32 bytes for password hint	32 bytes

Encrypted data column length	72 bytes

Therefore, define the column for encrypted data as VARCHAR(72) FOR BIT DATA.

Defining column-level encryption

For *column-level encryption*, all encrypted values in a column are encrypted with the same password.

Column-level encryption uses the SET ENCRYPTION PASSWORD statement to manage passwords and hints, the ENCRYPT keyword to indicate which data should be encrypted, and the DECRYPT_BIT, DECRYPT_CHAR, or DECRYPT_DB keyword to decrypt data. The following statement and keywords are used with column-level encryption:

SET ENCRYPTION PASSWORD

Sets the password (and optionally sets the password hint) that DB2 holds for encryption and decryption. DB2 holds this value until it is replaced or until the application finishes.

Recommendation: Use host variables instead of literal values for all passwords and password hints. If statements contain literal values for passwords and password hints, the security of the encrypted data can be compromised in the DB2 catalog and in a trace report.

ENCRYPT

Indicates which column or columns require encryption. DB2 sets the password on the indicated data to the password that DB2 holds at the time a statement with the ENCRYPT keyword is issued.

DECRYPT_BIT, DECRYPT_CHAR, DECRYPT_DB

Checks for the correct password and decrypts data when the data is selected.

When encrypted data is selected, DB2 must hold the same password that was held at the time of encryption to decrypt the data. To ensure that DB2 holds the correct password, issue a SET ENCRYPTION PASSWORD statement with the correct password immediately before selecting encrypted data.

Example: Suppose that you need to create an employee table EMP that contains employee ID numbers in encrypted format. Suppose also that you want to set the password for all rows in an encrypted column to the host variable hv_pass. Finally, suppose that you want to select employee ID numbers in decrypted format. Perform the following steps:

1. Create the EMP table with the EMPNO column. The EMPNO column must be defined with the VARCHAR data type, must be defined FOR BIT DATA, and must be long enough to hold the encrypted data. The following statement creates the EMP table:

```
CREATE TABLE EMP (EMPNO VARCHAR(32) FOR BIT DATA);
```

2. Set the encryption password. The following statement sets the encryption password to the host variable :hv_pass:

```
SET ENCRYPTION PASSWORD = :hv_pass;
```
3. Use the ENCRYPT keyword to insert encrypted data into the EMP table by issuing the following statements:

```
INSERT INTO EMP (EMPNO) VALUES(ENCRYPT('47138'));
INSERT INTO EMP (EMPNO) VALUES(ENCRYPT('99514'));
INSERT INTO EMP (EMPNO) VALUES(ENCRYPT('67391'));
```
4. Select the employee ID numbers in decrypted format:

```
SELECT DECRYPT_CHAR(EMPNO) FROM EMP;
```

If you provide the correct password, DB2 returns the employee ID numbers in decrypted format.

Creating views with column-level encryption

You can use column-level encryption in combination with views. You can create a view that selects decrypted data from a table.

To do this, define the view with a decryption function in the defining fullselect. If the correct password is provided when the view is queried, DB2 will return decrypted data.

Example: Suppose that you want to create a view that contains decrypted employee ID numbers from the EMP table.

1. Create a view on the EMP table by using the following statement:

```
CREATE VIEW CLR_EMP (EMPNO) AS SELECT DECRYPT_CHAR(EMPNO) FROM EMP;
```
2. Set the encryption password so that the fullselect in the view definition can retrieve decrypted data. Use the following statement:

```
SET ENCRYPTION PASSWORD = :hv_pass;
```
3. Select the desired data from the view by using the following statement:

```
SELECT EMPNO FROM CLR_EMP;
```

Using password hints with column-level encryption

DB2 can store encryption password hints to help with forgotten encryption passwords. Each password hint uses 32 bytes in the encrypted column.

For column-level encryption, the password hint is set with the SET ENCRYPTION PASSWORD statement. The GETHINT function returns the password hint.

Example: Use the following statement to set the password hint to the host variable hv_hint:

```
SET ENCRYPTION PASSWORD = :hv_pass WITH HINT = :hv_hint;
```

Example: Suppose that the EMPNO column in the EMP table contains encrypted data and that you submitted a password hint when you inserted the data. Suppose that you cannot remember the encryption password for the data. Use the following statement to return the password hint:

```
SELECT GETHINT (EMPNO) FROM EMP;
```

Defining value-level encryption

When you use *value-level encryption*, each value in a given column can be encrypted with a different password. You set the password for each value by using the ENCRYPT keyword with the password.

The following keywords are used with value-level encryption:

ENCRYPT

Indicates which data requires encryption. Also, encryption passwords, and optionally password hints, are indicated as part of the ENCRYPT keyword for value-level encryption.

Recommendation: Use host variables instead of literal values for all passwords and password hints. If statements contain literal values for passwords and password hints, the security of the encrypted data can be compromised in the DB2 catalog and in a trace report.

DECRYPT_BIT, DECRYPT_CHAR, DECRYPT_DB

Checks for the correct password and decrypts data when the data is selected.

Example: Suppose that a Web application collects user information about a customer. This information includes the customer name, which is stored in host variable `custname`; the credit card number, which is stored in a host variable `cardnum`; and the password for the card number value, which is stored in a host variable `userpswd`. The application uses the following statement to insert the customer information:

```
INSERT INTO CUSTOMER (CCN, NAME)
VALUES(ENCRYPT(:cardnum, :userpswd), :custname);
```

Before the application displays the credit card number for a customer, the customer must enter the password. The application retrieves the credit card number by using the following statement:

```
SELECT DECRYPT_CHAR(CCN, :userpswd) FROM CUSTOMER WHERE NAME = :custname;
```

Using password hints with value-level encryption

DB2 can store encryption password hints to help with forgotten encryption passwords. Each password hint uses 32 bytes in the encrypted column.

For value-level encryption, the password hint is set with the ENCRYPT keyword. The `GETHINT` function returns the password hint.

Recommendation: Use host variables instead of literal values for all passwords and password hints. If the statements contain literal values for passwords and password hints, the security of the encrypted data can be compromised in the DB2 catalog and in a trace report.

Example: Suppose that you want the application from the previous example to use a hint to help customers remember their passwords. The application stores the hint in the host variable `pswdhint`. For this example, assume the values 'Tahoe' for `userpswd` and 'Ski Holiday' for `pswdhint`. The application uses the following statement to insert the customer information:

```
INSERT INTO CUSTOMER (CCN, NAME)
VALUES(ENCRYPT(:cardnum, :userpswd, :pswdhint), :custname);
```

If the customer requests a hint about the password, the following query is used:

```
SELECT GETHINT(CCN) INTO :pswdhint FROM CUSTOMER WHERE NAME = :custname;
```

The value for `pswdhint` is set to 'Ski Holiday' and returned to the customer. Hopefully the customer can remember the password 'Tahoe' from this hint.

Encrypting non-character values

DB2 supports encryption for numeric and datetime data types indirectly through casting. If non-character data is cast as VARCHAR or CHAR, the data can be encrypted.

Example: Suppose that you need to encrypt timestamp data and retrieve it in decrypted format. Perform the following steps:

1. Create a table to store the encrypted values and set the column-level encryption password by using the following statements:

```
CREATE TABLE ETEMP (C1 VARCHAR(124) FOR BIT DATA);  
SET ENCRYPTION PASSWORD :hv_pass;
```

2. Cast, encrypt, and insert the timestamp data by using the following statement:

```
INSERT INTO ETEMP VALUES ENCRYPT(CHAR(CURRENT TIMESTAMP));
```

3. Recast, decrypt, and select the timestamp data by using the following statement:

```
SELECT TIMESTAMP(DECRYPT_CHAR(C1)) FROM ETEMP;
```

Using predicates for encrypted data

When data is encrypted, only = and <> predicates provide accurate results.

Predicates such as >, <, and LIKE will return inaccurate results for encrypted data.

Example: Suppose that the value 1234 is encrypted as H71G. Also suppose that the value 5678 is encrypted as BF62. If you use a <> predicate to compare these two values in encrypted format, you receive the same result as you will if you compare these two values in decrypted format:

```
Decrypted: 1234 <> 5678    True  
Encrypted: H71G <> BF62    True
```

In both case, they are not equal. However, if you use a < predicate to compare these values in encrypted format, you receive a different result than you will if you compare these two values in decrypted format:

```
Decrypted: 1234 < 5678     True  
Encrypted: H71G < BF62     False
```

To ensure that predicates such as >, <, and LIKE return accurate results, you must first decrypt the data.

Optimizing performance of encrypted data

Encryption, by its nature, degrades the performance of most SQL statements. Decryption requires extra processing, and encrypted data requires more space in DB2.

If a predicate requires decryption, the predicate is a stage 2 predicate, which can degrade performance. Encrypted data can also impact your database design, which can indirectly impact performance. To minimize performance degradation, use encryption only in cases that require encryption.

Recommendation: Encrypt only a few highly sensitive data elements, such credit card numbers and medical record numbers.

Some data values are poor candidates for encryption. For example, boolean values and other small value sets, such as the integers 1 through 10, are poor candidates

for encryption. Because few values are possible, these types of data can be easy to guess even when they are encrypted. In most cases, encryption is not a good security option for this type of data.

Data encryption and indexes: Creating indexes on encrypted data can improve performance in some cases. Exact matches and joins of encrypted data (if both tables use the same encryption key to encrypt the same data) can use the indexes that you create. Because encrypted data is binary data, range checking of encrypted data requires table space scans. Range checking requires all the row values for a column to be decrypted. Therefore, range checking should be avoided, or at least tuned appropriately.

Encryption performance scenario: The following scenario contains a series of examples that demonstrate how to improve performance while working with encrypted data.

Example: Suppose that you must store EMPNO in encrypted form in the EMP table and in the EMPPROJ table. To define tables and indexes for the encrypted data, use the following statements:

```
CREATE TABLE EMP (EMPNO VARCHAR(48) FOR BIT DATA, NAME VARCHAR(48));
CREATE TABLE EMPPROJ(EMPNO VARCHAR(48) FOR BIT DATA, PROJECTNAME VARCHAR(48));
CREATE INDEX IXEMPPRJ ON EMPPROJ(EMPNO);
```

Example: Next, suppose that one employee can work on multiple projects, and that you want to insert employee and project data into the table. To set the encryption password and insert data into the tables, use the following statements:

```
SET ENCRYPTION PASSWORD = :hv_pass;
SELECT INTO :hv_enc_val FROM FINAL TABLE
  (INSERT INTO EMP VALUES (ENCRYPT('A7513'),'Super Prog'));
INSERT INTO EMPPROJ VALUES (:hv_enc_val,'UDDI Project');
INSERT INTO EMPPROJ VALUES (:hv_enc_val,'DB2 UDB Version 10');
SELECT INTO :hv_enc_val FROM FINAL TABLE
  (INSERT INTO EMP VALUES (ENCRYPT('4NF18'),'Novice Prog'));
INSERT INTO EMPPROJ VALUES (:hv_enc_val,'UDDI Project');
```

You can improve the performance of INSERT statements by avoiding unnecessary repetition of encryption processing. Note how the host variable hv_enc_val is defined in the SELECT INTO statement and then used in subsequent INSERT statements. If you need to insert a large number of rows that contain the same encrypted value, you might find that the repetitive encryption processing degrades performance. However, you can dramatically improve performance by encrypting the data, storing the encrypted data in a host variable, and inserting the host variable.

Example: Next, suppose that you want to find the programmers who are working on the UDDI Project. Consider the following pair of SELECT statements:

- **Poor performance:** The following query shows how not to write the query for good performance:

```
SELECT A.NAME, DECRYPT_CHAR(A.EMPNO) FROM EMP A, EMPPROJECT B
  WHERE DECRYPT_CHAR(A.EMPNO) = DECRYPT_CHAR(B.EMPNO) AND
        B.PROJECT = 'UDDI Project';
```

Although the preceding query returns the correct results, it decrypts every EMPNO value in the EMP table and every EMPNO value in the EMPPROJ table where PROJECT = 'UDDI Project' to perform the join. For large tables, this unnecessary decryption is a significant performance problem.

- **Good performance:** The following query produces the same result as the preceding query, but with significantly better performance. To find the programmers who are working on the UDDI Project, use the following statement:

```
SELECT A.NAME, DECRYPT_CHAR(A.EMPNO) FROM EMP A, EMPPROJ B
WHERE A.EMPNO = B.EMPNO AND B.PROJECT = 'UDDI Project';
```

Example: Next, suppose that you want to find the projects that the programmer with employee ID A7513 is working on. Consider the following pair of SELECT statements:

- **Poor performance:** The following query requires DB2 to decrypt every EMPNO value in the EMPPROJ table to perform the join:

```
SELECT PROJECTNAME FROM EMPPROJ WHERE DECRYPT_CHAR(EMPNO) = 'A7513';
```

- **Good performance:** The following query encrypts the literal value in the predicate so that DB2 can compare it to encrypted values that are stored in the EMPNO column without decrypting the whole column. To find the projects that the programmer with employee ID A7513 is working on, use the following statement :

```
SELECT PROJECTNAME FROM EMPPROJ WHERE EMPNO = ENCRYPT('A7513');
```

Encrypting your data with Secure Socket Layer support

DB2 supports Secure Socket Layer (SSL) protocol because it uses the z/OS Communications Server IP Application Transparent Transport Layer service (AT-TLS).

The z/OS Communications Server for TCP/IP (beginning in V1R7 of z/OS) supports the AT-TLS function in the TCP/IP stack for applications that require secure TCP/IP connections. AT-TLS performs TLS on behalf of the application, such as DB2, by invoking the z/OS system SSL in the TCP layer of the TCP/IP stack. The z/OS system SSL supports TLS V1.0, SSL V3.0, and SSL V2.0 protocols.

AT-TLS also uses policies that provide system SSL configurations for connections that use AT-TLS. An application continues to send and receive clear text data over the socket while the transmission is protected by the system SSL.

AT-TLS support is policy-driven and can be deployed transparently underneath many existing sockets applications.

Related information

 [z/OS Communications Server: IP Configuration Guide](#)

AT-TLS configuration

A variety of configurations are required so that DB2 can take advantage of AT-TLS support.

You must complete the following configurations of your DB2 to utilize the AT-TLS support:

- PROFILE.TCPIP configuration

You can specify the TTLS or NOTTLS parameter on the TCPCONFIG statement in PROFILE.TCPIP to control whether you want to use the AT-TLS support.

- TCP/IP stack access control configuration

To protect TCP/IP connections, you can configure the RACF EZB.INITSTACK.sysname.tcpname resource in the SERVAUTH class to block all stack access except for the user IDs that are permitted to use the resource.

- Policy configuration

The policy agent provides AT-TLS policy rules to the TCP/IP stack. Each rule defines a set of security conditions that the policy agent compares to the conditions at the connection that it is checking. When the policy agent finds a match, it assigns the connection to the actions that are associated with the rule.

Configuring the DB2 server for SSL

To implement SSL support for a DB2 server, the TCP/IP SQL Listener service task of DDF must be capable of listening to a secondary secure port for inbound SSL connections.

The TCP/IP Listener accepts regular (non-SSL) connections on the DRDA port, whereas the secure port accepts only SSL connections to provide secure communications with a partner. Clients are assured of getting the SSL protocol connections that they require.

The secure port is used only for accepting secure connections that use the SSL protocol. When the TCP/IP Listener accepts an inbound connection on the secure port, DDF invokes SIOCTLSCTL IOCTL with `TTLSi_Req_Type=TTLS_QUERY_ONLY`; in this way, DDF retrieves policy status on the connection. If the IOCTL returns a policy status of `TTLS_POL_NO_POLICY`, a matching policy rule is not found for the connection.

If the IOCTL returns a policy status of `TTLS_POL_NOT_ENABLED`, a matching rule policy is found for the connection but a policy is not configured to allow a secure connection for that client.

If a secure port is not properly configured, DDF rejects the inbound connection requests on the secure port. You must change the client system to use the non-secure port, or you can configure the secure port. If the IOCTL returns a policy status of `TTLS_POL_ENABLED`, a matching policy rule is found, and SSL is enabled for the connection.

You can specify a secure port to DB2 in one of the following two ways:

- Specify the TCP/IP port number in the DRDA SECURE PORT field of the Distributed Data Facility Panel 2 (DSNTIP5) during DB2 installation.

The DRDA SECURE PORT field specifies the port number that is to be used for accepting TCP/IP connection requests from remote DRDA clients that want to establish a secure connection using the SSL protocol. The value of the port number is a decimal number between 1 and 65534, and it cannot have the same value as the values of the DRDA PORT and RESYNC PORT fields. Any non-zero port numbers are verified to ensure that they are all unique port numbers. If an error is detected, installation is not allowed to proceed until you resolve the error condition. If the DRDA SECURE PORT field is blank, SSL verification support is disabled, and the DB2 TCP/IP SQL Listener does not accept any inbound SSL connections on the secure port.

- Update the SECPORT parameter of the DDF statement in the BSDS with the change log inventory (DSNJU003) stand-alone utility.

The SECPORT parameter specifies the port number for the DB2 TCP/IP SQL Listener to accept inbound SSL connections. The value of the port number is a decimal number between 0 to 65535, and it cannot have the same value as the

values of the PORT and RESPORT parameters. If the value of SECPORT secure port is the same as the value of PORT or RESPORT, DB2 issues an error message. If you specify a value of 0 for the SECPORT parameter, SSL verification support is disabled, and the DB2 TCP/IP SQL Listener does not accept any inbound SSL connections on the secure port.

If the value of SECPORT is disabled, the client can still use the DRDA PORT and use SSL on it, but DB2 does not validate whether the connection uses SSL protocol.

Data sharing considerations: For a data sharing environment, each DB2 member with SSL support must specify a secure port. The secure port for each DB2 member of the group should be the same, just as the DRDA PORT for each member should also be the same. If each DB2 member specifies a unique secure port, unpredictable behaviors might occur. For example, Sysplex member workload balancing might not work correctly.

Similarly, for DB2 members that are defined as a subset of the data sharing group, each DB2 member that belongs to the subset needs to configure the secure port. You do not need to define a separate unique secure port for the location alias.

Configuring the DB2 requester for SSL

A DB2 requester must be able to insist on an SSL-protected connection to certain servers. To ensure SSL-protected connections, you can make communications database (CDB) changes that indicate that SSL-protected connections are required to certain remote locations.

If a secure connection is required, DDF must determine whether an AT-TLS policy rule is defined and whether AT-TLS is enabled for the connection. To obtain this AT-TLS information, DDF invokes SIOCTTLSCCTL IOCTL with TTLSi_Req_Type = TTLS_QUERY_ONLY. If the IOCTL returns a policy status of TTLS_POL_NO_POLICY, a matching policy rule is not found for the connection.

If the IOCTL returns a policy status of TTLS_POL_NOT_ENABLED, a policy rule is defined for the connection, but AT-TLS is not enabled, and a secure connection is not established with the remote server. DDF issues a message, and the connection is closed.

If the IOCTL returns a policy status of TTLS_POL_ENABLED, a matching policy rule is found, and SSL is enabled for the connection.

You can specify a secure connection to DB2 in either of the following ways:

- Specify 'Y' for the SECURE column in the SYSIBM.LOCATIONS table.
- Specify a desired value for the PORT column in the SYSIBM.LOCATIONS table for SSL connections.

For SSL support, the PORT column must contain the value of the configured secure DRDA port at the remote server. However, if the value of the PORT column is blank and the value of the SECURE column is 'Y', DB2 uses the reserved secure DRDA port (448) as the default.

Some DB2 applications might require SSL protection and accept the performance cost for this level of security. However, some applications might be satisfied with unprotected connections. This flexibility can be accomplished by the use of the LOCATION ALIAS name feature.

Consider a DB2 server that is configured to support both non-secure and secure connections. At the DB2 requester, you can define two rows in the SYSIBM.LOCATIONS table: one row that specifies the location name and the non-secure DRDA port of the server and another row that specifies a different location name and the secure DRDA port of the server and SECURE='Y'. At the DB2 server, you can define a LOCATION ALIAS name to provide alternative names for any DB2 requesters that need to access the server by using the SSL protocol.

Protecting data sets through RACF

To fully protect the data in DB2, you must take steps to ensure that no other process has access to the data sets in which DB2 data resides.

Use RACF, or a similar external security system, to control access to the data sets just as RACF controls access to the DB2 subsystem. This section explains how to create RACF profiles for data sets and allow their use through DB2.

Assume that the RACF groups DB2 and DB2USER, and the RACF user ID DB2OWNER, have been set up for DB2 IDs. Given that setting, the examples that follow show you how to:

- Add RACF groups to control data sets that use the default DB2 qualifiers.
- Create generic profiles for different types of DB2 data sets and permit their use by DB2 started tasks.
- Permit use of the profiles by specific IDs.
- Allow certain IDs to create data sets.

Adding groups to control DB2 data sets

The default high-level qualifier for data sets that contain DB2 databases and recovery logs is DSNC910; for distribution, target, SMP, and other installation data sets, the default high-level qualifier is DSN910.

The DB2OWNER user ID can create groups that control those data sets by issuing the following commands:

```
ADDGROUP DSNC910 SUPGROUP(DB2) OWNER(DB2OWNER)
ADDGROUP DSN910 SUPGROUP(DB2) OWNER(DB2OWNER)
```

Creating generic profiles for data sets

DB2 uses specific names to identify data sets for special purposes.

Suppose that SYSDSP is the RACF user ID for DB2 started tasks in the following examples. DB2OWNER can issue the following commands to create generic profiles for the data sets and give complete control over the data sets to DB2 started tasks:

- For active logs, issue the following commands:

```
ADDSD 'DSNC910.LOGCOPY*' UACC(NONE)
PERMIT 'DSNC910.LOGCOPY*' ID(SYSDSP) ACCESS(ALTER)
```
- For archive logs, issue the following commands:

```
ADDSD 'DSNC910.ARCHLOG*' UACC(NONE)
PERMIT 'DSNC910.ARCHLOG*' ID(SYSDSP) ACCESS(ALTER)
```
- For bootstrap data sets, issue the following commands:

```
ADDSD 'DSNC910.BSDS*' UACC(NONE)
PERMIT 'DSNC910.BSDS*' ID(SYSDSP) ACCESS(ALTER)
```

- For table spaces and index spaces, issue the following commands:

```
ADDSD 'DSNC910.DSNDBC.*' UACC(NONE)
PERMIT 'DSNC910.DSNDBC.*' ID(SYSDSP) ACCESS(ALTER)
```

- For installation libraries, issue the following command:

```
ADDSD 'DSN910.*' UACC(READ)
```

Started tasks do not need control.

- For other general data sets, issue the following commands:

```
ADDSD 'DSNC910.*' UACC(NONE)
PERMIT 'DSNC910.*' ID(SYSDSP) ACCESS(ALTER)
```

Although all of those commands are not absolutely necessary, the sample shows how you can create generic profiles for different types of data sets. Some parameters, such as universal access, could vary among the types. In the example, installation data sets (DSN910.*) are universally available for read access.

If you use generic profiles, specify NO on installation panel DSNTIPP for ARCHIVE LOG RACF, or you might get a z/OS error when DB2 tries to create the archive log data set. If you specify YES, DB2 asks RACF to create a separate profile for each archive log that is created, which means that you cannot use generic profiles for these data sets.

To protect VSAM data sets, use the cluster name. You do not need to protect the data component names, because the cluster name is used for RACF checking.

The VSAM resource that is used to store the administrative scheduler task list must be protected in RACF against unauthorized access. Only the administrative scheduler started task user has the UPDATE authority on VSAM resources.

Access by stand-alone DB2 utilities: The following DB2 utilities access objects that are outside of DB2 control:

- DSN1COPY and DSN1PRNT: table space and index space data sets
- DSN1LOGP: active logs, archive logs, and bootstrap data sets
- DSN1CHKR: DB2 directory and catalog table spaces
- Change Log Inventory (DSNJU003) and Print Log Map (DSNJU004): bootstrap data sets

The Change Log Inventory and Print Log Map utilities run as batch jobs that are protected by the USER and PASSWORD options on the JOB statement. To provide a value for the USER option, for example SVCAID, issue the following commands:

- For DSN1COPY:

```
PERMIT 'DSNC910.*' ID(SVCAID) ACCESS(CONTROL)
```

- For DSN1PRNT:

```
PERMIT 'DSNC910.*' ID(SVCAID) ACCESS(READ)
```

- For DSN1LOGP:

```
PERMIT 'DSNC910.LOGCOPY*' ID(SVCAID) ACCESS(READ)
PERMIT 'DSNC910.ARCHLOG*' ID(SVCAID) ACCESS(READ)
PERMIT 'DSNC910.BSDS*' ID(SVCAID) ACCESS(READ)
```

- For DSN1CHKR:

```
PERMIT 'DSNC910.DSNDBC.*' ID(SVCAID) ACCESS(READ)
```

- For Change Log Inventory:

```
PERMIT 'DSNC910.BSDS*' ID(SVCAID) ACCESS(CONTROL)
```

- For Print Log Map:

```
PERMIT 'DSNC910.BSDS*' ID(SVCAID) ACCESS(READ)
```

The level of access depends on the intended use, not on the type of data set (VSAM KSDS, VSAM linear, or sequential). For update operations, ACCESS(CONTROL) is required; for read-only operations, ACCESS(READ) is sufficient.

You can use RACF to permit programs, rather than user IDs, to access objects. When you use RACF in this manner, IDs that are not authorized to access the log data sets might be able to do so by running the DSN1LOGP utility. Permit access to database data sets through DSN1PRNT or DSN1COPY.

Authorizing DB2 IDs to use data set profiles

Authorization IDs with the installation SYSADM or installation SYSOPR authority need access to most DB2 data sets.

The following command adds the two default IDs that have the SYSADM and SYSOPR authorities if no other IDs are named when DB2 is installed:

```
ADDUSER (SYSADM SYSOPR)
```

The next two commands connect those IDs to the groups that control data sets, with the authority to create new RACF database profiles. The ID that has the installation SYSOPR authority (SYSOPR) does not need that authority for the installation data sets.

```
CONNECT (SYSADM SYSOPR) GROUP(DSNC910) AUTHORITY(CREATE) UACC(NONE)
CONNECT (SYSADM)          GROUP(DSN910)  AUTHORITY(CREATE) UACC(NONE)
```

The following set of commands gives the IDs complete control over DSNC910 data sets. The system administrator IDs also have complete control over the installation libraries. Additionally, you can give the system programmer IDs the same control.

```
PERMIT 'DSNC910.LOGCOPY*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC910.ARCHLOG*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC910.BSDS*'   ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC910.DSNDBC.*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC910.*'       ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSN910.*'        ID(SYSADM)      ACCESS(ALTER)
```

Enabling DB2 IDs to create data sets

You can issue the following command to connect several IDs to the DSNC910 group that has the CREATE authority:

```
CONNECT (USER1 USER2 USER3 USER4 USER5)
        GROUP(DSNC910) AUTHORITY(CREATE) UACC(NONE)
```

Those IDs can now explicitly create data sets whose names have DSNC910 as the high-level qualifier. Any such data sets that are created by DB2 or by these RACF user IDs are protected by RACF. Other RACF user IDs are prevented by RACF from creating such data sets.

If no option is supplied for PASSWORD on the ADDUSER command that adds those IDs, the first password for the new IDs is the name of the default group, DB2USER. The first time that the IDs sign on, they all use that password, but they must change the password during their first session.

Chapter 10. Auditing access to DB2

Security auditing allows you to inspect and examine the adequacy and effectiveness of the policies and procedures that you put in place to secure your data.

DB2 provides the ability for you to monitor if your security plan is adequately designed based on your security objectives and determine if your implementation techniques and procedures are effectively carried out to protect your data access and consistency. It enables you to address the following fundamental questions about your data security.

- What sensitive data requires authorized access?
- Who is privileged to access the data?
- Who has actually accessed the data?
- What attempts are made to gain unauthorized access?

The DB2 catalog contains critical authorization and authentication information. This information provides the primary audit trail for the DB2 subsystem. You can retrieve the information from the catalog tables by issuing SQL queries.

Most of the catalog tables describe the DB2 objects, such as tables, views, table spaces, packages, and plans. Other tables, particularly those with the “AUTH” character string in their names, hold records of every granted privilege and authority. Each catalog record of a grant contains the following information:

- Name of the object
- Type of privilege
- IDs that receive the privilege
- IDs that grant the privilege
- Time of the grant

The DB2 audit trace can help you monitor and track all the accesses to your protected data. The audit trace records provide another important trail for the DB2 subsystem. You can use the the audit trace to record the following access information:

- Changes in authorization IDs
- Changes to the structure of data, such as dropping a table
- Changes to data values, such as updating or inserting records
- Access attempts by unauthorized IDs
- Results of GRANT statements and REVOKE statements
- Mapping of Kerberos security tickets to IDs
- Other activities that are of interest to auditors

DB2 audit trace

The DB2 trace facility provides the ability to collect monitoring, auditing, and performance information of your data and environment.

The audit trace enables you to trace different events or categories of events by authorization IDs, object ownerships, and so on. When started, the audit trace records certain types of actions and sends the report to a named destination. The trace reports can indicate who has accessed data.

Authorization IDs traced by auditing

Audit traces generally identify a process by its primary authorization ID. The audit trace records the primary ID before and after invocation of an authorization exit routine. Therefore, you can identify the primary ID that is associated with a change.

GUPI **Exception:** If a primary ID has been translated many times, you might not be able to identify the primary ID that is associated with a change. Suppose that the server does not recognize the translated ID from the requesting site. In this case, you cannot use the primary ID to gather all audit records for a user that accesses remote data.

The AUTHCHG record shows the values of all secondary authorization IDs that are established by an exit routine.

With the audit trace, you can also determine which primary ID is responsible for the action of a secondary ID or a current SQL ID. Suppose that the user with primary ID SMITHJ sets the current SQL ID to TESTGRP to grant privileges over the table TESTGRP.TABLE01 to another user. The DB2 catalog records the grantor of the privileges as TESTGRP. However, the audit trace shows that SMITHJ issued the grant statement.

Recommendation: Consider carefully the consequences of altering that ID by using an exit routine because the trace identifies a process by its primary ID. If the primary ID identifies a unique user, individual accountability is possible. However, if several users share the same primary ID, you cannot tell which user issues a particular GRANT statement or runs a particular application plan. **GUPI**

Audit classes

When you start the trace, you choose the events to audit by specifying one or more audit classes.

GUPI

The trace records are limited to 5000 bytes; the descriptions that contain long SQL statements might be truncated. The following table describes the available classes and the events that they include.

Table 94. Audit classes and the events that they trace

Audit class	Events that are traced
1	Access attempts that DB2 denies because of inadequate authorization. This class is the default.
2	Explicit GRANT and REVOKE statements and their results. This class does not trace implicit grants and revokes.
3	CREATE, ALTER, and DROP statements that affect audited tables, and the results of these statements. This class traces the dropping of a table that is caused by DROP TABLESPACE or DROP DATABASE and the creation or alteration of a table with AUDIT CHANGES or AUDIT ALL.

Table 94. Audit classes and the events that they trace (continued)

Audit class	Events that are traced
4	Changes to audited tables. Only the first attempt to change a table, within a unit of recovery, is recorded. (If the agent or the transaction issues more than one COMMIT statement, the number of audit records increases accordingly.) The changed data is not recorded; only the attempt to make a change is recorded. If the change is not successful and is rolled back, the audit record remains; it is not deleted. This class includes access by the LOAD utility. Accesses to a dependent table that are caused by attempted deletions from a parent table are also audited. The audit record is written even if the delete rule is RESTRICT, which prevents the deletion from the parent table. The audit record is also written when the rule is CASCADE or SET NULL, which can result in deletions that cascade to the dependent table.
5	All read accesses to tables that are identified with the AUDIT ALL clause. As in class 4, only the first access within a DB2 unit of recovery is recorded. References to a parent table are also audited.
6	The bind of static and dynamic SQL statements of the following types: <ul style="list-style-type: none"> • INSERT, UPDATE, DELETE, CREATE VIEW, and LOCK TABLE statements for audited tables. Except for the values of host variables, the audit record contains the entire SQL statement. • SELECT statements on tables that are identified with the AUDIT ALL clause. Except for the values of host variables, the audit record contains the entire SQL statement.
7	Assignment or change of an authorization ID because of the following reasons: <ul style="list-style-type: none"> • Changes through an exit routine (default or user-written) • Changes through a SET CURRENT SQLID statement • An outbound or inbound authorization ID translation • An ID that is being mapped to a RACF ID from a Kerberos security ticket
8	The start of a utility job, and the end of each phase of the utility
9	Various types of records that are written to IFCID 0146 by the IFI WRITE function
10	CREATE and ALTER TRUSTED CONTEXT statements, establish trusted connection information and switch user information

GUIP

Limitations of the audit trace

The audit trace does not automatically record everything; it has the following limitations:

- **GUIP** The audit trace must be turned on; it is not on by default.
- The trace does not record old data after it is changed.
- If an agent or transaction accesses a table more than once in a single unit of recovery, the audit trace records only the first access.
- The audit trace does not record accesses if you do not start the audit trace for the appropriate class of events.
- Except class 8, the audit trace does not audit certain utilities. For example, the trace audits the first access of a table with the LOAD utility, but it does not

audit access by the COPY, RECOVER, and REPAIR utilities. The audit trace does not audit access by stand-alone utilities, such as DSN1CHKR and DSN1PRNT.

- The trace audits only the tables that you specifically choose to audit.
- You cannot audit access to auxiliary tables.
- You cannot audit the catalog tables because you cannot create or alter catalog tables.

This auditing coverage is consistent with the goal of providing a moderate volume of audit data with a low impact on performance. However, when you choose classes of events to audit, consider that you might ask for more data than you are willing to process. **GUPI**

Auditing in a distributed data environment

The DB2 audit trace records any access to your data, whether the request is from a remote location or from your local DB2.

GUPI The trace record for a remote request reports the authorization ID as the final result of one of the following conditions:

- An outbound translation
- An inbound translation
- Activity of an authorization exit routine

Essentially, the ID on a trace record for a remote request is the same as the ID to which you grant access privileges for your data. Requests from your location to a remote DB2 are audited only if an audit trace is active at the remote location. The output from the trace appears only in the records at that location. **GUPI**

Audit trace records

The audit trace records can contain the following information:

- **GUPI** The ID that initiated the activity
- The LOCATION of the ID that initiated the activity (if the access was initiated from a remote location)
- The type of activity and the time that the activity occurred
- The DB2 objects that were affected
- Whether access was denied
- The owner of a particular plan and package
- The database alias (DBALIAS) that was used to access a remote location or a location alias that was accepted from a remote application. **GUPI**

Audit trace reports

If you regularly start the audit trace for all classes, you can generate audit reports based on the data that you accumulate.

GUPI Consider producing audit trace reports that focus on the following important security events:

Use of sensitive data

You should define tables that contain sensitive data, such as employee

salary records, with the AUDIT ALL option. You can report use by table and by authorization ID to look for access by unusual IDs, at unusual times, or of unexpected types. You should also record any ALTER or DROP operations that affect the data. Use audit classes 3, 4, and 5.

Grants of critical privileges

Carefully monitor IDs with special authorities, such as SYSADM and DBADM. Also carefully monitor IDs with privileges over sensitive data, such as an update privilege on records of accounts payable. You can query the DB2 catalog to determine which IDs hold privileges and authorities at a particular time. To determine which IDs received privileges and then had them revoked, use audit class 2 and consult the audit records.

Unsuccessful access attempts

Investigate all unsuccessful access attempts. Although some access failures are only user errors, others can be attempts to violate security. If you have sensitive data, always use trace audit class 1. You can report by table or by authorization ID. 

Additional sources of audit information

In addition to the audit trace, DB2 offers the following sources of audit information for you to use:

GUPI

Additional DB2 traces

DB2 accounting, statistics, and performance traces are also available. You can also use DB2 Performance Expert to print reports of these traces.

Recovery log

Although the recovery log is not an all-purpose log, it can be useful for auditing. You can print information from the log by using the DSN1LOGP utility. For example, the summary report can show which table spaces have been updated within the range of the log that you scan. The REPORT utility can indicate what log information is available and where it is located.

Image copies of table spaces

Typical recovery procedures generate image copies of table spaces. You can inspect these copies, or use them with the RECOVER utility to recover a table space to a particular point in time. If you recover to a point in time, you narrow the time period during which a particular change could have been made.

z/OS console log

The z/OS console log contains messages about exceptional conditions that are encountered during DB2 operation. Inspect this log for symptoms of problems.

Audit trail

Whenever you install a new version, release, or maintenance of DB2, an automatic record provides an audit trail. The audit trail helps you determine whether the changes are appropriate and whether authorized personnel made them. The audit trail can also aid in investigation of application-related problems.

GUPI

Determining active security measures

If you are a security auditor, you must know the security measures that are enabled on the DB2 subsystem.

You can determine whether DB2 authorization checking, the audit trace, and data definition control are enabled in the following ways:

Audit trace

To see whether the trace is running, display the status of the trace by the command `DISPLAY TRACE(AUDIT)`.

DB2 authorization checking

Without changing anything, look at panel DSNTIPP. If the value of the `USE PROTECTION` field is YES, DB2 checks privileges and authorities before permitting any activity.

Data definition control

Data definition control is a security measure that provides additional constraints to existing authorization checks. With it, you control how specific plans or collections of packages can use data definition statements. To determine whether data definition control is active, look at option 1 on the DSNTIPZ installation panel.

Using DB2 audit trace and trace records

As with other types of DB2 traces, you can choose the following options for the audit trace:

- Categories of events
- Particular authorization IDs or plan IDs
- Methods to start and stop the audit trace
- Destinations for audit records

You can choose whether to audit the activity on a table by specifying an option of the `CREATE` and `ALTER` statements.

Starting the audit trace

You can automatically start an audit trace whenever DB2 is started.

GUIP

You can do so by setting the `AUDIT TRACE` field on the DSNTIPN installation panel to one of the following options:

- * (an asterisk) to provide a complete audit trace.
- NO, the default, if you do not want an audit trace to start automatically.
- YES to start a trace automatically for the default class (class 1: access denials) and the default destination (the SMF data set).
- A list of audit trace classes (for example, 1,3,5) to start a trace automatically for those classes. This option uses the default destination.

As with other types of DB2 traces, you can start an audit trace at any time by issuing the `START TRACE` command. You can choose the audit classes to trace and the destination for trace records. You can also include an identifying comment.

Example: The following command starts an audit trace for classes 4 and 6 with distributed activity:

```
-START TRACE (AUDIT) CLASS (4,6) DEST (GTF) LOCATION (*)  
  COMMENT ('Trace data changes; include text of dynamic DML statements.')
```

◀ **GUIP**

Stopping the audit trace

You can have multiple traces that run at the same time, including more than one audit trace. You can stop a particular trace by issuing the STOP TRACE command with the same options for START TRACE.

▶ **GUIP**

You must include enough options to uniquely identify a particular trace when you issue the command.

Example: The following command stops the trace that you started:

```
-STOP TRACE (AUDIT) CLASS (4,6) DEST (GTF)
```

If you did not save the START command, you can determine the trace number and stop the trace by its number. Use DISPLAY TRACE to find the number.

Example: DISPLAY TRACE (AUDIT) might return a message like the following output:

TNO	TYPE	CLASS	DEST	QUAL
01	AUDIT	01	SMF	NO
02	AUDIT	04,06	GTF	YES

The message indicates that two audit traces are active. Trace 1 traces events in class 1 and sends records to the SMF data set. Trace 1 can be a trace that starts automatically whenever DB2 starts. Trace 2 traces events in classes 4 and 6 and sends records to GTF.

You can stop either trace by using its identifying number (TNO).

Example: To stop trace 1, use the following command:

```
-STOP TRACE AUDIT TNO(1)
```

▶ **GUIP**

Auditing specific tables

You can use the AUDIT clause in the CREATE TABLE or ALTER TABLE statement to audit a specific table.

▶ **GUIP** **Example:** DB2 audits the department table whenever the audit trace is on if you create the table with the following statement:

```
CREATE TABLE DSN8910.DEPT  
  (DEPTNO CHAR(3) NOT NULL,  
   DEPTNAME VARCHAR(36) NOT NULL,  
   MGRNO CHAR(6) ,  
   ADMRDEPT CHAR(3) NOT NULL,
```

```

        LOCATION CHAR(16)
        PRIMARY KEY (DEPTNO)
    IN DSN8D91A.DSN8S91D
    AUDIT CHANGES;

```

Because this statement includes the AUDIT CHANGES option, DB2 audits the table for each access that inserts, updates, or deletes data (trace class 4).

Example: To also audit the table for read accesses (class 5), issue the following statement:

```

ALTER TABLE DSN8910.DEPT
    AUDIT ALL;

```

The statement is effective regardless of whether the table was previously chosen for auditing.

Example: To prevent all auditing of the table, issue the following statement:

```

ALTER TABLE DSN8910.DEPT
    AUDIT NONE;

```

For the CREATE TABLE statement, the default audit option is NONE. For the ALTER TABLE statement, no default option exists. If you do not use the AUDIT clause in an ALTER TABLE statement, the audit option for the table is unchanged.

When CREATE TABLE statements or ALTER TABLE statements affect the audit of a table, you can audit those statements. However, the results of those audits are in audit class 3, not in class 4 or class 5. Use audit class 3 to determine whether auditing was turned off for a table for an interval of time.

If an ALTER TABLE statement turns auditing on or off for a specific table, any plans and packages that use the table are invalidated and must be rebound. If you change the auditing status, the change does not affect plans, packages, or dynamic SQL statements that are currently running. The change is effective only for plans, packages, or dynamic SQL statements that begin running after the ALTER TABLE statement has completed. 

Auditing specific IDs or roles

As with other types of DB2 traces, you can start an audit trace for a particular plan name, a primary authorization ID, a role, or all of the above.

 You might consider having audit traces on at all times for IDs with the SYSADM authority because they have complete access to every table. If you have a network of DB2 subsystems, you might need to trace multiple authorization IDs if the primary authorization IDs are translated several times. For embedded SQL, the audited ID is the primary authorization ID of the plan or package owner. For dynamic SQL, the audited ID is the primary authorization ID.

You can also start an audit trace for a particular role in a trusted context by using the ROLE and XROLE filters. For example, you can issue the following command to write accounting records for threads with a ROLE = abc:

```
-start trace(acctg) dest(smfi) role(abc)
```

You can also issue the following command to write accounting records for threads with a ROLE= abc:

```
-start trace(acctg) dest(smfi) xrole(abc)
```

In addition, you can use the asterisk (*) wildcard character (as in "abc*") or the underscore (_) wildcard character (as in "a_c") for more flexibility in audit tracing.

GUIP

Determining ID privileges and authorities

As an auditor, you must be aware of the privileges and authorities that are associated with the IDs or roles in the DB2 subsystem.

You can use the following methods to determine the privileges and authorities that a specific ID or role holds:

- Query the DB2 catalog to determine which IDs or roles hold particular privileges.
- Check on individual IDs that are associated with group IDs or roles. Some authorization IDs that you encounter are probably group IDs, to which many individual IDs can be connected. To see which IDs are connected to a group, obtain a report from RACF or from whatever external security system you are using. These reports can tell you which IDs have the required privileges to use DB2 data sets and other resources.

Collecting audit trace records

You can prepare the System Management Facility (SMF) or Generalized Trace Facility (GTF) to accept audit trace records in the same way that you prepare performance or accounting trace records. The records are of SMF type 102, as are performance trace records.

GUIP If you send trace records to SMF (the default), data might be lost in the following circumstances:

- SMF fails while DB2 continues to run.
- An unexpected abend (such as a TSO interrupt) occurs while DB2 is transferring records to SMF.

In those circumstances, SMF records the number of records that are lost. z/OS provides an option to stop the system rather than to lose SMF data. **GUIP**

Formatting audit trace records

You can use any of the following methods to extract, format, and print the trace records:

- **GUIP** Use the DB2 Performance Expert.
- Write your own application program to access the SMF data.
- Use the instrumentation facility interface (IFI) as an online resource to retrieve audit records. **GUIP**

Managing data accuracy and consistency

DB2 provides many controls that you can apply to data entry and update.

Some of the controls are automatic; some are optional. All of the controls prohibit certain operations and provide error or warning messages if those operations are attempted. You can use these controls as a set auditing techniques to ensure data accuracy and consistency.

The set of techniques is not intended to be exhaustive. Other combinations of techniques are possible. For example, you can use table check constraints or a view with the check option to ensure that data values are members of a certain set. Or you can set up a master table and define referential constraints. You can also enforce the controls through application programs, and restrict the INSERT and UPDATE privileges only to those programs.

Ensuring data presence and uniqueness

You can define columns with the NOT NULL clause to ensure that the required data is present. You can also control the type of data by assigning column data types and lengths.

For example, you can specify that alphabetic data cannot be entered into a column with one of the numeric data types. You can also specify that the data for a DATE or TIME column must use a specific format.

You must ensure that the data in a column or a set of columns is unique. You can do so by creating a unique index on a column or set of columns.

Protecting data integrity

Triggers and table check constraints enhance the ability to control data integrity. Triggers are very useful for defining and enforcing rules that involve different states of DB2 data.

For example, a rule can prevent a salary column from more than a ten percent increase. A trigger can enforce this rule and provide the value of the salary before and after the increase for comparison.

Table check constraints designate the values that specific columns of a base table can contain. A check constraint can express simple constraints, such as a required pattern or a specific range, and rules that refer to other columns of the same table.

As an auditor, you can verify that the table definitions express the required constraints on column values as table check constraints. You can also create a view with the check option and insert or update values only through that view.

GUIP **Example:** Suppose that, in table T, data in column C1 must be a number between 10 and 20. Suppose also that data in column C2 is an alphanumeric code that must begin with A or B. Create view V1 with the following statement:

```
CREATE VIEW V1 AS
  SELECT * FROM T
    WHERE C1 BETWEEN 10 AND 20
      AND (C2 LIKE 'A%' OR C2 LIKE 'B%')
  WITH CHECK OPTION;
```

Because of the CHECK OPTION, view V1 allows only data that satisfies the WHERE clause. **GUIP**

You cannot use the LOAD utility with a view, but that restriction does not apply to user-written exit routines; you can consider using the following types of user-written routines:

Validation routines

You can use validation routines to validate data values. Validation routines

access an entire row of data, check the current plan name, and return a nonzero code to DB2 to indicate an invalid row.

Edit routines

Edit routines have the same access as validation routines, and can also change the row that is to be inserted. Auditors typically use edit routines to encrypt data and to substitute codes for lengthy fields. However, edit routines can also validate data and return nonzero codes.

Field procedures

Field procedures access data that is intended for a single column; they apply only to short-string columns. However, they accept input parameters, so generalized procedures are possible. A column that is defined with a field procedure can be compared only to another column that uses the same procedure.

Tracking data changes

Triggers offer an efficient means of maintaining an audit trail. You can define a trigger to activate in response to certain DELETE, INSERT, or UPDATE statements that change data.

You can qualify a trigger by providing a list of column names when you define the trigger. The qualified trigger is activated only when one of the named columns is changed. A trigger that performs validation for changes that are made in an UPDATE operation must access column values both before and after the update. Transition variables (available only to row triggers) contain the column values of the row change that activated the trigger. The old column values and the column values from after the triggering operation are both available.

Checking for lost and incomplete transactions

You can use the database balancing technique to alert you about lost and incomplete transactions. Database balancing determines, for each set of data, whether the opening balance, the control totals, and the processed transactions equal the closing balance and control totals.

DB2 has no automatic mechanism to calculate control totals and column balances and compare them with transaction counts and field totals. Therefore, to use database balancing, you must design these mechanisms into the application program.

Example: Use your application program to maintain a control table. The control table contains information to balance the control totals and field balances for update transactions against a user's view. The control table might contain these columns:

- View name
- Authorization ID
- Number of logical rows in the view (not the same as the number of physical rows in the table)
- Number of insert transactions and update transactions
- Opening balances
- Totals of insert transaction amounts and update transaction amounts
- Relevant audit trail information such as date, time, workstation ID, and job name

The program updates the transaction counts and amounts in the control table each time it completes an insert or update to the view. To maintain coordination during recovery, the program commits the work only after it updates the control table. After the application processes all transactions, the application writes a report that verifies the control total and balancing information.

Ensuring data consistency

When you control data entry, you perform only part of a complete security and auditing policy. You must also verify the results when data is accessed and changed and to make sure that your data is consistent.

Using referential integrity for data consistency

Referential integrity ensures that data is consistent across tables. When you define primary and foreign keys, DB2 automatically enforces referential integrity.

As a result, every value of a foreign key in a dependent table must be a value of a primary key in the appropriate parent table. However, DB2 does not enforce informational referential constraints across subsystems.

Recommendation: Use referential integrity to ensure that a column allows only specific values. Set up a master table of allowable values, and define its primary key. Define foreign keys in other tables that must have matching values in their columns. In most cases, you should use the SET NULL delete rule.

Using locks for data consistency

Locks can ensure that data remains consistent even when multiple users try to access the same data at the same time. From an auditing standpoint, you can use locks to ensure that only one user is privileged to change data at a given time. You can also ensure that no user is privileged to access uncommitted data.

If you use repeatable read (RR), read stability (RS), or cursor stability (CS) as your isolation level, DB2 automatically controls access to data by using locks. However, if you use uncommitted read (UR) as your isolation level, users can access uncommitted data and introduce inconsistent data. As an auditor, you must know the applications that use UR isolation and that can introduce inconsistent data or create security risks.

GUPI For static SQL, you can determine the plans and packages that use UR isolation by querying the catalog.

Example: For static SQL statements, use the following query to determine which plans use UR isolation:

```
SELECT DISTINCT Y.PLNAME
  FROM SYSIBM.SYSPLAN X, SYSIBM.SYSSTMT Y
  WHERE (X.NAME = Y.PLNAME AND X.ISOLATION = 'U')
        OR Y.ISOLATION = 'U'
  ORDER BY Y.PLNAME;
```

Example: For static SQL statements, use the following query to determine which packages use UR isolation:

```
SELECT DISTINCT Y.COLLOID, Y.NAME, Y.VERSION
  FROM SYSIBM.SYSPACKAGE X, SYSIBM.SYSPACKSTMT Y
  WHERE (X.LOCATION = Y.LOCATION AND
        X.LOCATION = ' ' AND
```

```

X.COLLID = Y.COLLID AND
X.NAME = Y.NAME AND
X.VERSION = Y.VERSION AND
X.ISOLATION = 'U')
OR Y.ISOLATION = 'U'
ORDER BY Y.COLLID, Y.NAME, Y.VERSION;

```

For dynamic SQL statements, turn on performance trace class 3 to determine which plans and packages use UR isolation. 

Consistency between systems: When an application program writes data to both DB2 and IMS, or to both DB2 and CICS, the subsystems prevent concurrent use of data until the program declares a point of consistency.

Checking data consistency

Whenever an operation changes the contents of a data page or an index page, DB2 verifies that the modifications do not produce inconsistent data.

Additionally, you can run the DSN1CHKR utility to verify the integrity of the DB2 catalog and the directory table spaces. You can also run this utility to scan the specified table space for broken links, damaged hash chains, or orphan entries.

Checking data consistency with SQL queries

If you suspect that a table contains inconsistent data, you can submit an SQL query to search for a specific type of error.

Example: Consider the view that is created by the following statement as an example:

```

CREATE VIEW V1 AS
SELECT * FROM T
WHERE C1 BETWEEN 10 AND 20
AND (C2 LIKE 'A%' OR C2 LIKE 'B%')
WITH CHECK OPTION;

```

The view allows an insert or update to table T1 only if the value in column C1 is between 10 and 20 and if the value in C2 begins with A or B. To check that the control has not been bypassed, issue the following statement:

```

SELECT * FROM T1
WHERE NOT (C1 BETWEEN 10 AND 20
AND (C2 LIKE 'A

```

If the control has not been bypassed, DB2 returns no rows and thereby confirms that the contents of the view are valid. You can also use SQL statements to get information from the DB2 catalog about referential constraints that exist.

Checking data consistency with the CHECK utilities

You can use the CHECK DATA, CHECK INDEX, and CHECK LOB online utilities to ensure data consistency.

CHECK DATA

The CHECK DATA utility checks referential constraints (but not informational referential constraints). It determines whether each foreign key value in each row is a value of the primary key in the appropriate parent table.

The CHECK DATA utility also checks table check constraints and checks the consistency between a base table space and any associated LOB or

XML table spaces. It determines whether each value in a row is within the range that was specified for that column when the table was created.

CHECK INDEX

The CHECK INDEX utility checks the consistency of indexes with the data to which the indexes point. It determines whether each index pointer points to a data row with the same value as the index key. If an index key points to a LOB, the CHECK INDEX utility determines whether the index key points to the correct LOB. If an index key points to an XML, the CHECK INDEX utility determines whether the index key points to the correct XML.

CHECK LOB

The CHECK LOB utility checks the consistency of a LOB table space. It determines whether any LOBs in the LOB table space are invalid.

Checking data consistency with the DISPLAY DATABASE command

If you allow a table to be loaded without enforcing referential constraints on its foreign key columns, the table might contain data that violates the constraints. DB2 places the table space that contains the table in the CHECK-pending status.

You can determine the table spaces with the CHECK-pending status by using the DISPLAY DATABASE command with the RESTRICT option. You can also use the DISPLAY DATABASE command to display table spaces with invalid LOBs.

Checking data consistency with the REPORT utility

You can use the REPORT utility with the TABLESPACESET keyword to determine and retrieve the following information:

- Table spaces that contain a set of tables interconnected by referential constraints
- LOB or XML table spaces that are associated with base tables
- Base table column and partition numbers that are associated with each LOB or XML table space.

Checking data consistency with the operation log

You can use the operation log to verify that DB2 is operated reliably and to reveal unauthorized operations and overrides. The operation log consists of an automated log of DB2 operator commands, such as those for starting and stopping the subsystem, and DB2 abends.

The operation log records the following information:

- Command or condition type
- Date and time when the command was issued
- Authorization ID that issued the command
- Database connection code

You can obtain this information from the system log (SYSLOG), the SMF data set, or the automated job scheduling system. To obtain the information, use SMF reporting, job-scheduler reporting, or a user-developed program. As a good practice, review the log report daily and keep a history file for comparison. Because abnormal DB2 termination can indicate integrity problems, implement an immediate notification procedure to alert the appropriate personnel (DBA, systems supervisor, and so on) of abnormal DB2 terminations.

Checking data consistency with internal integrity reports

You can generate internal integrity reports for application programs and utilities.

For application programs, you can record any DB2 return codes that indicate possible data integrity problems, such as inconsistency between index and table information, physical errors on database disk, and so on. All programs must check the SQLCODE or the SQLSTATE for the return code that is issued after an SQL statement is run. DB2 records, on SMF, the occurrence (but not the cause) of physical disk errors and application program abends. The program can retrieve and report this information; the system log (SYSLOG) and the DB2 job output also have this information. However, in some cases, only the program can provide enough detail to identify the exact nature of problem.

You can incorporate these integrity reports into application programs, or you can use them separately as part of an interface. The integrity report records the incident in a history file and writes a message to the operator's console, a database administrator's TSO terminal, or a dedicated printer for certain codes. The recorded information includes the following:

- Date
- Time
- Authorization ID
- Terminal ID or job name
- Application
- Affected view or affected table
- Error code
- Error description

When a DB2 utility reorganizes or reconstructs data in the database, it produces statistics to verify record counts and to report errors. The LOAD and REORG utilities produce data record counts and index counts to verify that no records were lost. In addition to that, keep a history log of any DB2 utility that updates data, particularly REPAIR. Regularly produce and review these reports, which you can obtain through SMF customized reporting or a user-developed program.

Part 3. Operation and recovery

Chapter 11. DB2 basic operational concepts

The simplest elements of operation for DB2 for z/OS include entering commands and understanding DB2 message identifiers. These topics provide information about entering DB2 commands and DB2 message identifiers.

Recommendations for entering commands

You can control most aspects of the operational environment by using DB2 commands.

GUPI You might need to use other types of commands, including:

- IMS commands that control IMS connections
- CICS commands that control CICS connections
- IMS and CICS commands that allow you to start and stop connections to DB2 and display activity on the connections
- z/OS commands that allow you to start, stop, and change the internal resource lock manager (IRLM)

GUPI

Related tasks

Chapter 15, “Monitoring and controlling DB2 and its connections,” on page 413

Related information

“Types of commands” (DB2 Command Reference)

DB2 operator commands

Commands are available to help you with all aspects of operating a DB2 for z/OS subsystem.

GUPI The DB2 commands, and their functions, are:

ALTER BUFFERPOOL

Sets or alters buffer pool size while DB2 is online.

ALTER GROUPBUFFERPOOL

Alters attributes of group buffer pools, which are used in a data sharing environment.

ALTER UTILITY

Alters the parameter values of an active REORG or REBUILD utility job.

ARCHIVE LOG

Archives (offloads) the current active log.

CANCEL THREAD

Cancels processing for specific local or distributed threads. This command can be used for parallel task threads.

DISPLAY ARCHIVE

Displays information about the specifications for archive parameters, status

of allocated dedicated tape units, volume and data set names that are associated with all active tape units, and correlation ID of the requester.

DISPLAY BUFFERPOOL

Displays buffer pool information while DB2 is online.

DISPLAY DATABASE

Displays the status of a database.

DISPLAY DDF

Displays information about the status and configuration of the distributed data facility (DDF) and about the connections or threads that DDF controls.

DISPLAY FUNCTION SPECIFIC

Displays the statistics about external user-defined functions that are accessed by DB2 applications.

DISPLAY GROUP

Displays information about the data sharing group to which a DB2 subsystem belongs.

DISPLAY GROUPBUFFERPOOL

Displays status and statistical information about DB2 group buffer pools, which are used in a data sharing environment.

DISPLAY LOCATION

Displays statistics about threads and conversations between the remote DB2 subsystem and the local subsystem.

DISPLAY LOG

Displays the current checkpoint frequency (CHKFREQ) value, information about the current active log data sets, and the status of the offload task.

DISPLAY PROCEDURE

Displays statistics about stored procedures that are accessed by DB2 applications.

DISPLAY RLIMIT

Displays the status of the resource limit facility (governor).

DISPLAY THREAD

Displays information about DB2, distributed subsystem connections, and parallel tasks.

DISPLAY TRACE

Displays the status of DB2 traces.

DISPLAY UTILITY

Displays the status of a utility.

MODIFY TRACE

Changes the trace events (IFCIDs) that are being traced for a specified active trace.

RECOVER BSDS

Re-establishes dual bootstrap data sets.

RECOVER INDOUBT

Recovers threads that are indoubt after DB2 is restarted.

RECOVER POSTPONED

Completes backout processing for units of recovery (URs) whose backout was postponed during an earlier restart, or cancels backout processing of the postponed URs if the CANCEL option is used.

RESET INDOUBT

Purges DB2 information about indoubt threads.

SET ARCHIVE

Controls or sets the limits for the allocation and the deallocation time of the tape units for archive log processing.

SET LOG

Modifies the checkpoint frequency (CHKFREQ) value dynamically without changing the value in the subsystem parameter load module.

SET SYSPARM

Loads the subsystem parameter module that is specified in the command.

START DATABASE

Starts a list of databases or table spaces and index spaces.

START DB2

Initializes the DB2 subsystem.

START DDF

Starts the distributed data facility.

START FUNCTION SPECIFIC

Activates an external function that is stopped.

START PROCEDURE

Starts a stored procedure that is stopped.

START RLIMIT

Starts the resource limit facility (governor).

START TRACE

Starts DB2 traces.

STOP DATABASE

Stops a list of databases or table spaces and index spaces.

STOP DB2

Stops the DB2 subsystem.

STOP DDF

Stops or suspends the distributed data facility.

STOP FUNCTION SPECIFIC

Prevents DB2 from accepting SQL statements with invocations of the specified functions.

STOP PROCEDURE

Prevents DB2 from accepting SQL CALL statements for a stored procedure.

STOP RLIMIT

Stops the resource limit facility (governor).

STOP TRACE

Stops traces.

TERM UTILITY

Terminates execution of a utility.



Where DB2 commands are entered

You can enter DB2 commands from different sources.

- “z/OS console or z/OS application program”
- “IMS terminal or program”
- “CICS terminal” on page 359
- “TSO terminal” on page 359
- “APF-authorized program” on page 360
- “IFI application program” on page 360

z/OS console or z/OS application program

You can enter all DB2 commands from a z/OS console or a z/OS application program. The START DB2 command must be issued from a z/OS console (or from an APF-authorized program, such as SDSF, that passes the START DB2 to the z/OS console). The command group authorization level must be SYS.

More than one DB2 subsystem can run under z/OS. You add a prefix to a DB2 command with special characters that identify which subsystem to direct the command to. The one- to eight-character prefix is called the *command prefix*. Specify the command prefix on installation panel DSNTIPM. The default character for the command prefix is -DSN1. Examples in this information use the hyphen (-) for the command prefix. For example, -START DB2.

IMS terminal or program

You can enter all DB2 commands except START DB2 from either an IMS terminal or program. The terminal or program must be authorized to enter the IMS /SSR command.

An IMS subsystem can attach to more than one DB2 subsystem, so you need to add a prefix. Commands that are directed from IMS to DB2 with a special character that identifies which subsystem to direct the command to. That character is called the *command recognition character* (CRC); specify it when you define DB2 to IMS, in the subsystem member entry in IMS.PROCLIB. (For details, see *DB2 Installation Guide*.)

Recommendation: Use the same character for the CRC and the command prefix for a single DB2 subsystem. You need to use a command prefix of one character; otherwise you cannot match these identifiers.

The examples in this information assume that both the command prefix and the CRC are the hyphen (-) . However, if you can attach to more than one DB2 subsystem, you must issue your commands using the appropriate CRC. In the following example, the CRC is a question mark character:

You enter:

```
/SSR ?DISPLAY THREAD
```

DB2 returns the following messages:

```
DFS058  SSR COMMAND COMPLETED
DSNV401I ? DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ? ACTIVE THREADS -
⋮
```

CICS terminal

You can enter all DB2 commands except START DB2 from a CICS terminal that is authorized to enter the DSN command code.

For example, you enter:

```
DSNC -DISPLAY THREAD
```

DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮
```

CICS can attach to only one DB2 subsystem at a time; therefore CICS does not use the DB2 command prefix. Instead, each command that is entered through the CICS attachment facility must be preceded by a hyphen (-), as in the previous example. The CICS attachment facility routes the commands to the connected DB2 subsystem and obtains the command responses.

TSO terminal

You can enter all DB2 commands except START DB2 from a DSN session.

Example: The TSO terminal displays:

```
READY
```

You enter:

```
DSN SYSTEM (subsystem-name)
```

The TSO terminal displays:

```
DSN
```

You enter:

```
-DISPLAY THREAD
```

DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
⋮
```

A TSO session can attach to only one DB2 subsystem at a time; therefore TSO does not use the DB2 command prefix. Instead, each command that is entered through the TSO attachment facility must be preceded by a hyphen (-), as the preceding example demonstrates. The TSO attachment facility routes the command to DB2 and obtains the command response.

All DB2 commands except START DB2 can also be entered from a DB2I panel using option 7, DB2 Commands.

APF-authorized program

As with IMS, DB2 commands (including START DB2) can be passed from an APF-authorized program to multiple DB2 subsystems by the MGCRC (SVC 34) z/OS service. Thus, the value of the command prefix identifies the particular subsystem to which the command is directed. The subsystem command prefix is specified, as in IMS, when DB2 is installed (in the SYS1.PARMLIB member IEFSSNxx). DB2 supports the z/OS WTO command and response token (CART) to route individual DB2 command response messages to the invoking application program. Use of the CART is necessary if multiple DB2 commands are issued from a single application program.

GUPI For example, to issue DISPLAY THREAD to the default DB2 subsystem from an APF-authorized program that runs as a batch job, use the following code:

```
MODESUPV DS    0H
          MODESET MODE=SUP,KEY=ZERO
SVC34    SR    0,0
          MGCRC  CMDPARM
          EJECT
CMDPARM  DS    0F
CMDFLG1  DC    X'00'
CMDLENG  DC    AL1(CMDEND-CMDPARM)
CMDFLG2  DC    X'0000'
CMDDATA  DC    C'-DISPLAY THREAD'
CMDEND   DS    0C
```

DB2 returns the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
:
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

GUPI

IFI application program

GUPI An application program can issue DB2 commands using the instrumentation facility interface (IFI). The IFI application program protocols are available through the IMS, CICS, TSO attachment facilities, the call attachment facility (CAF), and the Resource Recovery Services attachment facility (RRSAF). For an example in which the DB2 START TRACE command for monitor class 1 is issued, see *DB2 Performance Monitoring and Tuning Guide*. **GUPI**

Related tasks

Chapter 13, "Submitting work to DB2," on page 369

Where command responses go

In most cases, DB2 command responses are returned to the entering terminal or, for batch jobs, appear in the printed listing. In CICS, you can direct command responses to another terminal.

GUPI Name the other terminal as the destination (*dest*) in this command:

```
DSNC dest -START DATABASE
```

If a DB2 command is entered from an IMS or CICS terminal, the response messages can be directed to different terminals. If the response includes more than one message, the following cases are possible:

- If the messages are issued in a set, the entire set of messages is sent to the IMS or CICS terminal that entered the command. For example, DISPLAY THREAD issues a set of messages.
- If the messages are issued one after another, and not in a set, only the first message is sent to the terminal that entered the command. Subsequent messages are routed to one or more z/OS consoles using the WTO function. For example, START DATABASE issues several messages one after another.

You can choose alternative consoles to receive the subsequent messages by assigning them the routing codes that are placed in the DSNZPxxx module when DB2 is installed. If you want to have all of the messages available to the person who sent the command, route the output to a console near the IMS or CICS master terminal.

For APF-authorized programs that run in batch jobs, command responses are returned to the master console and to the system log if hardcopy logging is available. Hardcopy logging is controlled by the z/OS system command VARY. See

z/OS MVS System Commands for more information. 

Authorities for DB2 commands

The ability to issue DB2 commands, such as STOP DB2, and to use most other DB2 functions requires the appropriate privilege or authority. Privileges and authorities can be granted to authorization IDs in many combinations and can also be revoked.

 The individual authorities are listed in “Administrative authorities.” Each administrative authority has the individual authorities shown in its box, and the individual authorities for all the levels beneath it. For example, DBADM has ALTER, DELETE, INDEX, INSERT, SELECT, and UPDATE authorities, as well as those that are listed for DBCTRL and DBMAINT.

Any user with the STOPALL privilege can issue the STOP DB2 command. Besides those who have been granted STOPALL explicitly, the privilege belongs implicitly to anyone with SYSOPR authority or higher. When installing DB2 you can choose:

- One or two authorization IDs with installation SYSADM authority
- Zero, one, or two authorization IDs with installation SYSOPR authority

The IDs with those authorizations are contained in the load module for subsystem parameters (DSNZPxxx).

The START DB2 command can be entered only at a z/OS console that is authorized to enter z/OS system commands. The command group authorization level must be SYS.

DB2 commands that are entered from a logged-on z/OS console can be authorized by using secondary authorization IDs. The authorization ID that is associated with a z/OS console is SYSOPR, which carries the authority to issue all DB2 commands except:

- RECOVER BSDS
- START DATABASE
- STOP DATABASE

- ARCHIVE LOG

APF-authorized programs that issue commands through MGCRC (SVC 34) have SYSOPR authority unless DB2 can determine the RACF user ID of the program. In that case, DB2 uses that user ID for authorization. To avoid errors, the user should obtain SYSOPR authority for those DB2 subsystems.

The authority to start or stop any particular database must be specifically granted to an ID with SYSOPR authority. Likewise, an ID with SYSOPR authority must be granted specific authority to issue the RECOVER BSDS and ARCHIVE LOG commands.

The SQL GRANT statement can be used to grant SYSOPR authority to other user IDs such as the /SIGN user ID or the LTERM of the IMS master terminal.

For information about other DB2 authorization levels, see “Establishing RACF protection for DB2.” *DB2 Command Reference* also has authorization level information for specific commands. 

DB2 message identifiers

DB2 message identifiers have the form *DSNcxxx.t*.

 In a DB2 message identifier:

DSN Is the unique DB2 message prefix.

c Is a one-character code that identifies the DB2 subcomponent that issued the message. For example:

2	CICS attachment facility
M	IMS attachment facility
U	Utilities

xxx Is the message number.

t Is the message type, with these values and meanings:

A	Immediate action
D	Immediate decision
E	Eventual action
I	Information only

See *DB2 Messages* for an expanded description of message types.

A command prefix that identifies the DB2 subsystem precedes the message identifier, except in messages from the CICS and IMS attachment facilities. (The CICS attachment facility issues messages in the form *DSN2xxx.t*, and the IMS attachment facility issues messages in the form *DSNMxxx.t*.) CICS and IMS attachment facility messages identify the z/OS subsystem that generated the message.

The IMS attachment facility issues messages that are identified as *SSNMxxxx* and as *DFSTMxxx*. The *DFSxxx* messages are produced by IMS, under which the IMS attachment facility operates. 

Unsolicited DB2 messages

Unsolicited subsystem messages are sent to the z/OS console that issues the START DB2 command, or to consoles that are assigned the routing codes that were listed in the DSNZPxxx module during DB2 installation.

GUPI However, the following messages from the IMS and the CICS attachment facilities are exceptions to that rule:

- Specific IMS attachment facility messages are sent to the IMS master terminal.
- Unsolicited CICS messages are sent to the transient data entries that are specified for the MSGQUEUE(*name*) attribute in the RDO (resource definition online).
- CICS statistics messages that are issued because of shutdown are sent to the transient data entry that is specified in the RDO (STATSQUEUE).

Some DB2 messages that are sent to the z/OS console are marked as critical with the WTO descriptor code (11). This code signifies “critical eventual action requested” by DB2. Preceded by an at sign (@) or an asterisk (*), critical DB2 messages remain on the screen until they are specifically deleted. This prevents them from being missed by the operator, who is required to take a specific action.

GUPI

Operational control options

At an operator console or terminal, you can perform a variety of operational control activities, including issuing commands and receiving output.

GUPI The following table summarizes the operational control that is available at the operator console or terminal.

Table 95. Operational control summary

Type of operation	z/OS console	TSO terminal	IMS master terminal	Authorized CICS terminal
Issue DB2 commands and receive replies	Yes	Yes ¹	Yes ¹	Yes ¹
Receive DB2 unsolicited output	Yes	No	No	No
Issue IMS commands	Yes ²	No	Yes	No
Receive IMS attachment facility unsolicited output	No ³	No	Yes	No
Issue CICS commands	Yes ⁴	No	No	Yes
Receive CICS attachment facility unsolicited output	No ³	No	No	Yes ⁵

Table 95. Operational control summary (continued)

Type of operation	z/OS console	TSO terminal	IMS master terminal	Authorized CICS terminal
Notes:				
	1. This does not apply to START DB2. Commands that are issued from IMS must have the prefix /SSR. Commands that are issued from CICS must have the prefix DSNC.			
	2. This applies when using outstanding WTOR.			
	3. The "Attachment facility unsolicited output" does not include "DB2 unsolicited output."			
	4. Use the z/OS command MODIFY <i>jobname</i> CICS <i>command</i> . The z/OS console must already be defined as a CICS terminal.			
	5. Specify the output destination for the unsolicited output of the CICS attachment facility in the RDO.			



Chapter 12. Starting and stopping DB2

Starting and stopping DB2 is a simple process that you probably do not need to do often.

GUPI Before DB2 is stopped, the system takes a shutdown checkpoint. This checkpoint and the recovery log give DB2 the information it needs to restart.

This section describes the START DB2 and STOP DB2 commands, explains how you can limit access to data at startup, and contains a brief overview of startup after an abend. **GUPI**

Starting DB2

When it is installed, DB2 is defined as a formal z/OS subsystem.

GUPI Afterward, the following message appears during any IPL of z/OS:
DSN3100I - DSN3UR00 - SUBSYSTEM *ssnm* READY FOR -START COMMAND

where *ssnm* is the DB2 subsystem name. At that point, you can start DB2 from a z/OS console that is authorized to issue system control commands (z/OS command group SYS), by entering the command START DB2. The command must be entered from the authorized console and cannot be submitted through JES or TSO.

Starting DB2 by a JES batch job or a z/OS START command is impossible. The attempt is likely to start an address space for DB2 that then abends, probably with reason code X'00E8000F'.

You can also start DB2 from an APF-authorized program by passing a START DB2 command to the MGCRC (SVC 34) z/OS service. **GUPI**

Messages at start

DB2 issues a variety of messages when you start DB2. The specific messages vary based on the parameters that you specify.

GUPI At start time, DB2 issues some or all of the following messages.

```
$HASP373 xxxxMSTR STARTED
DSNZ002I - SUBSYS ssnm SYSTEM PARAMETERS
          LOAD MODULE NAME IS dsnzparm-name
DSNY001I - SUBSYSTEM STARTING
DSNJ127I - SYSTEM TIMESTAMP FOR BSDS=87.267 14:24:30.6
DSNJ001I - csect CURRENT COPY n ACTIVE LOG DATA
          SET IS DSNAME=...,
          STARTRBA=..., ENDRBA=...
DSNJ099I - LOG RECORDING TO COMMENCE WITH
          STARTRBA = xxxxxxxxxxxx
$HASP373 xxxxDBM1 STARTED
DSNR001I - RESTART INITIATED
DSNR003I - RESTART...PRIOR CHECKPOINT RBA=xxxxxxxxxxxxx
DSNR004I - RESTART...UR STATUS COUNTS...
          IN COMMIT=nnnn, INDOUBT=nnnn, INFLIGHT=nnnn,
```

```

        IN ABORT=nnnn, POSTPONED ABORT=nnnn
DSNR005I - RESTART...COUNTS AFTER FORWARD RECOVERY
        IN COMMIT=nnnn, INDOUBT=nnnn
DSNR006I - RESTART...COUNTS AFTER BACKWARD RECOVERY
        INFLIGHT=nnnn, IN ABORT=nnnn, POSTPONED ABORT=nnnn
DSNR002I - RESTART COMPLETED
DSN9002I - DSNYASCP 'START DB2' NORMAL COMPLETION
DSNV434I - DSNVRP NO POSTPONED ABORT THREADS FOUND
DSN9022I - DSNVRP 'RECOVER POSTPONED' NORMAL COMPLETION

```

If any of the *nnnn* values in message DSNR004I are not zero, message DSNR007I is issued to provide the restart status table. 

Options at start

Starting invokes the load module for subsystem parameters. This load module contains information that was specified when DB2 was installed.

 For example, the module contains the name of the IRLM to connect to. In addition, it indicates whether the distributed data facility (DDF) is available and, if it is, whether it should be automatically started when DB2 is started. You can specify PARM (*module-name*) on the START DB2 command to provide a parameter module other than the one that is specified at installation.

The START DB2 command starts the system services address space, the database services address space, and, depending on specifications in the load module for subsystem parameters (DSNZPARM by default), the distributed data facility address space. Optionally, another address space, for the internal resource lock manager (IRLM), can be started automatically.

A conditional restart operation is available, but no parameters indicate normal or conditional restart on the START DB2 command. 

Related concepts

“Conditional restart” on page 508

Related tasks

“Starting DDF” on page 461

Restricting access to data

You can restrict access to data with an option of the START DB2 command.

 To restrict access to data, on the START DB2 command, specify one of these options:

ACCESS(MAINT)

To limit access to users who have installation SYSADM or installation SYSOPR authority.

Users with those authorities can do maintenance operations such as recovering a database or taking image copies. To restore access to all users, stop DB2 and then restart it, either omitting the ACCESS keyword or specifying ACCESS(*).

ACCESS(*)

To allow all authorized users to connect to DB2.

Ending the wait state at start

If a JCL error, such as device allocation or region size, occurs while trying to start the database services address space, DB2 goes into wait status.

GUPI To end the wait, cancel the system services address space and the distributed data facility address space from the console. After DB2 stops, check the start procedures of all three DB2 address spaces for correct JCL syntax. See *Data Sharing: Planning and Administration* for more information.

To accomplish the check, compare the expanded JCL in the SYSOUT output with the correct JCL provided in *z/OS MVS JCL User's Guide* or *z/OS MVS JCL Reference*. Then, take the member name of the erroneous JCL procedure, which is also provided in the SYSOUT data set, to the system programmer who maintains your procedure libraries. After finding out which PROCLIB contains the JCL in question,

locate the procedure and correct it. **GUPI**

Restart options after an abend

Starting DB2 after it abends is different from starting it after the STOP DB2 command is issued.

GUPI After the STOP DB2 command, DB2 finishes its work in an orderly way and takes a shutdown checkpoint before stopping. When DB2 is restarted, it uses information from the system checkpoint and recovery log to determine the system status at shutdown.

When a power failure occurs, DB2 abends without being able to finish its work or take a shutdown checkpoint. When DB2 is restarted after an abend, it refreshes its knowledge of its status at termination by using information on the recovery log, DB2 then notifies the operator of the status of various units of recovery.

You can indicate that you want DB2 to postpone some of the backout work that is traditionally performed during system restart. You can delay the backout of long-running units of recovery by using installation options LIMIT BACKOUT and BACKOUT DURATION on panel DSNTIPL.

Normally, the restart process resolves all inconsistent states. In some cases, you have to take specific steps to resolve inconsistencies. There are steps you can take to prepare for those actions. For example, you can limit the list of table spaces that are recovered automatically when DB2 is started. **GUPI**

Related tasks

Chapter 17, "Restarting DB2 after termination," on page 501

Related reference

"Active log data set parameters: DSNTIPL" (DB2 Installation Guide)

Stopping DB2

Before DB2 stops, all DB2-related write to operator with reply (WTOR) messages must receive replies.

GUIP Then one of the following commands terminates the subsystem:

```
-STOP DB2 MODE(QUIESCE)
-STOP DB2 MODE(FORCE)
```

The following messages are returned:

```
DSNY002I - SUBSYSTEM STOPPING
DSN9022I - DSNYASCP '-STOP DB2' NORMAL COMPLETION
DSN3104I - DSN3EC00 - TERMINATION COMPLETE
```

Before restarting DB2, the following message must also be returned to the z/OS console that is authorized to enter the START DB2 command:

```
DSN3100I - DSN3EC00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

If the STOP DB2 command is not issued from a z/OS console, messages DSNY002I and DSN9022I are not sent to the IMS or CICS master terminal operator. They are routed only to the z/OS console that issued the START DB2 command.

For data sharing environment, see *Data Sharing: Planning and Administration*.

GUIP

Related concepts

“Normal termination” on page 501

Chapter 13. Submitting work to DB2

Application programs that run under TSO, IMS, or CICS can make use of DB2 resources by executing embedded SQL statements.

GUPI Application programs must meet certain conditions to embed SQL statements and to authorize the use of DB2 resources and data. These conditions vary based on the environment of the application program.

All application programming default values, including the subsystem name that the programming attachment facilities use, are in the DSNHDECP load module.

Make sure that your JCL specifies the proper set of program libraries. **GUPI**

Submitting work by using DB2I

Using the interactive program DB2I (DB2 Interactive), you can run application programs and perform many DB2 operations by entering values on panels. DB2I runs under TSO using ISPF (Interactive System Productivity Facility) services.

GUPI To submit work by using DB2I:

1. Log on to TSO by following your local procedures.
2. Enter ISPF.
3. Enter parameters to control operations. DB2 provides help panels to:
 - Explain how to use each operation.
 - Provide the syntax for and examples of DSN subcommands, DB2 operator commands, and DB2 utility control statements.

To access the help panels, press the HELP PF key. (The key can be set locally, but is typically PF1.) **GUPI**

Running TSO application programs

You use the DSN command and a variety of DSN subcommands to run TSO applications.

Requirement: A TSO application program that you run in a DSN session must be link-edited with the TSO language interface program (DSNELI). The program cannot include IMS DL/I calls because that requires the IMS language interface module (DFSLI000).

GUPI To run TSO application programs:

1. Log on to TSO.
2. Enter the DSN command.
3. Respond to the prompt by entering the RUN subcommand.

The terminal monitor program (TMP) attaches the DB2-supplied DSN command processor, which in turn attaches the application program.

The following example runs application program DSN8BC3. The program is in library *prefix.RUNLIB.LOAD*, which is the name that is assigned to the load module library.

```
DSN SYSTEM (subsystem-name)  
RUN PROGRAM (DSN8BC3) PLAN(DSN8BH91) LIB ('prefix.RUNLIB.LOAD')  
END
```

GUPI

DSN subcommands for TSO environments

The DSN command starts a DSN session, which in turn provides a variety of subcommands and other functions.

GUPI The DSN subcommands are:

ABEND

Causes the DSN session to terminate with a DB2 X'04E' abend completion code and with a DB2 abend reason code of X'00C50101'.

BIND PACKAGE

Generates an application package.

BIND PLAN

Generates an application plan.

DCLGEN

Produces SQL and host language declarations.

END Ends the DB2 connection and returns to TSO.

FREE PACKAGE

Deletes a specific version of a package.

FREE PLAN

Deletes an application plan.

REBIND PACKAGE

Regenerates an existing package.

REBIND PLAN

Regenerates an existing plan.

RUN Executes a user application program.

SPUFI Invokes a DB2I facility that executes SQL statements that are not embedded in an application program.

You can also issue the following DB2 and TSO commands from a DSN session:

- Any TSO command except TIME, TEST, FREE, or RUN.
- Any DB2 command except START DB2.

GUPI

Related reference

“DB2 operator commands” on page 355

Sources that DB2 uses to find authorization access by the application program

DB2 uses multiple sources to find an authorization for access by the application program.

GUPI DB2 checks the first source listed; if it is unavailable, it checks the second source, and so on.

1. RACF USER parameter supplied at logon
2. TSO logon user ID
3. Site-chosen default authorization ID
4. IBM-supplied default authorization ID

You can modify either the RACF USER parameter or the TSO user ID by a locally defined authorization exit routine. **GUPI**

Running IMS application programs

To run IMS application programs, you can either enter transactions from an IMS terminal, or you can invoke IMS transactions and commands by using the DB2-supplied stored procedures DSNAIMS or DSNAIMS2. Use DSNAIMS to send commands and single-segment transactions, and use DSNAIMS2 to send commands and multi-segment transactions.

Application programs that contain SQL statements run in the message processing program (MPP), the batch message processing (BMP), the Fast Path region, or the IMS batch region.

The program must be link-edited with the IMS language interface module (DFSLI000). It can write to and read from other database management systems using the distributed data facility, in addition to accessing DL/I and Fast Path resources.

DB2 checks whether the authorization ID that IMS provides is valid. For message-driven regions, IMS uses the SIGNON-ID or LTERM as the authorization ID. For non-message-driven regions and batch regions, IMS uses the ASXBUSER field (if RACF or another security package is active). The ASXBUSER field is defined by z/OS as seven characters. If the ASXBUSER field contains binary zeros or blanks (which indicates that RACF or another security package is not active), IMS uses the PSB name instead.

An IMS terminal operator probably notices few differences between application programs that access DB2 data and programs that access DL/I data because IMS sends no DB2-related messages to a terminal operator. However, your program can signal DB2 error conditions with a message of your choice. For example, at its first SQL statement, a program receives an SQL error code if the resources that are to run the program are not available or if the operator is not authorized to use the resources. The program can interpret the code and issue an appropriate message to the operator.

You can run batch DL/I jobs to access DB2 resources; DB2-DL/I batch support uses the IMS attachment facility.

Related reference

“DSNAIMS stored procedure” on page 832

Related information

IMS Application Programming: Design at ibm.com

“Loading and running a batch program” (DB2 Application Programming and SQL Guide)

Running CICS application programs

To run CICS applications, enter transactions from CICS terminals. You can also invoke CICS transactions by using the CICS transaction-invocation stored procedure.

For information about this stored procedure, see “DB2-supplied stored procedures.”

CICS transactions that issue SQL statements must be link-edited with the CICS attachment facility language interface module, DSNCLI, and the CICS command language interface module. CICS application programs can issue SQL, DL/I, or CICS commands. After CICS connects to DB2, any authorized CICS transaction can issue SQL requests that can write to and read from multiple DB2 instances using the distributed data facility. The application programs run as CICS applications.

DB2 checks an authorization ID that is related to the transaction against a plan that is assigned to it. The authorization ID for the transaction can be the operator ID, terminal ID, transaction ID, RACF-authenticated user ID, or another identifier that is explicitly provided by the RDO (resource definition online). See “Controlling access to a DB2 subsystem” for more information about DB2 authorization IDs.

Running batch application programs

Batch DB2 work can run in the TSO background under the TSO terminal monitor program (TMP) or in an IMS batch message processing (BMP) region. IMS batch regions can issue SQL statements.

For batch work that runs in the TSO background, the input stream can invoke TSO command processors, particularly the DSN command processor for DB2. This input stream can include DSN subcommands, such as RUN. An example of a TMP job follows:

```
//jobname JOB USER=SYSOPR ...
//GO EXEC PGM=IKJEFT01,DYNAMNBR=20
.
user DD statements
.
//SYSTSPRT DD SYSOUT=A
//SYSTSIN DD *
DSN SYSTEM (ssid)
.
subcommand (for example, RUN)
.
END
/*
```

In the example:

- IKJEFT01 identifies an entry point for TSO TMP invocation. Alternative entry points that are defined by TSO are also available to provide additional return

code and abend termination processing options. These options permit the user to select the actions to be taken by the TMP on completion of command or program execution.

Because invocation of the TSO TMP using the IKJEFT01 entry point might not be suitable for all user environments, refer to the TSO publications to determine which TMP entry point provides the termination processing options that are best suited to your batch execution environment.

- USER=SYSOPR identifies the user ID (SYSOPR in this case) for authorization checks.
- DYNAMNBR=20 indicates the maximum number of data sets (20 in this case) that can be dynamically allocated concurrently.
- z/OS checkpoint and restart facilities do not support the execution of SQL statements in batch programs that are invoked by the RUN subcommand. If batch programs stop because of errors, DB2 backs out any changes that were made since the last commit point.
- (*ssid*) is the subsystem name or group attachment name.

Related tasks

Chapter 19, “Backing up and recovering your data,” on page 531

Running application programs using CAF

The call attachment facility (CAF) allows you to customize and control your execution environments more extensively than the TSO, z/OS, or IMS attachment facilities. Programs that run in TSO foreground or TSO background can use either the DSN session or CAF. z/OS batch and started task programs can use only CAF.

GUPI IMS batch applications can also access DB2 databases through CAF, however, this method does not coordinate the commitment of work between the IMS and DB2 subsystems. Using the DB2 DL/I batch support for IMS batch applications is highly recommended.

To use CAF, you must first make available a load module known as the call attachment language interface or DSNALI. When the language interface is available, your program can use CAF to connect to DB2 in two ways:

- Implicitly, by including SQL statements or IFI calls in your program just as you would any program.
- Explicitly, by writing CALL DSNALI statements. **GUPI**

Related information

“Call attachment facility” (DB2 Application Programming and SQL Guide)

Running application programs using RRSF

The Resource Recovery Services attachment facility (RRSAF) is a DB2 attachment facility that relies on a z/OS component called Resource Recovery Services (z/OS RRS). z/OS RRS provides system-wide services for coordinating two-phase commit operations across z/OS subsystems.

GUPI Before you can run an RRSF application, z/OS RRS must be started. RRS runs in its own address space and can be started and stopped independently of DB2.

To use RRSAF, you must first make available a load module known as the RRSAF language interface or DSNRLI. When the language interface is available, your program can use RRSAF to connect to DB2 in two ways:

- Implicitly, by including SQL statements or IFI calls in your program just as you would any program.
- Explicitly, by using CALL DSNRLI statements to invoke RRSAF functions. Those functions establish a connection between DB2 and RRS and allocate DB2 resources.  **GUPI**

Related tasks

“Controlling RRS connections” on page 456

Related information

“Resource Recovery Services attachment facility” (DB2 Application Programming and SQL Guide)

Chapter 14. Scheduling administrative tasks

The administrative task scheduler runs tasks that are defined in a task list according to a requested schedule. Tasks can be stored procedures or JCL jobs.

You manage the scheduler task list through DB2 stored procedures that add and remove tasks. You can monitor the task list and the status of executed tasks through user-defined functions that are provided as part of DB2.

Tasks run according to a defined schedule, which can be based on an interval, a point in time, or an event. Activity can be further restricted by a limit on the number of invocations or by earliest and latest invocation dates.

Interacting with the administrative task scheduler

The administrative task scheduler is based on scheduled tasks. DB2 users can add, remove and list scheduled tasks that are executed at planned points in time by the scheduler.

At each point in time when the scheduler detects that a task should be executed, it drives the task execution according to the work described in the task definition. There is no user interaction. The scheduler delegates the execution of the task to one of its execution threads, which executes the stored procedure or the JCL job described in the work definition of the task. The execution thread waits for the end of the execution and notifies the administrative task scheduler. The scheduler stores the execution status of the task in its redundant task lists, in relation with the task itself.

Adding a task

You can use the stored procedure `ADMIN_TASK_ADD` to define new scheduled tasks. The parameters that you use when you call the stored procedure define the schedule and the work for each task.

The request and the parameters are transmitted to the administrative task scheduler associated with the DB2 subsystem where the stored procedure has been called. The parameters are checked and if they are valid, the task is added into the scheduler task lists with a unique task name. The task name and the return code are returned to the stored procedure for output.

At the same time, the scheduler analyzes the task to schedule its next execution.

Scheduling capabilities of the administrative task scheduler

The administrative task scheduler can execute a task once or many times, at fixed points in time, or in response to events.

Five parameters define the scheduling behavior of the task, in one of four ways:

- *interval*: elapsed time between regular executions
- *point-in-time*: specific times for execution
- *trigger-task-name* alone: specific task to trigger execution
- *trigger-task-name* with *trigger-task-cond* and *trigger-task-code*: specific task with required result to trigger execution

Only one of these definitions can be specified for any single task. The other parameters must be null.

Table 96. Relationship of null and non-null values for scheduling parameters

Parameter specified	Required null parameters
<i>interval</i>	<i>point-in-time</i> <i>trigger-task-name</i> <i>trigger-task-cond</i> <i>trigger-task-code</i>
<i>point-in-time</i>	<i>interval</i> <i>trigger-task-name</i> <i>trigger-task-cond</i> <i>trigger-task-code</i>
<i>trigger-task-name</i> alone	<i>interval</i> <i>point-in-time</i> <i>trigger-task-cond</i> <i>trigger-task-code</i>
<i>trigger-task-name</i> with <i>trigger-task-cond</i> and <i>trigger-task-code</i>	<i>interval</i> <i>point-in-time</i>

If *interval*, *point-in-time*, *trigger-task-name*, *trigger-task-cond*, and *trigger-task-code* are all null, *max-invocations* must be set to 1.

You can restrict scheduled executions either by defining a window of time during which execution is permitted or by specifying how many times a task can execute. Three parameters control restrictions:

- *begin-timestamp*: earliest permitted execution time
- *end-timestamp*: latest permitted execution time
- *max-invocations*: maximum number of executions

The *begin-timestamp* and *end-timestamp* parameters are timestamps that define a window of time during which tasks can start. Before and after this window, the task will not start even if the schedule parameters are met. If *begin-timestamp* is null, the window begins at the time when the task is added, and executions can start immediately. If *end-timestamp* is null, the window extends infinitely into the future, so that repetitive or triggered executions are not limited by time. Timestamps must either be null values or future times, and *end-timestamp* cannot be earlier than *begin-timestamp*.

For repetitive or triggered tasks, the number of executions can be limited using the *max-invocations* parameter. In this case, the task executes no more than the number of times indicated by the parameter, even if the schedule and the window of time would require the task to be executed. Executions that are skipped because they overlap with previous executions that are still running are not counted toward *max-invocations*.

The *max-invocations* parameter defines a limit but no requirement. If the task is executed fewer times than indicated during its execution window, the maximum number of executions will never be reached.

Defining task schedules

Use different combinations of parameters to define schedules for task executions.

The following ADMIN_TASK_ADD parameters provide control over when scheduled tasks execute:

- *interval*
- *point-in-time*
- *trigger-task-name*
- *trigger-task-cond*
- *trigger-task-code*
- *begin-timestamp*
- *end-timestamp*
- *max-invocations*

To define a new scheduled task, connect to the DB2 subsystem with sufficient authorization to call the stored procedure ADMIN_TASK_ADD. The following task definitions show some common scheduling options.

To define	Do this
A task that executes one time only:	<p>Set <i>max-invocations</i> to 1.</p> <p>Optionally, provide a value for the <i>begin-timestamp</i> parameter to control when execution happens. Leave other parameters null.</p> <p>For example, if <i>max-invocations</i> is set to 1 and <i>begin-timestamp</i> is set to 2008-05-27-06.30.0, the task executes at 6:30 AM on May 27, 2008.</p> <p>With this definition, the task executes one time. If <i>begin-timestamp</i> has been provided, execution happens as soon as permitted.</p>
A regular repetitive execution:	<p>Set <i>interval</i> to the number of minutes that you want to pass between the start of one execution and the start of the next execution.</p> <p>Optionally, provide values for the <i>max-invocations</i>, <i>begin-timestamp</i>, and <i>end-timestamp</i> parameters to limit execution. Leave other parameters null.</p> <p>For example, if <i>interval</i> is set to 5 and <i>begin-timestamp</i> is set to 2008-05-27-06.30.0, the task executes at 6:30 AM on May 27, 2008, then again at 6:35, 6:40, and so forth.</p> <p>With this definition, the task executes every <i>interval</i> minutes, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is postponed <i>interval</i> minutes. Execution continues to be postponed until the running task completes.</p>

To define	Do this
<p>An irregular repetitive execution:</p>	<p>Set <i>point-in-time</i> to a valid UNIX cron format string. The string specifies a set of times.</p> <p>Optionally, provide values for the <i>max-invocations</i>, <i>begin-timestamp</i> and <i>end-timestamp</i> parameters to limit execution. Leave other parameters null.</p> <p>For example, if <i>point-in-time</i> is set to 0 22 * * 1,5, the task executes at 10:00 PM each Monday and Friday.</p> <p>With this definition, the task executes at each time specified, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is skipped. Subsequent executions continue to be skipped until the running task completes.</p>
<p>An execution that is triggered when another task completes:</p>	<p>Set <i>trigger-task-name</i> to the name of the triggering task. Optionally set <i>trigger-task-cond</i> and <i>trigger-task-code</i> to limit execution based on the result of the triggering task. The <i>trigger-task-cond</i> and <i>trigger-task-code</i> parameters must either both be null or both be non-null.</p> <p>Optionally, provide values for the <i>max-invocations</i>, <i>begin-timestamp</i> and <i>end-timestamp</i> parameters to limit execution. Leave other parameters null.</p> <p>For example, assume that a scheduled INSERT job has a task name of test_task. If <i>trigger-task-name</i> is test_task, <i>trigger-task-cond</i> is EQ, and <i>trigger-task-code</i> is 0, then this task executes when the INSERT job completes with a return code of 0.</p> <p>With this definition, the task executes at each time specified, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is skipped. Subsequent executions continue to be skipped until the running task completes.</p>

To define	Do this
An execution that is triggered when DB2 starts:	<p>Set <i>trigger-task-name</i> to DB2START.</p> <p>Optionally, provide values for the <i>max-invocations</i>, <i>begin-timestamp</i> and <i>end-timestamp</i> parameters to limit execution. Leave other parameters null.</p> <p>For example, if <i>trigger-task-name</i> is DB2START, <i>begin-timestamp</i> is 2008-01-01-00.00.0, and <i>end-timestamp</i> is 2009-01-01-00.00.0, the task executes each time that DB2 starts during 2008.</p> <p>With this definition, the task executes at each DB2 start, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is skipped. Subsequent executions continue to be skipped until the running task completes.</p>
An execution that is triggered when DB2 stops:	<p>Set <i>trigger-task-name</i> to DB2STOP.</p> <p>Optionally, provide values for the <i>max-invocations</i>, <i>begin-timestamp</i> and <i>end-timestamp</i> parameters to limit execution. Leave other parameters null.</p> <p>With this definition, the task executes at each DB2 stop, so long as the previous execution has finished. If the previous execution is still in progress, the new execution is skipped. Subsequent executions continue to be skipped until the running task completes.</p>

Choosing an administrative task scheduler in a data sharing environment

In a data sharing group, tasks can be added, removed, or executed in any of the administrative task schedulers with the same result. Tasks are not localized to a scheduler: a task can be added by one scheduler, and then executed by any of the schedulers in the data sharing group.

To force a task to be executed on a particular scheduler, specify the associated DB2 subsystem ID in the *DB2-SSID* parameter when you schedule the task.

ADMIN_TASK_ADD

The SYSPROC.ADMIN_TASK_ADD stored procedure adds a task to the scheduler task list.

GUPI

Environment

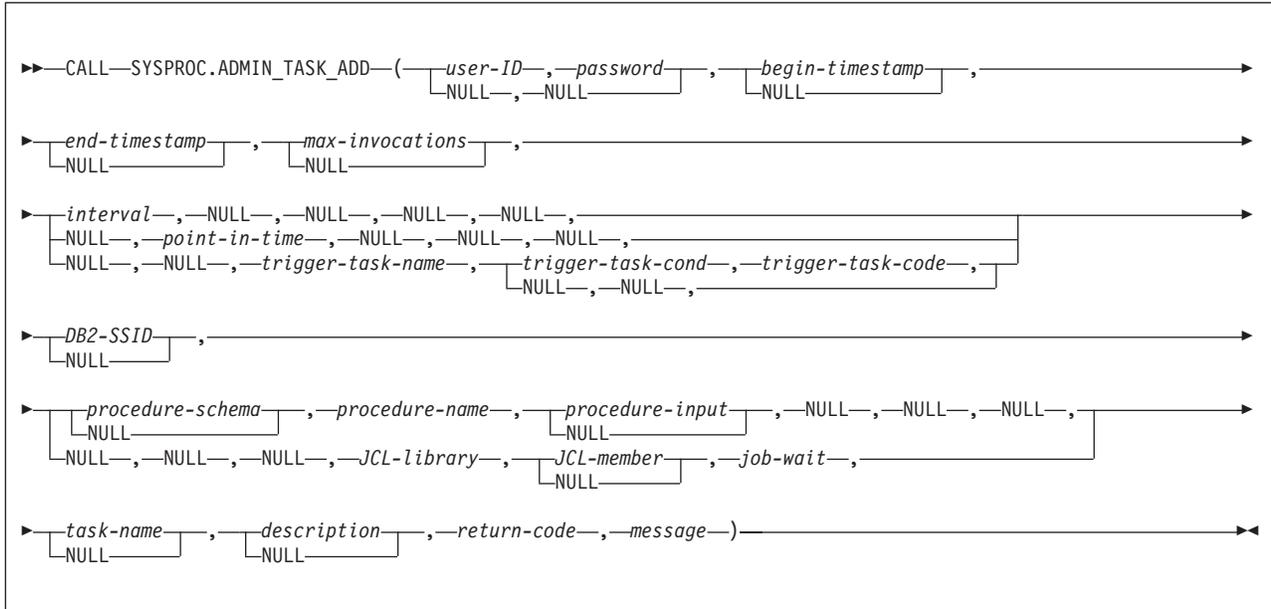
ADMIN_TASK_ADD runs in a WLM-established stored procedure address space and uses the Resource Recovery Services attachment facility to connect to DB2.

Authorization

Anyone who can execute this DB2 stored procedure is allowed to add a task.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:



Option descriptions

user-ID

Specifies the user ID under which the task execution is performed.

If this parameter is set to NULL, task execution is performed with the default authorization ID associated with the administrative task scheduler instead.

This is an input parameter of type VARCHAR(128).

password

Specifies the password associated with the input parameter *user-ID*.

The value of *password* is passed to the stored procedure as part of payload, and is not encrypted. It is not stored in dynamic cache when parameter markers are used.

Recommendation: Have the application that invokes this stored procedure pass an encrypted single-use password called a *passticket*.

This is an input parameter of type VARCHAR(24). This parameter is NULL only when *user-ID* is set to NULL, and must be NULL when *user-ID* is NULL.

begin-timestamp

Specifies when a task can first begin execution. When task execution begins depends on how this and other parameters are set:

Non-null value for *begin-timestamp*

| **At** *begin-timestamp*

| The task execution begins at *begin-timestamp* if *point-in-time* and
| *trigger-task-name* are NULL.

| **Next point in time defined at or after** *begin-timestamp*

| The task execution begins at the next point in time defined at
| or after *begin-timestamp* if *point-in-time* is non-null.

| **When** *trigger-task-name* **completes at or after** *begin-timestamp*

| The task execution begins the next time that *trigger-task-name*
| completes at or after *begin-timestamp*.

| **Null value for** *begin-timestamp*

| **Immediately**

| The task execution begins immediately if *point-in-time* and
| *trigger-task-name* are NULL.

| **Next point in time defined**

| The task execution begins at the next point in time defined if
| *point-in-time* is non-null.

| **When** *trigger-task-name* **completes**

| The task execution begins the next time that *trigger-task-name*
| completes.

| The value of this parameter cannot be in the past, and it cannot be later than
| *end-timestamp*.

| This is an input parameter of type `TIMESTAMP`.

| *end-timestamp*

| Specifies when a task can last begin execution. If this parameter is set to
| NULL, then the task can continue to execute as scheduled indefinitely.

| The value of this parameter cannot be in the past, and it cannot be earlier than
| *begin-timestamp*.

| This is an input parameter of type `TIMESTAMP`.

| *max-invocations*

| Specifies the maximum number of executions allowed for a task. This value
| applies to all schedules: triggered by events, recurring by time interval, and
| recurring by points in time. If this parameter is set to NULL, then there is no
| limit to the number of times this task can execute.

| For tasks that execute only one time, *max-invocations* must be set to 1 and
| *interval*, *point-in-time* and *trigger-task-name* must be NULL.

| If both *end-timestamp* and *max-invocations* are specified, the first limit reached
| takes precedence. That is, if *end-timestamp* is reached, even though the number
| of task executions so far has not reached *max-invocations*, the task will not be
| executed again. If *max-invocations* have occurred, the task will not be executed
| again even if *end-timestamp* is not reached.

| This is an input parameter of type `INTEGER`.

| *interval*

| Defines a time duration between two executions of a repetitive regular task.
| The first execution occurs at *begin-timestamp*. If this parameter is set to NULL,
| the task is not regularly executed. If this parameter contains a non-null value,
| the parameters *point-in-time* and *trigger-task-name* must be set to NULL.

| This is an input parameter of type `INTEGER`.

| *point-in-time*

| Defines one or more points in time when a task is executed. If this parameter
| is set to NULL, the task is not scheduled at fixed points in time. If this
| parameter contains a non-null value, the parameters *interval* and
| *trigger-task-name* must be set to NULL.

| The *point-in-time* string uses the UNIX cron format. The format contains the
| following pieces of information separated by blanks: given minute or minutes,
| given hour or hours, given day or days of the month, given month or months
| of the year, and given day or days of the week. For each part, you can specify
| one or several values, ranges, and so forth.

| This is an input parameter of type VARCHAR(400).

| *trigger-task-name*

| Specifies the name of the task which, when its execution is complete, will
| trigger the execution of this task.

| Task names of DB2START and DB2STOP are reserved for DB2 stop and start
| events respectively. Those events are handled by the scheduler associated with
| the DB2 subsystem that is starting or stopping.

| If this parameter is set to NULL, the execution of this task will not be triggered
| by another task. If this parameter contains a non-null value, the parameters
| *interval* and *point-in-time* must be set to NULL.

| This is an input parameter of type VARCHAR(128).

| *trigger-task-cond*

| Specifies the type of comparison to be made to the return code after the
| execution of task *trigger-task-name*. Possible values are:

- | GT Greater than
- | GE Greater than or equal to
- | EQ Equal to
- | LT Less than
- | LE Less than or equal to
- | NE Not equal to

| If this parameter is set to NULL, the task execution is triggered without
| considering the return code of task *trigger-task-name*. This parameter must be
| set to NULL if *trigger-task-name* is set to NULL or is either DB2START or
| DB2STOP.

| This is an input parameter of type CHAR(2).

| *trigger-task-code*

| Specifies the return code from executing *trigger-task-name*.

| If the execution of this task is triggered by a stored procedure, *trigger-task-code*
| contains the SQLCODE that must be returned by the triggering stored
| procedure in order for this task to execute.

| If the execution of this task is triggered by a JCL job, *trigger-task-code* contains
| the MAXRC that must be returned by the triggering job in order for this task
| to execute.

| To find out what the MAXRC or SQLCODE of a task is after execution, invoke
| the user-defined function DSNADM. ADMIN_TASK_STATUS returns these
| information in the columns MAXRC and SQLCODE.

The following restrictions apply to the value of *trigger-task-code*:

- If *trigger-task-cond* is null, then *trigger-task-code* must also be null.
- If *trigger-task-cond* is non-null, then *trigger-task-code* must also be non-null.

If *trigger-task-cond* and *trigger-task-code* are not null, they are used to test the return code from executing *trigger-task-name* to determine whether to execute this task or not.

For example, if *trigger-task-cond* is set to "GE" and *trigger-task-code* is set to "8", then this task will execute if and only if the previous execution of *trigger-task-name* returned a MAXRC (for a JCL job) or an SQLCODE (for a stored procedure) greater than or equal to 8.

This is an input parameter of type INTEGER.

DB2-SSID

Specifies the DB2 subsystem ID whose associated scheduler should execute the task.

This parameter is used in a data sharing environment where, for example different DB2 members have different configurations and executing the task relies on a certain environment. However, specifying a value in *DB2-SSID* will prevent schedulers of other members to execute the task, so that the task can only be executed as long as the scheduler of *DB2-SSID* is running.

For a task being triggered by a DB2 start or DB2 stop event in *trigger-task-name*, specifying a value in *DB2-SSID* will let the task be executed only when the named subsystem is starting and stopping. If no value is given, each member that starts or stops will trigger a local execution of the task, provided that the executions are serialized.

If this parameter is set to NULL, any scheduler can execute the task.

This is an input parameter of type VARCHAR(4).

procedure-schema

Specifies the schema of the DB2 stored procedure this task will execute. If this parameter is set to NULL, DB2 uses a default schema. This parameter must be set to NULL if *procedure-name* is set to NULL.

This is an input parameter of type VARCHAR(128).

procedure-name

Specifies the name of the DB2 stored procedure this task will execute. If this parameter is set to NULL, no stored procedure will be called. In this case, a JCL job must be specified.

This is an input parameter of type VARCHAR(128).

procedure-input

Specifies the input parameters of the DB2 stored procedure this task will execute. This parameter must contain a DB2 SELECT statement that returns one row of data. The returned values will be passed as parameter to the stored procedure.

If this parameter is set to NULL, no parameters are passed to the stored procedure. This parameter must be set to NULL when *procedure-name* is set to NULL.

This is an input parameter of type VARCHAR(4096).

JCL-library

Specifies the name of the data set where the JCL job to be executed is saved.

If this parameter is set to NULL, no JCL job will be executed. In this case, a stored procedure must be specified.

This is an input parameter of type VARCHAR(44).

JCL-member

Specifies the name of the library member where JCL job to be executed is saved.

If this parameter is set to NULL, the data set specified in *JCL-library* must be sequential and contain the JCL job to be executed. This parameter must be set to NULL if *JCL-library* is set to NULL.

This is an input parameter of type VARCHAR(8).

job-wait

Specifies whether the job can be executed synchronously or not. This parameter can only be set to NULL if *JCL-library* is set to NULL. Otherwise, it must be one of the following values:

NO Asynchronous execution

YES Synchronous execution

PURGE

Synchronous execution after which the job status in z/OS is purged

This is an input parameter of type VARCHAR(8).

task-name

Specifies a unique name assigned to the task.

A unique task name is returned when the task is created with a NULL *task-name* value. This name is of the format "TASK_ID_xxxx" where xxxx is 0001 for the first task named, 0002 for the second task, and so forth.

The following task names are reserved and cannot be given as the value of *task-name*:

- Names starting with "TASK_ID_"
- DB2START
- DB2STOP

This is an input-output parameter of type VARCHAR(128).

description

Specifies a description assigned to the task.

This is an input parameter of type VARCHAR(128).

return-code

Provides the return code from the stored procedure. Possible values are:

0 The call completed successfully.

12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. The first messages in this area, if any, are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Output

The output of this stored procedure is the task name, *task-name* and the following output parameters, which are described in “Option descriptions” on page 380:

- *return-code*
- *message*



UNIX cron format

The UNIX cron format is a way of specifying time for the *point-in-time* parameter of the ADMIN_TASK_ADD stored procedure.

The cron format has five time and date fields separated by at least one blank. There can be no blank within a field value. Scheduled tasks are executed when the minute, hour, and month of year fields match the current time and date, and at least one of the two day fields (day of month, or day of week) match the current date.

The allowed values for the time and date fields are:

Field Allowed values

minute

0-59

hour

0-23

day of month

1-31

month

- 1-12, where 1 is January
- Upper-, lower-, or mixed-case three-character strings, based on the English name of the month: jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, or dec.

day of week

- 0-7, where 0 or 7 is Sunday
- Upper-, lower-, or mixed-case three-character strings, based on the English name of the day: mon, tue, wed, thu, fri, sat, or sun.

Ranges and lists

Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive.

Example: The range 8-11 for an hour entry specifies execution at hours 8, 9, 10 and 11.

Lists are allowed. A list is a set of numbers or ranges separated by commas.

Examples:

1,2,5,9

0-4,8-12

Unrestricted range

A field can contain an asterisk (*), which represents all possible values in the field.

The day of a command's execution can be specified by two fields: day of month and day of week. If both fields are restricted by the use of a value other than the asterisk, the command will run when either field matches the current time.

Example: The value 30 4 1,15 * 5 causes a command to run at 4:30 AM on the 1st and 15th of each month, plus every Friday.

Step values

Step values can be used in conjunction with ranges. The syntax *range/step* defines the range and an execution interval.

If you specify *first-last/step*, execution takes place at *first*, then at all successive values that are distant from *first* by *step*, until *last*.

Example: To specify command execution every other hour, use 0-23/2. This expression is equivalent to the value 0,2,4,6,8,10,12,14,16,18,20,22.

If you specify **/step*, execution takes place at every interval of *step* through the unrestricted range.

Example: As an alternative to 0-23/2 for execution every other hour, use */2.

Listing scheduled tasks

You can use the function ADMIN_TASK_LIST to list tasks that are scheduled for execution by the administrative task scheduler.

To list scheduled tasks, connect to the DB2 subsystem with sufficient authorization to call the function ADMIN_TASK_LIST.

The function contacts the scheduler in order to update the DB2 task list in the table SYSIBM.ADMIN_TASKS if necessary, and then reads the tasks from the DB2 task list directly. The parameters that were used to create the task are column values of the returned table. The table also includes the authorization ID of the task creator, in the CREATOR column, and the time that the task was created, in the LAST_MODIFIED column.

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.



```
▶▶ ADMIN_TASK_LIST() ◀◀
```

The schema is DSNADM.

The result of the function is a table with the format shown in the following table. All the columns are nullable except TASK_NAME.

Table 97. Format of the resulting table for ADMIN_TASK_LIST

Column name	Data type	Contains
BEGIN_TIMESTAMP	TIMESTAMP	<p>Contains the timestamp of when the task can first run. When the task begins to run depends on what values this and other columns contain:</p> <ul style="list-style-type: none"> • If BEGIN_TIMESTAMP contains a non-null value: <ul style="list-style-type: none"> – If POINT_IN_TIME and TRIGGER_TASK_NAME contain null values, the task begins to run at the timestamp in BEGIN_TIMESTAMP – If POINT_IN_TIME contains a non-null value, the task begins to run at the next point in time that is defined at or after the timestamp in BEGIN_TIMESTAMP – If TRIGGER_TASK_NAME is a non-null value, the task begins to run at the next time that the task identified in TRIGGER_TASK_NAME completes or after the timestamp in BEGIN_TIMESTAMP • If BEGIN_TIMESTAMP contains a null value: <ul style="list-style-type: none"> – If POINT_IN_TIME and TRIGGER_TASK_NAME contain null values, the task begins to run immediately – If POINT_IN_TIME contains a non-null value, the task begins to run at the next point in time that is defined – If TRIGGER_TASK_NAME is a non-null value, the task begins to run at the next time that the task identified in TRIGGER_TASK_NAME completes
END_TIMESTAMP	TIMESTAMP	<p>Contains the timestamp of when the task is last able to run. If this column is NULL, there are no restrictions as to when the task must not run.</p>
MAX_INVOCATIONS	INTEGER	<p>Contains the maximum number of times the task can run. The maximum number applies to all types of schedules: triggered by events, scheduled by time interval, or by point in time. If this column is null, the task has no limit on the number of times it can be run.</p> <p>If both END_TIMESTAMP and MAX_INVOCATIONS contain values, the value in END_TIMESTAMP takes precedence over the value for MAX_INVOCATIONS. That is, if the value in END_TIMESTAMP is reached, even though the number of times the task has run has not reached the value for MAX_INVOCATIONS, the task will not run again</p>
INTERVAL	INTEGER	<p>Contains an integer that indicates the duration between the start of one instance of a task and the start of the next instance of the same task. If the value of this column is NULL, the task is not scheduled to run at a regular interval.</p>

Table 97. Format of the resulting table for ADMIN_TASK_LIST (continued)

Column name	Data type	Contains
POINT_IN_ TIME	VARCHAR(400)	<p>Contains one or more points in time (in UNIX cron format) for which the task is scheduled to run. If the value of this column is NULL, the task is not scheduled to run at a specific point in time.</p> <p>The format contains the following pieces of information separated by blanks: given hour, given minute, given day of the week, given day of the month, given month of the year.</p>
TRIGGER_ TASK_NAME	VARCHAR(128)	<p>Contains the task name of the task that, when its execution is complete, will trigger the running of the task that is described in the row.</p> <p>Task name DB2STOP is reserved for DB2 stop events and task name DB2START is reserved for DB2 start events. Those events are handled by the administrative scheduler that is associated with the DB2 subsystem that is starting or stopping.</p> <p>If the value of this column is NULL, the task that is described in this row will not be triggered to run by another task.</p>
TRIGGER_ TASK_COND	CHAR(2)	<p>Contains the type of comparison that is to be made to the return code after the running of task that is indicated in TRIGGER_TASK_NAME. The following values are possible:</p> <p>GT Greater than</p> <p>GE Greater than or equal to</p> <p>EQ Equal to</p> <p>LT Less than</p> <p>LE Less than or equal to</p> <p>NE Not equal to</p> <p>If this column contains NULL, the task is triggered to run without consideration of the return code of the task that is indicated in TRIGGER_TASK_NAME.</p>

Table 97. Format of the resulting table for ADMIN_TASK_LIST (continued)

Column name	Data type	Contains
TRIGGER_TASK_CODE	INTEGER	<p>Contains the return code from running the task indicated in TRIGGER_TASK_NAME.</p> <p>If the running of this task is triggered by a stored procedure, TRIGGER_TASK_CODE contains the SQLCODE that must be returned by the stored procedure in order for this task to run.</p> <p>If the running of this task is triggered by a JCL job, TRIGGER_TASK_CODE contains the MAXRC that must be returned by the job in order for this task to run.</p> <p>“ADMIN_TASK_STATUS” on page 392 returns the SQLCODE or MAXRC value in the SQLCODE or MAXRC column.</p> <p>If TRIGGER_TASK_COND is NULL, this column will also be NULL.</p>
DB2_SSID	VARCHAR(4)	<p>Contains the DB2 subsystem ID of the DB2 subsystem that is associated with the administrative scheduler that should run this task.</p> <p>The value in this column is used in a data sharing environment where, for example different DB2 members have different configurations and running the task relies on a certain environment. A value in DB2_SSID will prevent an administrative scheduler of other members to run this task, so that the task can only be run as long as the administrative scheduler of the subsystem indicated in DB2_SSID is running.</p> <p>For a task that is being triggered by a DB2 start or DB2 stop event as indicated in the TRIGGER_TASK_NAME column, a value in DB2_SSID will allow the task to be run only when the indicated subsystem is starting or stopping. If no value is indicated in DB2_SSID, each subsystem that starts or stops will trigger a the task to be run locally, provided that the triggered task is run serially.</p> <p>If this column is NULL, any administrative scheduler can run this task.</p>
PROCEDURE_SCHEMA	VARCHAR(128)	<p>Contains the schema of the DB2 stored procedure that this task will run. If the value of this column is null, DB2 uses a default schema.</p>
PROCEDURE_NAME	VARCHAR(128)	<p>Contains the name of the DB2 stored procedure that this task will run. If the value of this column is NULL, no stored procedure will be called when this task is run.</p>

Table 97. Format of the resulting table for ADMIN_TASK_LIST (continued)

Column name	Data type	Contains
PROCEDURE_INPUT	VARCHAR(4096)	Contains a statement that returns one row of data. The returned value will be used as the input parameter of the stored procedure that this task will run. If this column contains the null value, no parameters are passed to the stored procedure when this task is run.
JCL_LIBRARY	VARCHAR(44)	Contains the name of the data set that contains the JCL job that is run when this task is run. If the value of this column is the null value, no JCL job will be run when this task is run.
JCL_MEMBER	VARCHAR(8)	Contains the name of the library member that contains the JCL job that is run when this task is run. If the value of this column is the null value, the data set that is specified in JCL_LIBRARY is sequential and contains the JCL job that is run when this task is run.
JOB_WAIT	VARCHAR(8)	Contains one of the following values, which indicates whether the JCL job can be run synchronously. If the value in the column is not null, this column contains one of the following values: NO Runs asynchronously YES Runs synchronously PURGE Runs synchronously and then the job status in z/OS is purged
TASK_NAME	VARCHAR(128)	Contains the unique name that is assigned to this task.
DESCRIPTION	VARCHAR(128)	Contains a description of the task if one exists.
USERID	VARCHAR(128)	Contains the authorization ID of the user under which the task will be invoked. If this column is NULL, the task is invoked by the default authorization ID that is associated with the administrative scheduler.
CREATOR	VARCHAR(128)	Contains the authorization ID that added the task to the administrative scheduler task list.
LAST_MODIFIED	TIMESTAMP	Timestamp of when the task was added or last modified.

Example 1: Retrieve information about all of the tasks that are defined in the administrative scheduler task list:

```
SELECT *
FROM TABLE (DSNADM.ADMIN_TASK_LIST()) AS T;
```

Listing the last execution status of scheduled tasks

You can use the function ADMIN_TASK_STATUS to view the last execution status of scheduled tasks.

Before a task is first scheduled, all columns of its execution status contain null values, as returned by the table function ADMIN_TASK_STATUS. Afterwards, at

least the TASK_NAME, USERID, DB2_SSID, STATUS, NUM_INVOCATIONS and START_TIMESTAMP columns contain a non-null value. This information indicates when and under which user ID the task status last changed, as well as the number of times this task was already executed. The rest of the execution status can be interpreted according to the different values of the STATUS column.

The table function ADMIN_TASK_STATUS contacts the administrative task scheduler in order to update the DB2 task list in table SYSIBM.ADMIN_TASKS, if necessary, and then reads the tasks from this task list directly.

To determine the last execution status of a scheduled task:

1. Execute the table function ADMIN_TASK_STATUS to generate the status table.
2. Select the rows in the table that correspond to the task name.

Tip: You can relate the task execution status to the task definition by joining the output tables from ADMIN_TASK_LIST and ADMIN_TASK_STATUS on the TASK_NAME column.

The table created by ADMIN_TASK_STATUS indicates the last execution of scheduled tasks. Each row is indexed by the task name and contains the last execution status of the corresponding task.

If task execution has never been attempted, because the execution criteria have not been met, the STATUS column contains a null value.

If the scheduler was not able to start executing the task, the STATUS column contains NOTRUN. The START_TIMESTAMP and END_TIMESTAMP columns are the same, and the MSG column indicates why the task execution could not be started. All JCL job execution status columns are NULL, but the DB2 execution status columns contain values if the reason for the failure is related to DB2. (For example, a DB2 connection could not be established.)

If the scheduler started executing the task but the task has not yet completed, the STATUS column contains RUNNING. All other execution status columns contain null values.

If the task execution has completed, the STATUS column contains COMPLETED. The START_TIMESTAMP and END_TIMESTAMP columns contain the actual start and end times. The MSG column might contain informational or error messages. The DB2 and JCL columns are filled with values when they apply.

If the scheduler was stopped during the execution of a task, the status remains RUNNING until the scheduler is restarted. When the scheduler starts again, the status is changed to UNKNOWN, because the scheduler cannot determine if the task was completed.

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 that indicates the status of the task for the last time it was run.

▶▶—ADMIN_TASK_STATUS()—◀◀

The schema is DSNADM.

The result of the function is a table with the format shown in the following table.

Table 98. Format of the resulting table for ADMIN_TASK_STATUS

Column name	Data type	Contains
TASK_NAME	VARCHAR(128)	Contains the name of the task that has run, is running, or has been bypassed.
STATUS	VARCHAR(10)	Contains one of the following values that indicates task status: RUNNING The task is currently running COMPLETED The task has finished running. For asynchronous tasks (JCL jobs), this column contains COMPLETED whenever the job is submitted to be run. Otherwise, this column contains COMPLETED only after the task has finished running. NOTRUN The task was not run at the scheduled invocation time. The MSG column contains the error or warning message that indicates why the task was not run. UNKNOWN The scheduler shut down while the task was running. The scheduler is started again but cannot know the execution status of this interrupted task.
NUM_INVOCATIONS	INTEGER	Contains the number of times the administrative scheduler attempted to run the task, including the current time if the task is currently running. The values in this column does not indicate if the task was successfully run.
START_TIMESTAMP	TIMESTAMP	Contains the time when the task started running if the STATUS column contains COMPLETED, RUNNING, or UNKNOWN. Otherwise, this column contains the time that the task should have started to run but could not.
END_TIMESTAMP	TIMESTAMP	Contains the time when the task finished running.
JOB_ID	CHAR(8)	Contains the job ID that is assigned to the JCL job submitted by the administrative scheduler. This column contains NULL if the task is a stored procedure or if the STATUS column does not contain COMPLETED.

Table 98. Format of the resulting table for ADMIN_TASK_STATUS (continued)

Column name	Data type	Contains																		
MAXRC	INTEGER	<p>Contains the highest return code from submitting a JCL job. If the task is synchronous, the value in this column is changed to the return code that is returned when the job finishes running.</p> <p>This column is set to NULL if the task is a stored procedure, if the STATUS column does not contain COMPLETED, or if a synchronous task is finished and has run with JES3 in a z/OS 1.7 or earlier system.</p>																		
COMPLETION_ TYPE	INTEGER	<p>Contains one of the following values that indicates the completion type of the JCL job submitted by the administrative scheduler:</p> <table border="0"> <tr> <td>0</td> <td>No completion information</td> </tr> <tr> <td>1</td> <td>Job ended normally</td> </tr> <tr> <td>2</td> <td>Job ended by completion code</td> </tr> <tr> <td>3</td> <td>Job had a JCL error</td> </tr> <tr> <td>4</td> <td>Job was canceled</td> </tr> <tr> <td>5</td> <td>Job abended</td> </tr> <tr> <td>6</td> <td>Converter abended while processing the job</td> </tr> <tr> <td>7</td> <td>Job failed security checks</td> </tr> <tr> <td>8</td> <td>Job failed in end-of-memory</td> </tr> </table> <p>This column contains NULL if the task is a stored procedure, if the STATUS column does not contain COMPLETED, or if the JCL job is run with JES3 in a z/OS 1.7 or earlier system.</p>	0	No completion information	1	Job ended normally	2	Job ended by completion code	3	Job had a JCL error	4	Job was canceled	5	Job abended	6	Converter abended while processing the job	7	Job failed security checks	8	Job failed in end-of-memory
0	No completion information																			
1	Job ended normally																			
2	Job ended by completion code																			
3	Job had a JCL error																			
4	Job was canceled																			
5	Job abended																			
6	Converter abended while processing the job																			
7	Job failed security checks																			
8	Job failed in end-of-memory																			
SYSTEM_ ABENDCD	INTEGER	<p>Contains the system abend code returned by a failed JCL job that was submitted by the administrative scheduler.</p> <p>This column contains NULL if the task is a stored procedure, if the STATUS column does not contain COMPLETED, or if the JCL job is run with JES3 in a z/OS 1.7 or earlier system.</p>																		
USER_ABENDCD	INTEGER	<p>Contains the user abend code returned by a failed JCL job that was submitted by the administrative scheduler.</p> <p>This column contains NULL if the task is a stored procedure, if the STATUS column does not contain COMPLETED, or if the JCL job is run with JES3 in a z/OS 1.7 or earlier system.</p>																		
MSG	VARCHAR(128)	Contains the error or warning message from the last time the task was run.																		
SQLCODE	INTEGER	Contains the SQLCODE set by DB2 when a stored procedure was called by the administrative scheduler. This column contains NULL if the task is a JCL job or if the STATUS column does not contain COMPLETED.																		

Table 98. Format of the resulting table for ADMIN_TASK_STATUS (continued)

Column name	Data type	Contains
SQLSTATE	CHAR(5)	Contains the SQLSTATE set by DB2 when a stored procedure was called by the administrative scheduler. This column contains NULL if the task is a JCL job or if the STATUS column does not contain COMPLETED.
SQLERRP	VARCHAR(8)	Contains the SQLERRP set by DB2 when a stored procedure was called by the administrative scheduler. This column contains NULL if the task is a JCL job or if the STATUS column does not contain COMPLETED.
SQLERRMC	VARCHAR(70)	Contains the SQLERRMC set by DB2 when a stored procedure was called by the administrative scheduler. This column contains NULL if the task is a JCL job or if the STATUS column does not contain COMPLETED.
DB2_SSID	VARCHAR(4)	Contains the DB2 subsystem ID that is associated with the administrative scheduler that ran the task or should have run the task.
USERID	VARCHAR(128)	Contain the user ID that the task ran under.

Example 1: Retrieve status information about all of the tasks that have run in the administrative scheduler task list:

```
SELECT *
FROM TABLE (DSNADM.ADMIN_TASK_STATUS()) AS T;
```

Removing a scheduled task

You can remove a scheduled task from the task list using the stored procedure ADMIN_TASK_REMOVE.

Even if a task has finished all of its executions and will never be executed again, it remains in the task list until it is explicitly removed through a call to the stored procedure ADMIN_TASK_REMOVE.

Restrictions:

- Only the user who scheduled a task or a user with SYSOPR, SYSADM, or SYSCTRL authority can delete a task.
- A task cannot be removed while it is executing.
- A task that is the trigger for another task cannot be removed.

To remove a scheduled task:

1. Optional: Issue the following SQL statement to identify tasks that will never execute again:

```
SELECT T.TASK_NAME
FROM TABLE (DSNADM.ADMIN_TASK_LIST()) T,
TABLE (DSNADM.ADMIN_TASK_STATUS()) S
WHERE T.TASK_NAME = S.TASK_NAME AND
(S.NUM_INVOCATIONS = T.MAX_INVOCATIONS OR
T.END_TIMESTAMP < CURRENT_TIMESTAMP) AND
STATUS <> 'RUNNING'
```

2. Confirm the name of the task that you want to remove.

3. Call the stored procedure ADMIN_TASK_REMOVE. You must provide the task name as a parameter to the stored procedure. The scheduled task is removed from the task list and its last execution status is deleted. Listing the scheduled tasks and execution statuses no longer returns a row for this task. The task name is freed up for future reuse.

ADMIN_TASK_REMOVE

The SYSPROC.ADMIN_TASK_REMOVE stored procedure removes a task from the task list of the administrative scheduler. If the task is currently running, it continues to execute until completion, and the task is not removed from the administrative scheduler task list. If other tasks depend on the execution of the task to be removed, this task is not removed from the administrative scheduler task list.

GUIP

Environment

See the recommended environment in the installation job DSNTIJRA.

Authorization

Users with SYSOPR, SYSCTRL, or SYSADM authority can remove any task. Other users who have EXECUTE authority on this stored procedure can remove tasks that they added. Attempting to remove a task that was added by a different user returns an error in the output.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
▶▶—CALL—SYSPROC.ADMIN_TASK_REMOVE—(—task-name,—return-code,—message—)————▶▶
```

Option descriptions

task-name

Specifies the task name of the task to be removed from the administrative scheduler task list.

This is an input parameter of type VARCHAR(128) and cannot be null.

return-code

Provides the return code from the stored procedure. Possible values are:

0 The call completed successfully.

12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure.

The first messages in this area, if any, are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following Java sample shows how to invoke ADMIN_TASK_REMOVE:

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.Timestamp;
import java.sql.Types;

Connection con =
    DriverManager.getConnection("jdbc:db2://myserver:myport/mydatabase",
        "myuser", "mypassword");
CallableStatement callStmt =
    con.prepareCall("CALL SYSPROC.ADMIN_TASK_REMOVE(?, ?, ?)");
// provide the id of the task to be removed
callStmt.setString(1, "mytask");
// register output parameters for error management
callStmt.registerOutParameter(2, Types.INTEGER);
callStmt.registerOutParameter(3, Types.VARCHAR);
// execute the statement callStmt.execute();
// manage the return code
if ( callStmt.getInt(2) == 0 )
    {
        System.out.print("\nSuccessfully removed task "
            + callStmt.getString(1));
    }
else
    {
        System.out.print("\nError code and message are: "
            + callStmt.getInt(2) + "/"
            + callStmt.getString(3));
    }
}
```

Output

The output of this stored procedure includes the following output parameters, which are described in “Option descriptions” on page 396:

- *return-code*
- *message*

 GUIP

Manually starting the administrative task scheduler

The administrative task scheduler normally starts when DB2 starts, but you can start it manually if necessary.

Use one of the following commands to start the scheduler:

- To start a scheduler named *admtproc* from the operator’s console using the default tracing option, issue the MVS system command:

```
start admtproc
```
- To start a scheduler named *admtproc* from the operator’s console with tracing enabled, issue the MVS system command:

```
start admtproc,trace=on
```

- To start a scheduler named *admtproc* from the operator's console with tracing disabled, issue the MVS system command:

```
start admtproc,trace=off
```

When the administrative scheduler starts, message DSNA671I displays on the console.

Manually stopping the administrative task scheduler

You can stop the administrative task scheduler as part of problem determination or in preparation for maintenance.

To stop the scheduler, use one of the following commands.

- **Recommended method:** To stop a scheduler named *admtproc* from the operator's console, issue the MVS system command:

```
modify admtproc,appl=shutdown
```

- The scheduler stops accepting requests and will not start new task executions. It waits until the execution of all currently running tasks completes and then terminates.
- Message DSNA670I displays on the console.

- **Alternate method:** If the MODIFY command does not shut down the scheduler, issue the MVS system command:

```
stop admtproc
```

- Any task that was invoked by the scheduler and is currently executing is interrupted.
- Message DSNA670I displays on the console.

Interrupted tasks keep their status as RUNNING and are not rescheduled until the scheduler is started again. At startup, the status of the interrupted tasks is set to UNKNOWN and message DSNA690I is written into the status. Look for UNKNOWN in the results of the ADMIN_TASK_STATUS user-defined function. If UNKNOWN is present in the STATUS column of the output table, then you should check to see if the task has completed. If an interrupted task has not completed, you should terminate the work.

Synchronization between administrative task schedulers in a data sharing environment

The administrative task schedulers of a data sharing group synchronize themselves through their task lists.

When a task is added, the stored procedure ADMIN_TASK_ADD is called by a DB2 member. The scheduler associated with this DB2 member adds the task to the common task list. The task list is shared among all schedulers associated with the data sharing group members. The next time a scheduler accesses the list, it will detect the new task.

All schedulers of the data sharing group access this task list once per minute, to check for new tasks. The scheduler that adds a task does not have to check the list, and can execute the task immediately. Any other scheduler can execute a task only after finding it in the updated task list. Any scheduler can remove a task without waiting.

In order to remove a task, the stored procedure ADMIN_TASK_REMOVE is called by a DB2 member. The scheduler associated with this DB2 member removes the

task from the common task list. The next time a scheduler checks the list, within one minute after the task has been removed, it detects that the task has been deleted.

No scheduler can execute a task without first locking it in the task list. This locking prevents deleted tasks from being executed: the task is no longer present in the list, so it cannot be locked. Because it cannot be locked, it cannot be executed. The locking also prevents double executions: a task that one scheduler has in progress is already locked, so no other scheduler can lock the task.

Troubleshooting the administrative task scheduler

Error and informational messages from the administrative task scheduler might display on the console. Error messages typically indicate a problem with the configuration of the scheduler itself or an error accessing its resources. Informational messages identify important steps in the life cycle of the scheduler: starting, stopping, automatically recovering one of the task lists, and so forth.

Related information

DB2 Messages

DB2 Codes

Enabling tracing for administrative task scheduler problem determination

If you need to perform problem determination on the administrative task scheduler in response to a message, you can use a trace.

Use the MVS system command `MODIFY` to enable or disable tracing.

- To start a trace for a scheduler named *admtproc*, issue the following MVS system command:

```
modify admtproc,appl=trace=on
```

Substitute the name of your scheduler for *admtproc*.

- To stop a trace for a scheduler named *admtproc*, issue the following MVS system command:

```
modify admtproc,appl=trace=off
```

- To configure the system so that tracing starts automatically when the scheduler starts, modify the procedure parameter `TRACE` in the JCL job that starts the scheduler. This job has the name that was assigned when the scheduler was installed. The job was copied into one of the `PROCLIB` library during the installation. Specify `TRACE=ON`.

To disable tracing, change the parameter to `TRACE=OFF`.

Recovering the administrative task scheduler task list

Two redundant, active copies of the task list exist in case of media failure. Console message `DSNA679I` indicates that one or both of these copies is not accessible or is corrupted. If this happens, you can recover the task list.

One copy of the task list is a shared VSAM data set, by default `DSNC910.TASKLIST`, where `DSNC910` is the DB2 catalog prefix. The other copy is stored in the table `ADMIN_TASKS` in the `SYSIBM` schema. Include these redundant copies as part of your backup and recovery plan.

Tip: If DB2 is offline, message `DSNA679I` displays on the console. As soon as DB2 starts, the administrative task scheduler performs an autonomic recovery of the `ADMIN_TASKS` table using the contents of the VSAM task list. When the recovery

is complete, message DSN695I displays on the console to indicate that both task lists are again available. (By default, message DSN679I displays on the console once per minute when DB2 is offline. You can change the frequency of this message by modifying the ERRFREQ parameter either as part of the started task or with a console command.)

Use the following procedures to recover the task list if it is lost or damaged:

- To recover if the ADMIN_TASKS task list is corrupted:
 1. Create a new and operable version of the table.
 2. Grant SELECT, UPDATE, INSERT and DELETE privileges on the table to the administrative scheduler started task user.

As soon as the ADMIN_TASKS table is accessible again, the scheduler performs an autonomic recovery of the table using the content of the VSAM task list.

- To recover if the VSAM file is corrupted, create an empty version of the VSAM task list. As soon as the VSAM task list is accessible again, the scheduler performs an autonomic recovery using the content of the ADMIN_TASKS task list.
- If both task lists (the VSAM data set and the ADMIN_TASKS table) are corrupted and inaccessible, the scheduler is no longer operable. Messages DSN681I and DSN683I display on the console and the scheduler terminates. To recover from this situation:
 1. Create an empty version of the VSAM task list.
 2. Recover the table space DSNADMDB.DSNADMTS, where the ADMIN_TASKS table is located.
 3. Restart the administrative scheduler.

As soon as both task lists are accessible again, the scheduler performs an autonomic recovery of the VSAM task list using the content of the recovered ADMIN_TASKS table.

Problem executing a task

When the administrative task scheduler tries to execute a task without success, or when an error is detected by the scheduler at the end of a task execution, the error description is written into the last execution status.

Symptoms

A task was scheduled successfully, but the action did not complete or did not complete correctly.

Diagnosing the problem

Use the function ADMIN_TASK_STATUS to review the last execution status of a task and identify any messages or return codes that were passed back to the scheduler.

Important: The task status is overwritten as soon as the next execution of the task starts.

Resolving the problem

Correct the underlying problem and review the schedule. The task can now be executed successfully, but execution occurs only according to the schedule. Failed executions are not rescheduled. If the task is no longer scheduled, for example

because it had a defined number of executions, you must remove it and add it again, with adjusted criteria. If the task is still scheduled, you do not need to take any further action unless the failed execution is required. You cannot adjust a schedule, so if you do require the failed execution and all future executions, you must remove the scheduled task and re-create it.

Problem in user-defined table functions

An SQL code and a few characters of an SQL error message are returned in response to either ADMIN_TASK_LIST or ADMIN_TASK_STATUS.

Symptoms

An SQL code is returned. When SQLCODE is -443, the error message cannot be read directly, because only a few characters are available.

Diagnosing the problem

Problem diagnosis and resolution depends on the SQLCODE returned.

-443 The SQLCODE of -443 indicates that an error occurred in the function. Use the message number, which is at the beginning of the truncated return string, to diagnose the problem.

Any other value

Any other SQLCODE indicates that the error is not in the function or in the scheduler. Troubleshoot the task itself.

Problem in stored procedures

An SQL code and a few characters of an SQL error message are returned in response to either ADMIN_TASK_ADD or ADMIN_TASK_REMOVE.

Symptoms

An SQL code is returned.

Diagnosing the problem

An SQL code other than 0 indicates that DB2 encountered a problem calling the stored procedure.

An SQL code of 0 is accompanied by a return code and an error message in the output parameters RETURN_CODE and MSG. The return code is 0 if the scheduled task could be added or removed successfully. If the return code is 12, an error occurred adding or removing the task, and the returned error message describes the cause. The first eight characters of the error message contain the error message ID.

Resolving the problem

Errors can originate with the stored procedure itself or with the scheduler, in which case the error information is transmitted back to the stored procedure for output. Most error messages are clearly in one category or the other. For example, DSN650I *csect-name* CANNOT CONNECT TO ADMIN SCHEDULER *proc-name* indicates an error from the stored procedure. DSN652I *csect-name* THE USER *user-name* IS NOT ALLOWED TO ACCESS TASK *task-name* belongs to the administrative task scheduler, which is checking the parameters and authorization information passed to it.

Understanding the source of the error should be enough to correct the cause of the problem. Most problems are incorrect usage of the stored procedure or an invalid configuration.

Correct the underlying problem and resubmit the call to the stored procedure to add or remove the task.

Architecture of the administrative task scheduler

The administrative task scheduler is a started task that can be seen as an additional DB2 address space, even if in a separate process. It is accessed through an SQL API and stores the scheduled tasks in two redundant task lists.

The scheduler is part of DB2 for z/OS. When properly configured, it is available and operable with the first DB2 start. The scheduler starts as a task on the z/OS system during DB2 startup. The scheduler has its own address space, named after the started task name.

Each DB2 subsystem has its own distinct scheduler connected to it. DB2 is aware of the scheduler whose name is defined in the subsystem parameter ADMTPROC. The scheduler is aware of DB2 by the subsystem name that is defined in the DB2SSID parameter of the started task.

The scheduler has an SQL interface consisting of stored procedures (ADMIN_TASK_ADD and ADMIN_TASK_REMOVE) and user-defined table functions (ADMIN_TASK_LIST and ADMIN_TASK_STATUS) defined in DB2. This SQL interface allows you to remotely add or remove administrative tasks, and to list those tasks and their execution status.

The scheduler executes the tasks according to their defined schedules. The status of the last execution is stored in the task lists as well, and you can access it through the SQL interface.

The following figure shows the architecture of the administrative task scheduler.

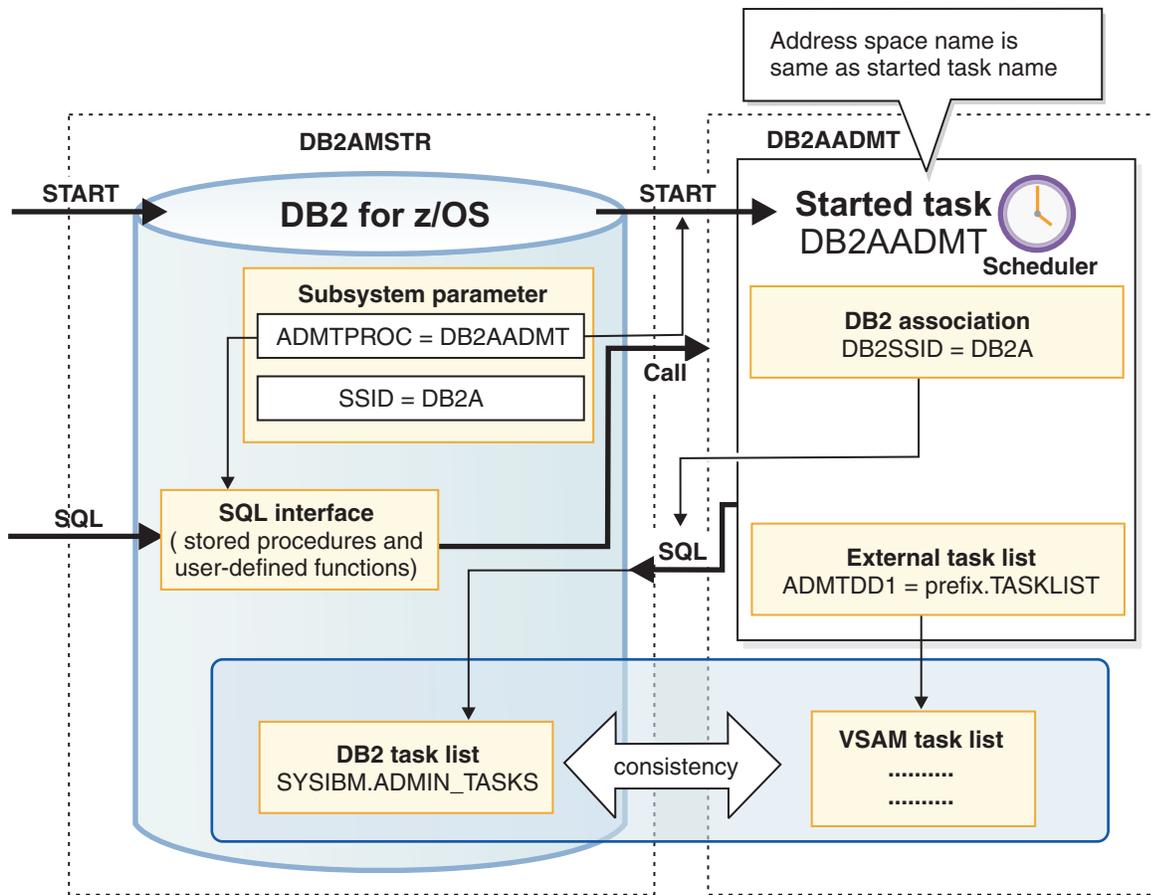


Figure 28. Architecture of the administrative task scheduler

Related reference

- “ADMIN_TASK_ADD” on page 379
- “ADMIN_TASK_REMOVE” on page 396
- “ADMIN_TASK_LIST” on page 387
- “ADMIN_TASK_STATUS” on page 392

The lifecycle of the administrative task scheduler

The administrative task scheduler starts as a task on the z/OS system during DB2 startup or initialization. It 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.

Two instances of the same scheduler cannot run simultaneously. To avoid starting up a duplicate scheduler, at startup the scheduler checks that there is no address space other than itself with the same name. If another address space with the same name is active, then the new scheduler immediately shut downs and displays a console message (DSNA674I). The scheduler can check the address spaces only in the same system, not the entire Sysplex.

When DB2 terminates, the administrative scheduler remains active so that scheduled JCL jobs can run. When DB2 starts again, it connects to the scheduler

automatically. It does not need to be restarted.

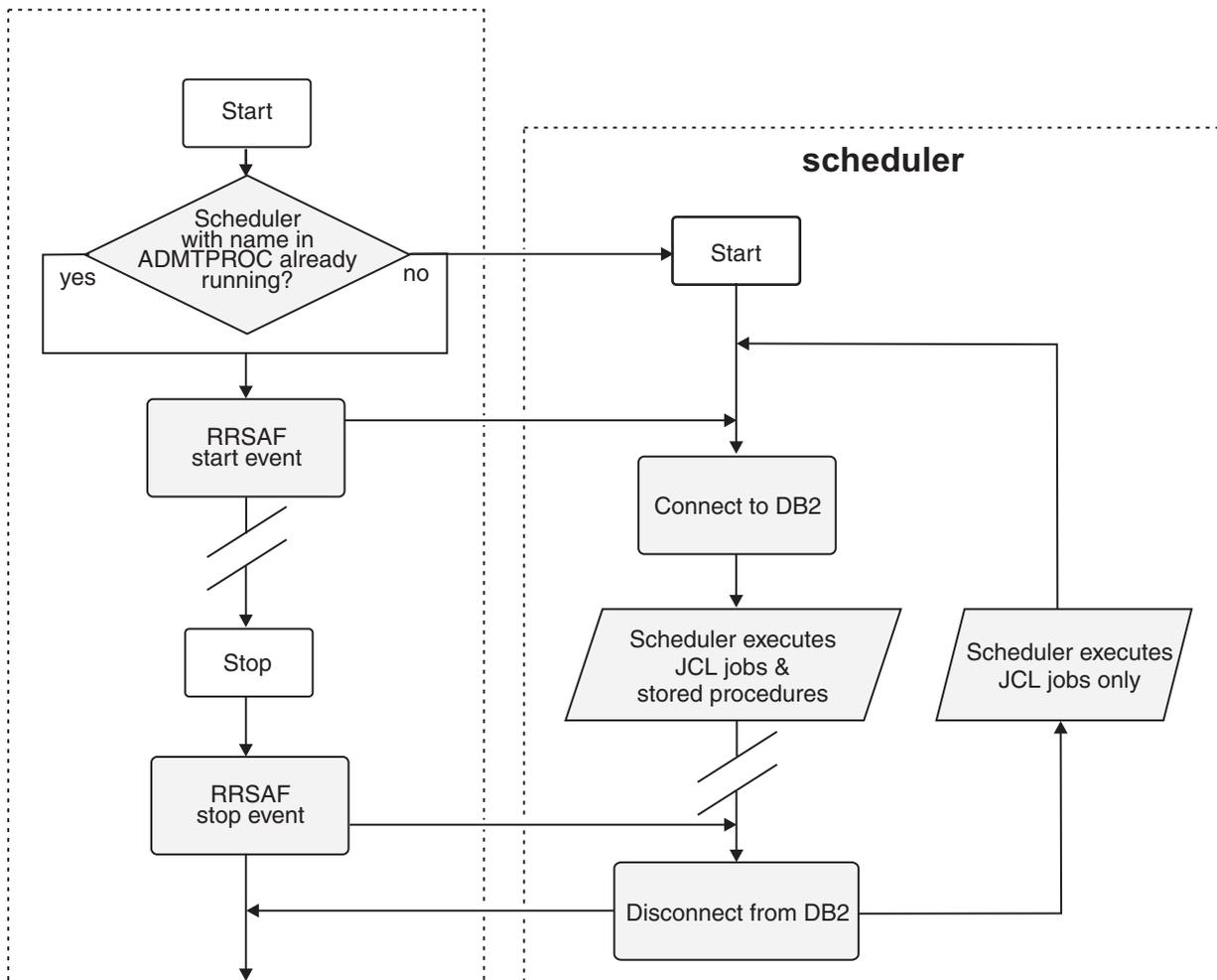


Figure 29. The lifecycle of the administrative task scheduler

Scheduler task lists

The administrative task scheduler manages tasks defined by users and stores them in two redundant task lists, so that the scheduler can continue working even if one task list is unavailable.

The two task lists are the DB2 table `SYSIBM.ADMIN_TASKS` and the VSAM data set that is indicated in the data definition `ADMTDD1` of the scheduler started task. The scheduler maintains the consistency between the two task lists.

The DB2 task list, `SYSIBM.ADMIN_TASKS`, is accessed through a connection to the DB2 subsystem identified in the `DB2SSID` parameter of the scheduler started task.

The scheduler works and updates both task lists redundantly, and remains operable so long as at least one of the task lists is available. In this way, the scheduler continues working when DB2 is offline. If a task list becomes unavailable, the scheduler continues to update the task list. When both task lists are available again, the scheduler automatically synchronizes them.

Architecture of the administrative task scheduler in a data sharing environment

In a data sharing environment, there is one administrative task scheduler for each DB2 for z/OS Version 9.1 or later member of a data sharing group, even when those members run in the same z/OS system. The schedulers share their resources and interface.

The scheduler task list is shared by all schedulers in a data sharing group, accessing a shared task file on shared storage (VSAM data set defaulting to DSNC910.TASKLIST, where DSNC910 is the DB2 catalog prefix) and a redundant task list in the DB2 system table SYSIBM.ADMIN_TASKS.

The following figure shows a data sharing group with two DB2 members and their associated schedulers.

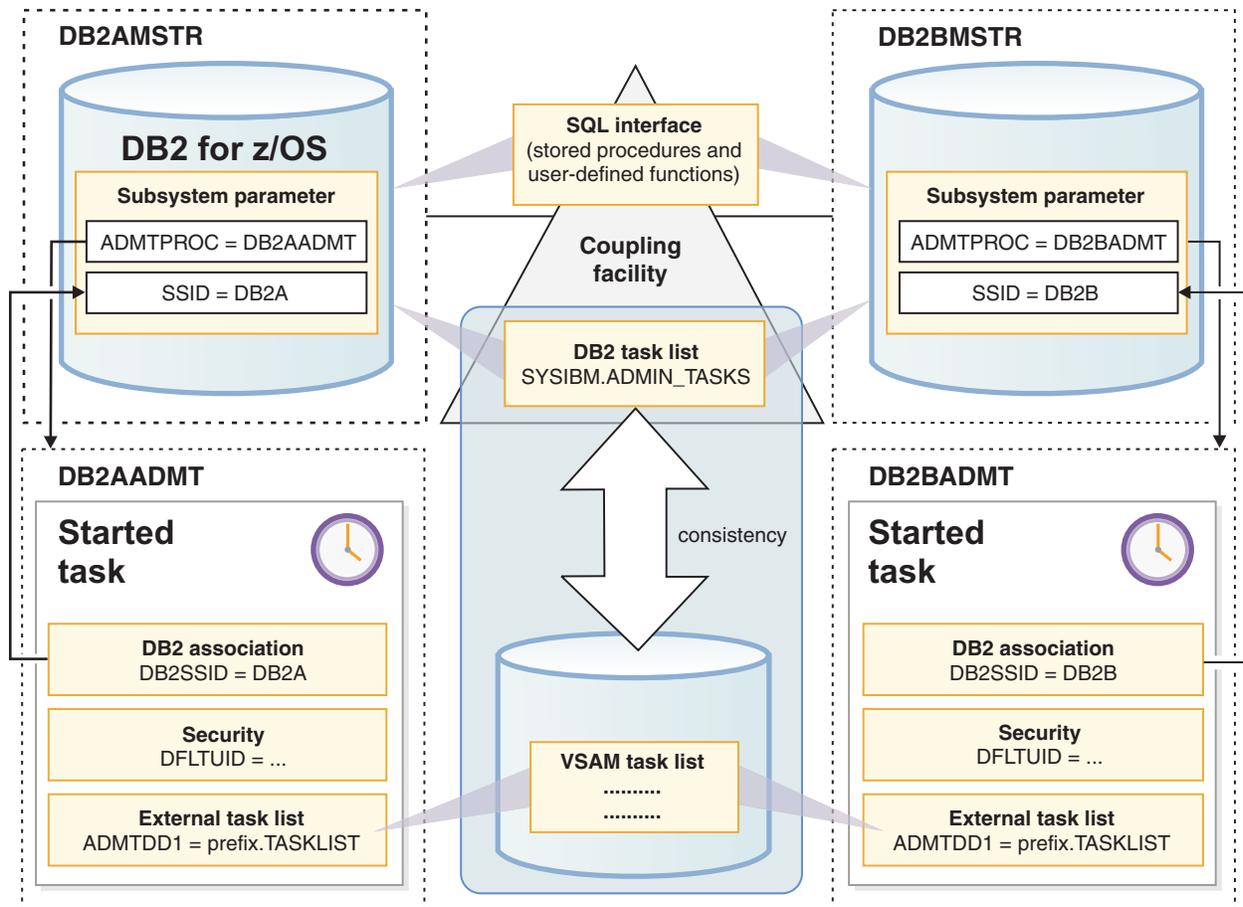


Figure 30. Administrative task schedulers in a data sharing group

Tasks are not localized to a scheduler. They can be added, removed, or executed in any of the schedulers in the data sharing group with the same result. However, you can force the task to execute on a given scheduler by specifying the associated DB2 subsystem ID in the *DB2SSID* parameter when you schedule the task. The tasks that have no affinity to a given DB2 subsystem are executed among all schedulers. Their distribution cannot be predicted.

Security guidelines for the administrative task scheduler

The administrative task scheduler employs controls on access and execution to help maintain a secure environment.

Installation job DSNTIJRA is responsible for establishing the security environment for the administrative task scheduler. Installation job DSNTIJSJ is responsible for establishing the security environment in DB2 for accessing the scheduler interface.

The following figure shows all of the security checkpoints that are associated with using the scheduler.

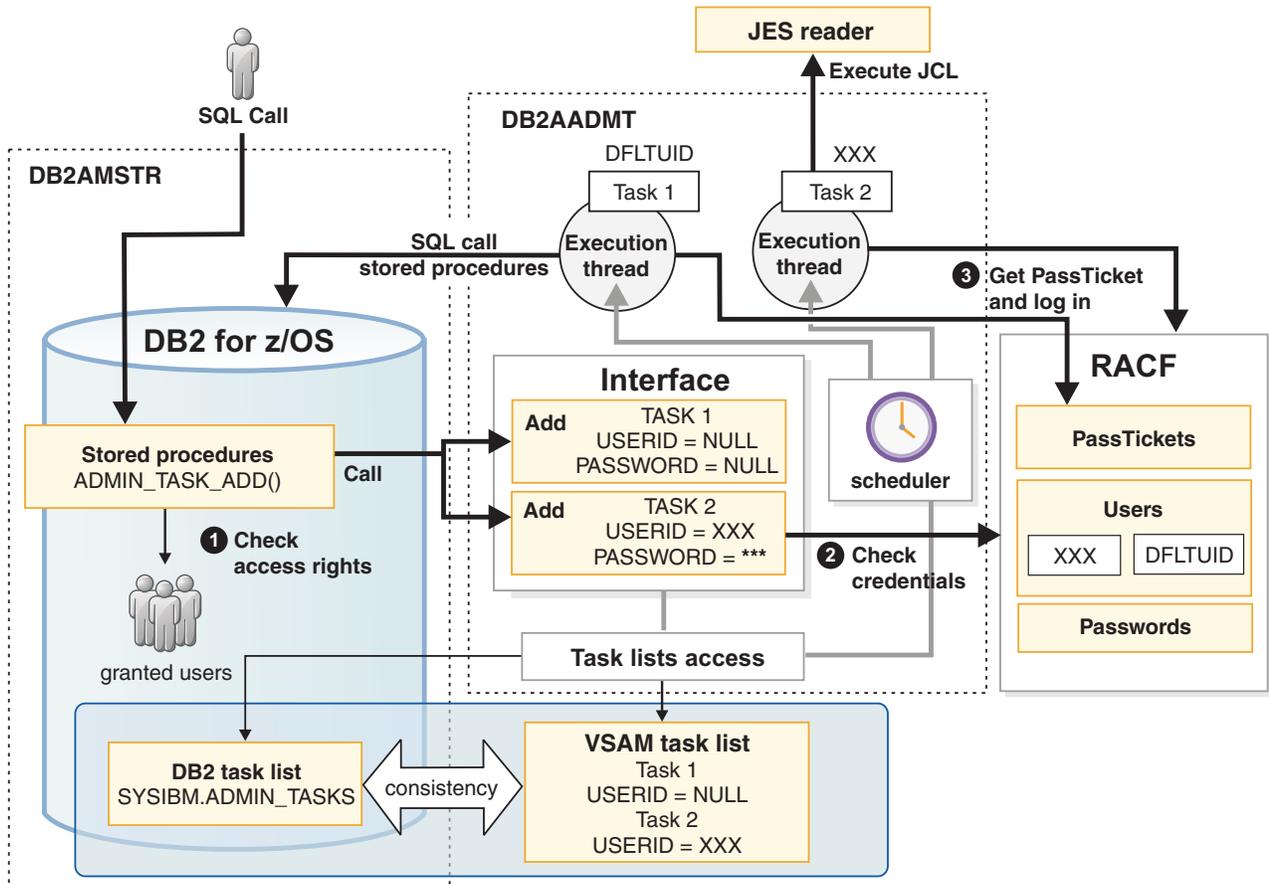


Figure 31. Security checkpoints for the administrative task scheduler

Related information

"Installation step 19: Set up the administrative scheduler" (DB2 Installation Guide)

"Migration step 25: Set up the administrative scheduler" (DB2 Installation Guide)

"Installation step 16: Define and bind DB2 objects: DSNTIJSJ" (DB2 Installation Guide)

"Migration step 22: Bind SPUFI and DCLGEN and user-maintained database activity: DSNTIJSJ" (DB2 Installation Guide)

 BPX.DAEMON FACILITY

User roles in the administrative task scheduler

Three user roles are involved in the use of the administrative task scheduler: the started task user, the interface users and the execution users.

The scheduler started task is associated to the user STARTUID in RACF, so that the scheduler is running in the security context of this user. This user, the started task user, should have access to the resources of the scheduler: it is granted UPDATE access on the DB2 table SYSIBM.ADMIN_TASKS and it is allowed to write into the VSAM data set containing the redundant task list.

The users or groups of users who have access to the SQL interface of the scheduler are allowed to add, remove or list scheduled tasks. To specify who is authorized to add, remove or list a scheduled task, use the GRANT command in DB2. All interface users are granted EXECUTE access on the scheduler stored procedures and user-defined table functions. They also are granted READ access on the DB2 table SYSIBM.ADMIN_TASKS.

Each scheduled task in the scheduler is associated with an execution user who will execute this task. When not explicitly given by the user, a default execution user DFLTUID defined in the scheduler started task is used. The scheduler execution threads switch to the security context of this user before executing the task.

Protection of the interface of the administrative task scheduler

The administrative task scheduler interface is protected against unauthorized access by other users. Credentials of a task are checked but not stored.

Users with EXECUTE rights on one of the stored procedures or user-defined table functions of the scheduler interface are allowed to execute the corresponding functionality: adding a scheduled task, removing a scheduled task, or listing the scheduled tasks or their execution status. The entire interface is configured by default with PUBLIC access rights during the installation.

Recommendations:

- Grant rights to groups or roles, rather than to individual authorization IDs.
- Restrict access to the ADMIN_TASK_ADD and ADMIN_TASK_REMOVE stored procedures to users with a business need for their use. Access to the user-defined table functions that list tasks and execution status can remain unrestricted.

The authorization ID of the DB2 thread that called the stored procedure ADMIN_TASK_ADD is passed to the scheduler and stored in the task list with the the task definition. The ADMIN_TASK_ADD stored procedure gathers the authorities granted to this authorization ID from the subsystem parameters and from the catalog table, and passes them over to the scheduler. The same mechanism is used in ADMIN_TASK_REMOVE to verify that the user is permitted to remove the task.

A task in the scheduler task list can be removed by the owner of the task, or by any user that has SYSOPR, SYSCTRL, or SYSADM privileges. The owner of a task is the CURRENT SQLID of the process at the time the task was added to the task list.

Protection of the resources of the administrative task scheduler

The administrative task scheduler task lists are protected against unauthorized use by other users than the started task execution user.

The VSAM resource (by default DSNC910.TASKLIST, where DSNC910 is the DB2 catalog prefix) that stores the administrative scheduler task list must be protected in RACF against unauthorized access. Only the administrative scheduler started task user has UPDATE authority on the VSAM resources. No other users should have any access.

A similar security concept is implemented for the SYSIBM.ADMIN_TASKS table, which stores a redundant copy of the scheduled tasks. Only the scheduler started tasks user has SELECT, INSERT, DELETE, or UPDATE authority on this resource. Users with EXECUTE rights on the user-defined functions ADMIN_TASK_LIST and ADMIN_TASK_STATUS have only SELECT authority on the table SYSIBM.ADMIN_TASKS.

Secure execution of tasks in the administrative task scheduler

The administrative task scheduler execution threads always switch to the security context of the execution user before executing a task. The user is authenticated through the use of passtickets.

The first action that is taken by the scheduler when starting a task execution is to switch its security context to the context of the execution user. The execution user can be explicitly identified by the ADMIN_TASK_ADD *user-ID* parameter or can be the default user.

If the task must run under a certain authority, including an authority different from the caller of the stored procedure, credentials are passed on to the administrative task scheduler. These credentials are not stored anywhere. They are validated by RACF to ensure that the caller of the stored procedure at the task definition time has the right to assume the desired security context. Therefore you can use a *passticket* (encrypted single-use password) in the password parameter of the ADMIN_TASK_ADD stored procedure. If no credentials are provided, then the administrative scheduler executes the tasks under its default execution user.

The scheduler generates and uses passtickets for executing the tasks under the corresponding authority. Each task executes after switching to the requested security context using the user ID and the generated passticket.

No password is stored in the task scheduler, but the scheduler is defined as a trusted program in RACF, and is allowed to get passtickets for any user. The scheduler sub-thread requires a passticket from RACF and logs in using this single-use password. The execution of the task then occurs in the switched security concept, allowing the task to have access to the resources defined for this execution user. After the execution, the security context is switched back to the scheduled task user.

The administrative scheduler started task module (DSNADMT0) uses the pthread_security_np() function to switch users. If the BPX.DAEMON facility class is active, then DSNADMT0 must be defined to RACF program control. If the BPX.DAEMON facility class is not active or if the scheduler is not defined to RACF program control, error EDC5139I is returned when trying to switch to another security level.

The scheduler resources should be protected from unintended impact when executing a task. Therefore, the scheduler started task user STARTUID, which has access to the scheduler resources, must not be used as default execution user DFLTUID, and it should not be specified in the *user-ID* parameter. The scheduler will not start if the started task user and the default execution user are identical. The default execution user should have as few rights as possible, to avoid impacting any resources if no user is defined for a task.

The default execution user has no authority except to attach to DB2 and write to the JES reader.

Execution of scheduled tasks in the administrative task scheduler

The administrative task scheduler manages the point in time, the security, the input parameters and the status of the task execution.

The scheduler work is based on scheduled tasks defined by the user. A task is mainly composed of a schedule definition and a work definition. The scheduler work is based on scheduled tasks, each scheduled task is a basic user-defined unit of work. Each task is associated with a unique task name. Up to 9999 tasks are supported in the scheduler at one time.

A scheduled task consists of a schedule and a work definition. The schedule part tells the scheduler when to execute the task. A user defines an execution window of time during which execution is permitted and either a time-based schedule or an event that triggers the execution of the job in this window. The work definition specifies what to execute, either a JCL job or a stored procedure, and the authority (user) under which to execute the task.

Multi-threading in the administrative task scheduler

The administrative task scheduler uses a pool of execution threads that allow it to execute many tasks simultaneously.

The scheduler is multi-threaded to be able to simultaneously execute different tasks: it starts the execution of scheduled tasks and then waits for their completion. The execution of a task is delegated by the scheduler to one of its sub-threads that starts the execution, waits until the execution completes, and gathers the execution status. Each sub-thread can be used for any type of tasks.

The maximum number of sub-threads is determined by the parameter MAXTHD of the scheduler started task, which by default is 99. Therefore, up to 99 tasks can be executed simultaneously by the scheduler. To reduce the memory usage of the scheduler, reduce the number of sub-threads by specifying a lower MAXTHD parameter value.

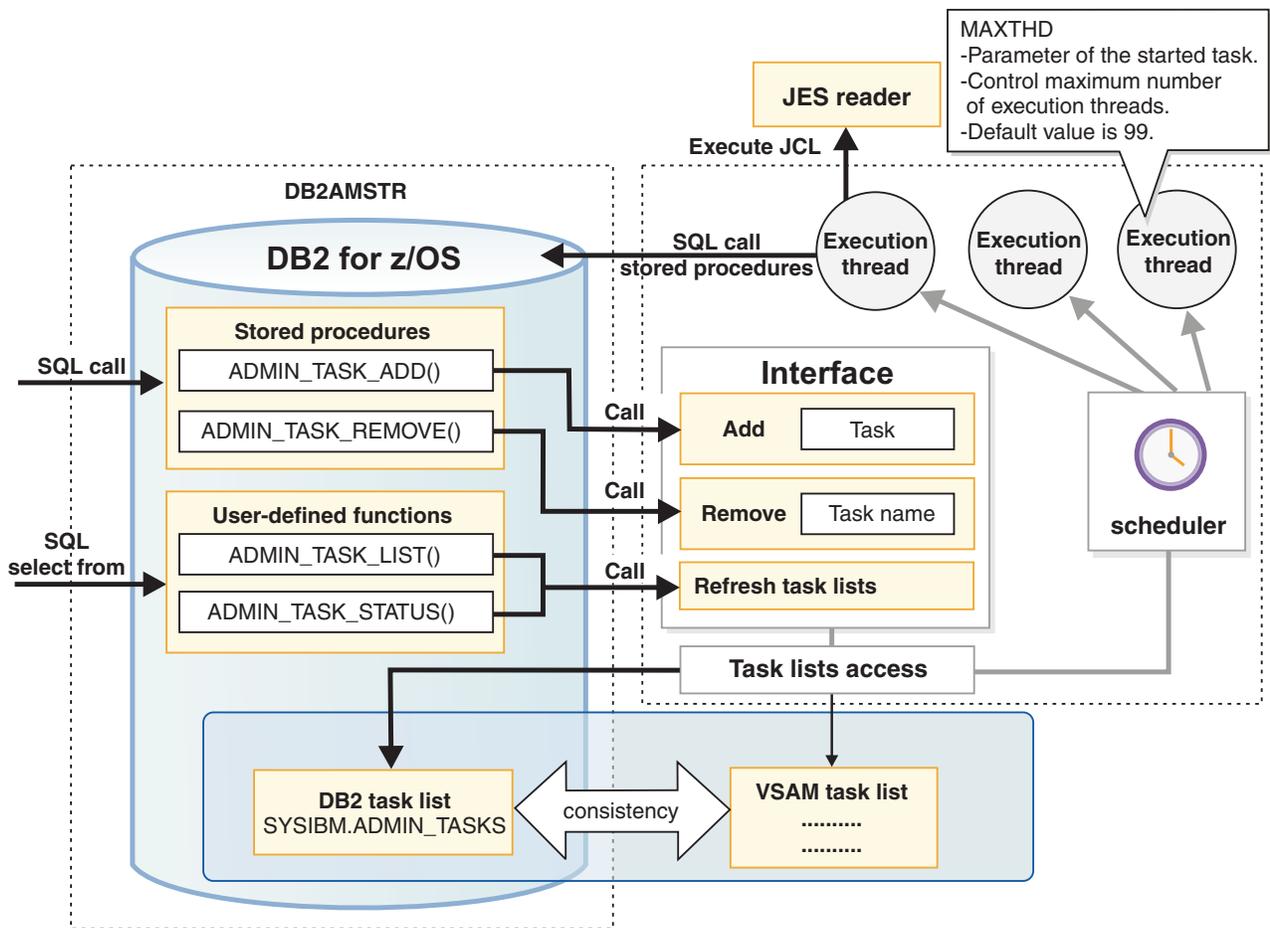


Figure 32. Multi-threading in the administrative task scheduler

The minimum permitted value for MAXTHD is 1, but should not be lower than the maximum number of tasks that you expect to execute simultaneously. If there are more tasks to be executed simultaneously than sub-threads available, some tasks will not start executing immediately. The scheduler will try to find an available sub-thread within one minute of when the task is scheduled for execution. As a result, multiple short tasks might be serialized in the same sub-thread, provided that their total execution time does not go over this minute.

The parameters of the scheduler started task are not positional. Place parameters in a single string separated by blank spaces.

If a task execution still cannot be started one minute after it should have been, the execution is skipped and the last execution status of this task is set to the NOTRUN state. The following message displays on the operator's console.

```
DSNA678I csect-name THE NUMBER OF TASKS TO BE CONCURRENTLY
EXECUTED BY THE ADMIN SCHEDULER proc-name EXCEEDS max-threads
```

If this happens, increase the MAXTHD parameter value and restart the scheduler.

Scheduled execution of a stored procedure

The scheduler gets parameter values for the stored procedure executing the given SQL select statement, then calls the stored procedure in DB2.

One execution sub-thread of the scheduler is used to execute a task that describes a stored procedure call. The execution sub-thread first connects to the DB2 member indicated in the *DB2SSID* parameter of the scheduler started task. If the connection cannot be established, the execution is skipped and the last execution status is set to the NOTRUN state.

After connection, the parameter values of the stored procedure are retrieved from DB2 using the select statement defined in the task parameter *procedure-input*. If an error occurs retrieving those parameter values, the last execution status of the task is set to the error returned by DB2, and the stored procedure itself is not called.

Otherwise, the stored procedure is called with the retrieved parameter values. The stored procedure name is concatenated from the task parameters *procedure-schema* and *procedure-name*. The SQL CALL command is synchronous and the execution thread is blocked until the stored procedure finishes execution. Then the last execution status is set to the values returned by DB2. Finally, a COMMIT statement is issued and the connection to DB2 is closed.

The execution status of DB2 stored procedures always contain null values in the JOB_ID, MAXRC, COMPLETION_TYPE, SYSTEM_ABENDCD and USER_ABENDCD fields. In the case of a DB2 error, the fields SQLCODE, SQLSTATE, SQLERRMC, and SQLERRP will contain what DB2 returned to the stored procedure.

How the administrative task scheduler works with Unicode

The administrative task scheduler is able to retrieve and pass Unicode parameters to a DB2 stored procedure.

If the stored procedure accepts Unicode parameters, or if it does not accept Unicode parameters but the retrieved parameter values do not contain any special character that cannot be expressed in EBCDIC or ASCII, no character will be broken.

However, the Unicode values must be retrieved through a select statement expressed in EBCDIC, so that special characters cannot be used in the table names or in the column names where the parameter values are retrieved.

Scheduled execution of a JCL job

The administrative task scheduler sends the JCL job to the JES reader. It can optionally wait for the job to terminate and purge it from the JES job list.

One execution sub-thread of the scheduler is used to execute a task that locates a data set containing a JCL job. The sub-thread reads the JCL job from the data set where it is stored, identified by the task parameters *JCL-library* and *JCL-member*. The data set can be sequential or partitioned. For a sequential data set, *JCL-member* is NULL.

In the case of an error, the error is written into the last execution status of the task. Otherwise, the job is submitted to the internal JES reader. According to the *job_wait* parameter, the sub-thread waits for the execution completion or not. When the sub-thread waits for completion, the last execution status includes the complete returned values provided by the JES reader. Otherwise, it contains the JCL job ID and a success message.

- If *job_wait* is set to NO, the sub-thread does not wait until the job completes execution and returns immediately after the job submission. The task execution status is set to the submission status, the result of the job execution itself will not be available.
- If *job_wait* is set to YES, the sub-thread simulates a synchronous execution of the JCL job. It waits until the job execution completes, get the job status from the JES reader and fills in the last execution status of the task.
- If *job_wait* is set to PURGE, the sub-thread purges the job output from the JES reader after execution. Execution is the same as for *job_wait*=YES.

JCL job execution status always contains a null value in the SQLCODE, SQLSTATE, SQLERRMC, and SQLERRP fields. If the job can be submitted successfully to the JES reader, the field JOB_ID contains the ID of the job in the JES reader. If the job is executed asynchronously, the MAXRC, COMPLETION_TYPE, SYSTEM_ABENDCD and USER_ABENDCD fields will also be null values, because the sub-thread does not wait for job completion before writing the status. If the job was executed synchronously, those fields contain the values returned by the JES reader.

Execution of scheduled tasks in a data sharing environment

All administrative task schedulers of the data sharing group cooperate in the execution of the scheduled tasks.

In a data sharing environment, several schedulers cooperate in the execution of the scheduled tasks. If a task has a member affinity, that is, if its parameter DB2-SSID contains the name of a DB2 member, only the scheduler that is associated with this DB2 member will execute this task. The task is executed as in a non-data sharing mode. If this scheduler is unavailable, the task will not be executed.

When a task has no member affinity, that is, if *DB2-SSID* is a null value, the scheduler that wakes first executes the task. If the task execution can complete on this scheduler, for example because its associated DB2 member is not running or because all of its execution threads are busy, other schedulers will not try executing this task. However, if the scheduler cannot start the execution, the other schedulers will try successively to start executing the task until one succeeds or all fail in executing the task.

The distribution of a task execution on the one or the other scheduler cannot be predicted. Successive executions of the same task can be done on different schedulers.

Chapter 15. Monitoring and controlling DB2 and its connections

You can control and monitor various aspects of a DB2 for z/OS environment.

The following operations, described in this section, require more understanding of what DB2 is doing:

Related concepts

Chapter 11, “DB2 basic operational concepts,” on page 355

Controlling DB2 databases and buffer pools

DB2 databases are controlled by the START DATABASE, DISPLAY DATABASE, and STOP DATABASE commands.

GUPI

START DATABASE

Makes a database or individual partitions available. Also removes pages from the logical page list (LPL).

DISPLAY DATABASE

Displays status, user, and locking information for a database.

STOP DATABASE

Makes a database or individual partitions unavailable after existing users have quiesced. DB2 also closes and deallocates the data sets.

GUPI

Related tasks

“Starting databases”

“Monitoring databases” on page 415

“Using the STOP DATABASE command to make objects available” on page 422

Starting databases

The command START DATABASE (*) starts all databases for which you have the STARTDB privilege.

GUPI

The privilege can be explicitly granted, or it can belong implicitly to a level of authority (DBMAINT and above). The command starts the database, but not necessarily all the objects that it contains. Any table spaces or index spaces in a restricted mode remain in a restricted mode and are not started.

START DATABASE (*) does not start the DB2 directory (DSNDB01), the DB2 catalog (DSNDB06), or the DB2 work file database (called DSNDB07, except in a data sharing environment). Start these databases explicitly using the SPACENAM option. Also, START DATABASE (*) does not start table spaces or index spaces that have been explicitly stopped by the STOP DATABASE command.

Use the PART keyword of the START DATABASE command to start the individual partitions of a table space. It can also be used to start individual partitions of a partitioning index or logical partitions of a nonpartitioning index. The started or stopped state of other partitions is unchanged.

The START DATABASE and STOP DATABASE commands can be used with the SPACENAM and PART options to control table spaces, index spaces, or partitions. For example, the following command starts two partitions of table space DSN8S91E in the database DSN8D91A:

```
-START DATABASE (DSN8D91A) SPACENAM (DSN8S91E) PART (1,2)
```

GUIP

Starting an object with a specific status

You can start a database, table space, or an index space with a specific status that limits access to it.

GUIP

Status Provides this access

RW Read-write. This is the default value.

RO Read only. You cannot change the data.

UT Utility only. The object is available only to the DB2 utilities.

Databases, table spaces, and index spaces are started with RW status when they are created. You can make any of them unavailable by using the STOP DATABASE command. DB2 can also make them unavailable when it detects an error.

In cases when the object was explicitly stopped, you can make them available again using the command START DATABASE. For example, the following command starts all table spaces and index spaces in database DSN8D91A for read-only access:

```
-START DATABASE (DSN8D91A) SPACENAM(*) ACCESS(RO)
```

The system responds with this message:

```
DSN9022I - DSNTDDIS '-START DATABASE' NORMAL COMPLETION
```

GUIP

Starting a table space or index space that has restrictions

DB2 can make an object unavailable for a variety of reasons. Typically, in those cases, the data is unreliable, and the object needs some attention before it can be started.

GUIP

An example of such a restriction is when the table space is placed in COPY-pending status. That status makes a table space or partition unavailable until an image copy of the object is taken.

Important: These restrictions are a necessary part of protecting the integrity of the data. If you start an object that has restrictions, the data in the object might not be reliable.

However, in certain circumstances, it might be reasonable to force availability. For example, a table might contain test data whose consistency is not critical. In those cases, start the objects by using the ACCESS(FORCE) option of the START DATABASE command. For example:

```
-START DATABASE (DSN8D91A) SPACENAM (DSN8S91E) ACCESS(FORCE)
```

The command releases most restrictions for the named objects. These objects must be explicitly named in a list following the SPACENAM option.

DB2 cannot process the START DATABASE ACCESS(FORCE) request if postponed-abort or indoubt units of recovery exist. The RESTP (restart-pending) status and the AREST (advisory restart-pending) status remain in effect until either automatic backout processing completes or you perform one of the following actions:

- Issue the RECOVER POSTPONED command to complete backout activity.
- Issue the RECOVER POSTPONED CANCEL command to cancel all of the postponed-abort units of recovery.
- Conditionally restart or cold start DB2.

DB2 cannot apply the START DATABASE ACCESS(FORCE) command to that object if a utility from a previous release of DB2 places an object in one of the following restrictive states:

- UTRO (utility restrictive state, read-only access allowed)
- UTRW (utility restrictive state, read and write access allowed)
- UTUT (utility restrictive state, utility exclusive control)

To reset these restrictive states, you must start the release of DB2 that originally ran the utility and terminate the utility from that release.

GUPI

Related tasks

“Resolving postponed units of recovery” on page 512

Monitoring databases

You can use the command DISPLAY DATABASE to obtain information about the status of databases and the table spaces and index spaces within each database. If applicable, the output also includes information about physical I/O errors for those objects.

GUPI

To monitor databases:

1. Issue the DISPLAY DATABASE command as follows:

```
-DISPLAY DATABASE (dbname)
```

This command results in the following messages:

```
11:44:32 DSNT360I - *****
11:44:32 DSNT361I - * DISPLAY DATABASE SUMMARY
11:44:32          *      report_type_list
11:44:32 DSNT360I - *****
11:44:32 DSNT362I -      DATABASE = dbname  STATUS = xx
                        DBD LENGTH = yyyy

11:44:32 DSNT397I -
NAME      TYPE PART  STATUS          PHYERRLO  PHYERRHI  CATALOG  PIECE
-----
```

```

D1      TS      RW,UTRO
D2      TS      RW
D3      TS      STOP
D4      IX      RO
D5      IX      STOP
D6      IX      UT
LOB1    LS      RW
***** DISPLAY OF DATABASE dbname ENDED *****
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION

```

In the preceding messages:

- *report_type_list* indicates which options were included when the DISPLAY DATABASE command was issued.
 - *dbname* is an 8-byte character string that indicates the database name. The pattern-matching character, *, is allowed at the beginning, middle, and end of *dbname*.
 - STATUS is a combination of one or more status codes, delimited by commas. The maximum length of the string is 17 characters. If the status exceeds 17 characters, those characters are wrapped onto the next status line. Anything that exceeds 17 characters on the second status line is truncated.
2. Optional: Use the pattern-matching character, *, in the DISPLAY DATABASE, START DATABASE, and STOP DATABASE commands. You can use the pattern-matching character in the beginning, middle, and end of the database and table space names.
 3. Use additional keywords to tailor the DISPLAY DATABASE command so that you can monitor what you want:
 - The keyword ONLY can be added to the command DISPLAY DATABASE. When ONLY is specified with the DATABASE keyword but not the SPACENAM keyword, all other keywords except RESTRICT, LIMIT, and AFTER are ignored. Use DISPLAY DATABASE ONLY as follows:

```
-DISPLAY DATABASE(*S*DB*) ONLY
```

This command results in the following messages:

```

11:44:32 DSNT360I - *****
11:44:32 DSNT361I - * DISPLAY DATABASE SUMMARY
11:44:32          * GLOBAL
11:44:32 DSNT360I - *****
11:44:32 DSNT362I - DATABASE = DSNDB01 STATUS = RW
                   DBD LENGTH = 8066
11:44:32 DSNT360I - *****
11:44:32 DSNT362I - DATABASE = DSNDB04 STATUS = RW
                   DBD LENGTH = 21294
11:44:32 DSNT360I - *****
11:44:32 DSNT362I - DATABASE = DSNDB06 STATUS = RW
                   DBD LENGTH = 32985
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION

```

In the preceding messages:

- DATABASE (*S*DB*) displays databases that begin with any letter, have the letter S followed by any letters, and then the letters DB followed by any letters.
- ONLY restricts the display to databases names that fit the criteria.
- The RESTRICT option of the DISPLAY DATABASE command limits the display to objects that are currently set to a restrictive status. You can additionally specify one or more keywords to limit the display further to include only objects that are set to a particular restrictive status.

- The ADVISORY option on the DISPLAY DATABASE command limits the display to table spaces or indexes that require some corrective action. Use the DISPLAY DATABASE ADVISORY command without the RESTRICT option to determine when:
 - An index space is in the informational COPY-pending (ICOPY) advisory status.
 - A base table space is in the auxiliary-warning (AUXW) advisory status.
- The OVERVIEW option of the DISPLAY DATABASE command displays all objects within a database. This option shows each object in the database on one line, does not isolate an object by partition, and does not show exception states. The OVERVIEW option displays only object types and the number of data set partitions in each object. OVERVIEW is mutually exclusive with all keywords other than SPACENAM, LIMIT, and AFTER. Use DISPLAY DATABASE OVERVIEW as follows:

```
-DISPLAY DATABASE(DB486A) SPACENAM(*) OVERVIEW
```

This command results in the following messages:

```
DSNT360I = *****
DSNT361I = *   DISPLAY DATABASE SUMMARY 483
          *   GLOBAL OVERVIEW
DSNT360I = *****
DSNT362I =     DATABASE = DB486A STATUS = RW 485
          DBD LENGTH = 4028
DSNT397I = 486
NAME      TYPE PARTS
-----
TS486A    TS     0004
IX486A    IX     L0004
IX486B    IX     0004
TS486C    TS
IX486C    IX
***** DISPLAY OF DATABASE DB486A ENDED *****
DSN9022I = DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

The display indicates that five objects are in database DB486A: two table spaces and three indexes. Table space TS486A has four parts, and table space TS486C is nonpartitioned. Index IX486A is a nonpartitioning index for table space TS486A, and index IX486B is a partitioned index with four parts for table space TS486A. Index IX486C is a nonpartitioned index for table space

TS486C. 

Related reference

"-DISPLAY DATABASE (DB2)" (DB2 Command Reference)

"Advisory or restrictive states" (DB2 Utility Guide and Reference)

Obtaining information about application programs

You can obtain various kinds of information about application programs using particular databases, table spaces or index spaces by using the DISPLAY DATABASE command. You can identify who or what is using the object and what locks are being held on the objects.

Identifying who and what are using an object

You can obtain information about users and applications that are using an object, and about the units of work that are accessing data.

 You can obtain the following information:

- The names of the application programs currently using the database or space

- The authorization IDs of the users of these application programs
- The logical unit of work IDs of the database access threads that access data on behalf of the specified remote locations

For example, to obtain this information you can issue a command that names partitions 2, 3, and 4 in table space TPAUGF01 in database DBAUGF01:

```
-DISPLAY DATABASE (DBAUGF01) SPACENAM (TPAUGF01) PART (2:4) USE
```

DB2 returns a list similar to this one:

```
DSNT360I : *****
DSNT361I : * DISPLAY DATABASE SUMMARY
          * GLOBAL USE
DSNT360I : *****
DSNT362I : DATABASE = DBAUGF01 STATUS = RW
          DBD LENGTH = 8066
DSNT397I :
NAME      TYPE PART  STATUS          CONNID  CORRID  USERID
-----
TPAUGF01 TS    0002 RW           BATCH    S3341209  ADMF001
-
          MEMBER NAME V61A
TPAUGF01 TS    0003 RW           BATCH    S3341209  ADMF001
-
          MEMBER NAME V61A
TPAUGF01 TS    0004 RW           BATCH    S3341209  ADMF001
-
          MEMBER NAME V61A
***** DISPLAY OF DATABASE DBAUGF01 ENDED *****
DSN9022I : DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

GUIP

Determining which programs are holding locks on an object

You can use the DISPLAY DATABASE command to determine which programs are holding locks on an object.

GUIP

To determine which application programs currently hold locks on the database or space, issue a command that names table space TSPART in database DB01:

```
-DISPLAY DATABASE(DB01) SPACENAM(TSPART) LOCKS
```

DB2 returns a list similar to this one:

```
17:45:42 DSNT360I - *****
17:45:42 DSNT361I - * DISPLAY DATABASE SUMMARY
17:45:42          * GLOBAL LOCKS
17:45:42 DSNT360I - *****
17:45:42 DSNT362I - DATABASE = DB01 STATUS = RW
17:45:42          DBD LENGTH = yyyy
17:45:42 DSNT397I -
NAME      TYPE PART  STATUS          CONNID  CORRID  LOCKINFO
-----
TSPART    TS    0001 RW           LSS001  DSN2SQL  H-IX,P,C
TSPART    TS    0002 RW           LSS001  DSN2SQL  H-IX,P,C
TSPART    TS    0003 RW           LSS001  DSN2SQL  H-IX,P,C
TSPART    TS    0004 RW           LSS001  DSN2SQL  H-IX,P,C
***** DISPLAY OF DATABASE DB01 ENDED *****
17:45:44 DSN9022I . DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

Use the LOCKS ONLY keywords on the DISPLAY DATABASE command to display only spaces that have locks. You can substitute the LOCKS keyword with USE, CLAIMERS, LPL, or WEPR to display only databases that fit the criteria. Use DISPLAY DATABASE as follows:

```
-DISPLAY DATABASE (DSNDB06) SPACENAM(*) LOCKS ONLY
```

This command results in the following messages:

```
11:44:32 DSNT360I - *****
11:44:32 DSNT361I - * DISPLAY DATABASE SUMMARY
11:44:32          * GLOBAL LOCKS
11:44:32 DSNT360I - *****
11:44:32 DSNT362I - DATABASE = DSNDB06 STATUS = RW
                   DBD LENGTH = 60560

11:44:32 DSNT397I -
NAME      TYPE PART STATUS          CONNID  CORRID      LOCKINFO
-----
SYSDBASE TS          RW          DSN      020.DBCMD 06 H-IS,S,C
***** DISPLAY OF DATABASE DSNDB06 ENDED *****
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

GUPI

Related reference

"DSNT396I" (DB2 Messages)

"-DISPLAY DATABASE (DB2)" (DB2 Command Reference)

Obtaining information about and handling pages in error

You can view information about pages that are in error.

Characteristics of pages that are in error

A page can be logically or physically in error.

GUPI A page is logically in error if its problem can be fixed without redefining new disk tracks or volumes. For example, if DB2 cannot write a page to disk because of a connectivity problem, the page is logically in error. DB2 inserts entries for pages that are logically in error in a *logical page list* (LPL).

A page is physically in error if physical errors exist, such as device errors. Such errors appear on the *write error page range* (WEPR). The range has a low and high page, which are the same if only one page has errors.

If the cause of the problem is undetermined, the error is first recorded in the LPL. If recovery from the LPL is unsuccessful, the error is then recorded on the error page range.

Write errors for large object (LOB) table spaces that are defined with LOG NO cause the unit of work to be rolled back. Because the pages are written during normal deferred write processing, they can appear in the LPL and WEPR. The LOB data pages for a LOB table space with the LOG NO attribute are not written to LPL or WEPR. The space map pages are written during normal deferred write processing and can appear in the LPL and WEPR.

A program that tries to read data from a page that is listed on the LPL or WEPR receives an SQLCODE for “resource unavailable.” To access the page (or pages in the error range), you must first recover the data from the existing database copy and the log. **GUPI**

Displaying the logical page list

You can check the existence of LPL entries by issuing the DISPLAY DATABASE command with the LPL option.

GUPI The ONLY option restricts the output to objects that have LPL pages. For example:

```
-DISPLAY DATABASE(DBFW8401) SPACENAM(*) LPL ONLY
```

The following output is produced:

```
DSNT360I = *****
DSNT361I = * DISPLAY DATABASE SUMMARY
          * GLOBAL LPL
DSNT360I = *****
DSNT362I = DATABASE = DBFW8401 STATUS = RW,LPL
          DBD LENGTH = 8066
DSNT397I =
NAME     TYPE PART  STATUS          LPL PAGES
-----
TPFW8401 TS     0001 RW,LPL          000000-000004
ICFW8401 IX     L0001 RW,LPL          000000,000003
IXFW8402 IX              RW,LPL          000000,000003-000005
-----
          000007,000008-00000B
          000080-000090
***** DISPLAY OF DATABASE DBFW8401 ENDED *****
DSN9022I = DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

The display indicates that the pages that are listed in the LPL PAGES column are unavailable for access. **GUPI**

Related reference

“-DISPLAY DATABASE (DB2)” (DB2 Command Reference)

Removing pages from the logical page list

Although the DB2 subsystem always attempts automated recovery of LPL pages when the pages are added to the LPL, you can also perform manual recovery.

GUPI When an object has pages on the LPL, you can use one of several methods to manually remove those pages and make them available for access when DB2 is running:

- Start the object with access (RW) or (RO). That command is valid even if the table space is already started.

When you issue the START DATABASE command, you see message DSNI006I, indicating that LPL recovery has begun. Message DSNI022I is issued periodically to give you the progress of the recovery. When recovery is complete, you see DSNI021I.

When you issue the START DATABASE command for a LOB table space that is defined as LOG NO, and DB2 detects that log records that are required for LPL recovery are missing due to the LOG NO attribute, the LOB table space is placed in AUXW status, and the LOB is invalidated.

- Run the RECOVER or REBUILD INDEX utility on the object.

The only exception to this is when a logical partition of a nonpartitioned index is in the LPL and has RECP status. If you want to recover the logical partition by using REBUILD INDEX with the PART keyword, you must first use the START DATABASE command to clear the LPL pages.

- Run the LOAD utility with the REPLACE option on the object.
- Issue an SQL DROP statement for the object.

Only the following utilities can be run on an object with pages in the LPL:

- LOAD with the REPLACE option
- MERGECOPY
- REBUILD INDEX
- RECOVER, *except*:
 - RECOVER...PAGE
 - RECOVER...ERROR RANGE
- REPAIR with the SET statement
- REPORT 

Displaying a write error page range

Use DISPLAY DATABASE to display the range of error pages.

 For example:

```
-DISPLAY DATABASE (DBPARTS) SPACENAM (TSPART01) WEPR
```

The preceding command returns a list similar to this one:

```
11:44:32 DSNT360I - *****
11:44:32 DSNT361I - * DISPLAY DATABASE SUMMARY
11:44:32          *          GLOBAL WEPR
11:44:32 DSNT360I - *****
11:44:32 DSNT362I -          DATABASE = DBPARTS STATUS = RW
                  DBD LENGTH = yyyy

11:44:32 DSNT397I -
NAME      TYPE PART  STATUS          PHYERRLO  PHYERRHI CATALOG  PIECE
-----
TSPART01 TS   0001 RW,UTRO        00000002 00000004 DSNCAT   000
TSPART01 TS   0002 RW,UTRO        00000009 00000013 DSNCAT   001
TSPART01 TS   0003 RO
TSPART01 TS   0004 STOP
TSPART01 TS   0005 UT
***** DISPLAY OF DATABASE DBPARTS ENDED          *****
11:45:15 DSN9022I - DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

In the previous messages:

- PHYERRLO and PHYERRHI identify the range of pages that were being read when the I/O errors occurred. PHYERRLO is an 8-digit hexadecimal number representing the lowest page that is found in error, and PHYERRHI represents the highest page that is found in error.
- PIECE, a 3-digit integer, is a unique identifier for the data set and supports the page set that contains physical I/O errors. 

Related reference

"DSNT392I" (DB2 Messages)

Using the STOP DATABASE command to make objects available

You can make databases, table spaces, and index spaces unavailable by using the STOP DATABASE command.

If an object is in STOPP status, you must first issue the START DATABASE command to remove the STOPP status, and then issue the STOP DATABASE command.

When you issue the STOP DATABASE command for a table space, the data sets that contain that table space are closed and deallocated.

DB2 subsystem databases (catalog, directory, and work file) can also be stopped. After the directory is stopped, installation SYSADM authority is required to restart it.

GUIP To stop databases, table spaces, or index spaces:

Issue the STOP DATABASE command with the appropriate options.

Type of object that you want to stop	How to issue the STOP DATABASE command
To stop a physical partition of a table space:	Use the PART option.
To stop a physical partition of an index space:	Use the PART option.
To stop a logical partition within a nonpartitioning index that is associated with a partitioned table space:	Use the PART option.
To stop any kind of object as quickly as possible:	Use the AT(COMMIT) option.
To stop user-defined databases:	Start database DSNDB01 and table spaces DSNDB01.DBD01 and DSNDB01.SYSLGRNX before you stop user-defined databases. If you do not do this, you will receive message DSNI003I. Resolve the problem and run the job again.
To stop the work file database:	Start database DSNDB01 and table spaces DSNDB01.DBD01 and DSNDB01.SYSLGRNX before you stop the work file database. If you do not do this, you will receive message DSNI003I. Resolve the problem and run the job again.

GUIP

Commands to stop databases

The STOP DATABASE command has several options that you can use to control how the database stops.

GUIP The following examples illustrate ways to use the STOP DATABASE command:

-STOP DATABASE (*)

Stops all databases for which you have STOPDB authorization, except the DB2 directory (DSNDB01), the DB2 catalog (DSNDB06), or the DB2 work file database (called DSNDB07, except in a data sharing environment), all of which must be stopped explicitly.

-STOP DATABASE (dbname)

Stops a database and closes all of the data sets of the table spaces and index spaces in the database.

-STOP DATABASE (dbname, ...)

Stops the named databases and closes all of the table spaces and index spaces in the databases. If DSNDB01 is named in the database list, it should be last on the list because stopping the other databases requires that DSNDB01 be available.

-STOP DATABASE (dbname) SPACENAM (*)

Stops and closes all of the data sets of the table spaces and index spaces in the database. The status of the named database does not change.

-STOP DATABASE (dbname) SPACENAM (space-name, ...)

Stops and closes the data sets of the named table space or index space. The status of the named database does not change.

-STOP DATABASE (dbname) SPACENAM (space-name, ...) PART (integer)

Stops and closes the specified partition of the named table space or index space. The status of the named database does not change. If the named index space is nonpartitioned, DB2 cannot close the specified logical partition.

The data sets containing a table space are closed and deallocated by the preceding commands. 

Altering buffer pools

DB2 stores buffer pool attributes in the DB2 bootstrap data set. You can use the ALTER BUFFERPOOL command to alter buffer pool attributes.

 Buffer pool attributes, including buffer pool sizes, sequential steal thresholds, deferred write thresholds, and parallel sequential thresholds, are initially defined during the DB2 installation process. 

Related reference

"-ALTER BUFFERPOOL (DB2)" (DB2 Command Reference)

Related information

"Buffer pool thresholds" (DB2 Performance Monitoring and Tuning Guide)

Monitoring buffer pools

To monitor buffer pools, you can use the DISPLAY BUFFERPOOL command, which displays the current status for one or more active or inactive buffer pools. You can also request a summary or detail report.

 For example:

```
-DISPLAY BUFFERPOOL(BP0)
```

This command might produce a summary report such as this:

```

!DIS BUFFERPOOL(BP0)
DSNB401I ! BUFFERPOOL NAME BP0, BUFFERPOOL ID 0, USE COUNT 27
DSNB402I ! BUFFER POOL SIZE = 2000 BUFFERS
          ALLOCATED          =      2000   TO BE DELETED   =          0
          IN-USE/UPDATED     =          0
DSNB406I ! PAGE STEALING METHOD = LRU
DSNB404I ! THRESHOLDS -
          VP SEQUENTIAL      = 80
          DEFERRED WRITE     = 85   VERTICAL DEFERRED WRT = 10,15
          PARALLEL SEQUENTIAL = 50   ASSISTING PARALLEL SEQT= 0
DSN9022I ! DSNB1CMD '-DISPLAY BUFFERPOOL' NORMAL COMPLETION

```

GUPI

Related reference

"-DISPLAY BUFFERPOOL (DB2)" (DB2 Command Reference)

Controlling user-defined functions

User-defined functions are extensions to the SQL language, which you can invoke in an SQL statement wherever you can use expressions or built-in functions.

GUPI User-defined functions, like stored procedures, run in WLM-established address spaces. Because you might have an existing user-defined function named `DECFLOAT`, `QUANTIZE`, `NORMALIZE_DECFLOAT`, or `IS_IDENTICAL_TO`, you should fully qualify that function or use the `SET CURRENT PATH` special register to make sure that the function resolution resolves to the correct function.

You control DB2 user-defined functions by using various commands.

START FUNCTION SPECIFIC

Activates an external function that is stopped.

DISPLAY FUNCTION SPECIFIC

Displays statistics about external user-defined functions accessed by DB2 applications.

STOP FUNCTION SPECIFIC

Prevents DB2 from accepting SQL statements with invocations of the specified functions.

GUPI

Related tasks

"Monitoring and controlling stored procedures" on page 467

Related reference

"Sample user-defined functions" (DB2 SQL Reference)

Starting user-defined functions

Activate external functions that are stopped by using the DB2 `START FUNCTION SPECIFIC` command.

GUPI You cannot start built-in functions or user-defined functions that are sourced on another function.

To activate all or a specific set of stopped external functions, issue the `START FUNCTION SPECIFIC` command.

For example, assume that you want to start functions USERFN1 and USERFN2 in the PAYROLL schema. Issue this command:

```
START FUNCTION SPECIFIC(PAYROLL.USERFN1,PAYROLL.USERFN2)
```

The following output is produced:

```
DSNX973I START FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN1
DSNX973I START FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN2
```

GUIP

Monitoring user-defined functions

You monitor external user-defined functions by using the DISPLAY FUNCTION SPECIFIC command. This command displays statistics about the functions and lists the range of functions that are stopped because of a STOP FUNCTION SPECIFIC command.

GUIP

The DB2 command DISPLAY FUNCTION SPECIFIC displays statistics about external user-defined functions accessed by DB2 applications. This command displays one output line for each function that has been accessed by a DB2 application. Information returned by this command reflect a dynamic status. By the time DB2 displays the information, the status might have changed. Built-in functions or user-defined functions that are sourced on another function are not applicable to this command.

Use the DISPLAY FUNCTION SPECIFIC command to list the range of functions that are stopped because of a STOP FUNCTION SPECIFIC command. For example, to display information about functions in the PAYROLL schema and the HRPROD schema, issue this command:

```
-DISPLAY FUNCTION SPECIFIC(PAYROLL.*,HRPROD.*)
```

The following output is produced:

```
DSNX975I DSNX9DIS - DISPLAY FUNCTION SPECIFIC REPORT FOLLOWS-
----- SCHEMA=PAYROLL
FUNCTION      STATUS  ACTIVE  QUED  MAXQ  TIMEOUT  FAIL  WLM_ENV
PAYRFNC1     STARTED    0    0    1    0    0  PAYROLL
PAYRFNC2     STOPQUE    0    5    5    3    0  PAYROLL
PAYRFNC3     STARTED    2    0    6    0    0  PAYROLL
USERFNC4     STOPREJ    0    0    1    0    1  SANDBOX
----- SCHEMA=HRPROD
FUNCTION      STATUS  ACTIVE  QUED  MAXQ  TIMEOUT  FAIL  WLM_ENV
HRFNC1       STARTED    0    0    1    0    0  HRFUNCS
HRFNC2       STOPREJ    0    0    1    0    0  HRFUNCS
DSNX9DIS DISPLAY FUNCTION SPECIFIC REPORT COMPLETE
DSN9022I - DSNX9COM '-DISPLAY FUNC' NORMAL COMPLETION
```

GUIP

Stopping user-defined functions

You can prevent DB2 from accepting SQL statements with invocations of specific functions by using the DB2 command STOP FUNCTION SPECIFIC. This command

does not prevent SQL statements with invocations of the functions from running if they have already been queued or scheduled by DB2. You cannot stop built-in functions or user-defined functions that are sourced on another function.

GUPI To stop access to all or a specific set of external functions, issue the STOP FUNCTION SPECIFIC command. For example, issue a command like the following one, which stops functions USERFN1 and USERFN3 in the PAYROLL schema:

```
STOP FUNCTION SPECIFIC(PAYROLL.USERFN1,PAYROLL.USERFN3)
```

The following output is produced:

```
DSNX974I STOP FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN1
DSNX974I STOP FUNCTION SPECIFIC SUCCESSFUL FOR PAYROLL.USERFN3
```

While the STOP FUNCTION SPECIFIC command is in effect, attempts to execute the stopped functions are queued. **GUPI**

Controlling DB2 utilities

DB2 utilities are classified into two groups: online and stand-alone.

GUPI The online utilities require DB2 to be running and can be controlled in several different ways. The stand-alone utilities do not require DB2 to be up, and they can be controlled only by means of JCL. **GUPI**

Related reference

"DB2 online utilities" (DB2 Utility Guide and Reference)

"DB2 stand-alone utilities" (DB2 Utility Guide and Reference)

Starting online utilities

To start a DB2 utility, prepare an appropriate set of JCL statements for a utility job and include DB2 utility control statements in the input stream for that job.

GUPI DB2 utilities can dynamically create object lists from a pattern-matching expression and can dynamically allocate the data sets that are required to process those objects. **GUPI**

Monitoring and changing online utilities

You can monitor the status of online utilities, change parameter values of some utilities, and terminate a utility job prior to completion

GUPI Use the following commands for monitoring and changing DB2 utility jobs.

ALTER UTILITY

Alters parameter values of an active REORG or REBUILD utility.

DISPLAY UTILITY

Displays the status of utility jobs.

TERM UTILITY

Terminates a utility job before its normal completion.

If a utility is not running, you need to determine whether the type of utility access is allowed on an object of a specific status. The following table shows the

compatibility of utility types and object status.

Table 99. Compatibility of utility types and object status

Utility type	Object access
Read-only	RO
All	RW ¹
DB2	UT

Note:

1. RW is the default access type for an object.

To change the status of an object, use the ACCESS option of the START DATABASE command to start the object with a new status. For example:

```
-START DATABASE (DSN8D61A) ACCESS(RO)
```

GUIP

Related reference

"DB2 online utilities" (DB2 Utility Guide and Reference)

Controlling DB2 stand-alone utilities

You use JCL to run the DB2 stand-alone utilities, most of which can run while DB2 is running.

GUIP

To run a DB2 stand-alone utility:

1. Stop the table spaces and index spaces that are the object of the utility job. If you do not do this, you might receive inconsistent output.
2. If the utility is one that requires that DB2 be stopped during utility execution, use this command:
-STOP DB2 MODE (FORCE)
3. If the utility is one that requires that DB2 be running during utility execution and if DB2 is not running, issue this command:
-START DB2
4. Create a JCL job that includes the utility control statement with code specific data set names and associated parameters for your utility. **GUIP**

Stand-alone utilities

Some stand-alone utilities can be run only by means of JCL.

GUIP

- DSN1CHKR
- DSN1COPY
- DSN1COMP
- DSN1PRNT
- DSN1SDMP
- DSN1LOGP
- DSNJLOGF
- DSNJU003 (change log inventory)
- DSNJU004 (print log map)

Most of the stand-alone utilities can be used while DB2 is running. However, for consistency of output, the table spaces and index spaces must be stopped first because these utilities do not have access to the DB2 buffer pools. In some cases, DB2 *must* be running or stopped before you invoke the utility.

Stand-alone utility job streams require that you code specific data set names in the JCL. To determine the fifth qualifier in the data set name, you need to query the DB2 catalog tables SYSIBM.SYSTABLEPART and SYSIBM.SYSINDEXPART to determine the IPREFIX column that corresponds to the required data set.

The change log inventory utility (DSNJU003) enables you to change the contents of the bootstrap data set (BSDS). This utility cannot be run while DB2 is running because inconsistencies could result. Use the STOP DB2 MODE(QUIESCE) command to stop the DB2 subsystem, run the utility, and then restart DB2 with the START DB2 command.

The print log map utility (DSNJU004) enables you to print the bootstrap data set contents. The utility can be run when DB2 is active or inactive; however, when it is run with DB2 active, the user's JCL and the DB2 started task must both specify DISP=SHR for the BSDS data sets. 

Related reference

"DB2 stand-alone utilities" (DB2 Utility Guide and Reference)

Controlling the IRLM

The internal resource lock manager (IRLM) subsystem manages DB2 locks.

 The particular IRLM to which DB2 is connected is specified in the DB2 load module for subsystem parameters. The particular IRLM is also identified as a z/OS subsystem in the SYS1.PARMLIB member IEFSSNxx. That name is used as the IRLM procedure name (*irlmproc*) in z/OS commands.

Each IMS and DB2 subsystem must use a separate instance of IRLM.

In a data sharing environment, the IRLM handles global locking, and each DB2 member has its own corresponding IRLM.

You can use the following z/OS commands to control the IRLM. In each command description, *irlmproc* is the IRLM procedure name and *irlmmn* is the IRLM subsystem name.

MODIFY *irlmproc*,ABEND,DUMP

Abnormally terminates the IRLM and generates a dump.

MODIFY *irlmproc*,ABEND,NODUMP

Abnormally terminates the IRLM but does not generate a dump.

MODIFY *irlmproc*,DIAG

Initiates diagnostic dumps for IRLM subsystems in a data sharing group when a delay occurs.

MODIFY *irlmproc*,SET

Dynamically sets the maximum amount of private virtual (PVT) storage or the number of trace buffers that are used for this IRLM.

MODIFY *irlmproc*,STATUS

Displays the status for the subsystems on this IRLM.

START *irlmproc*
Starts the IRLM.

STOP *irlmproc*
Stops the IRLM normally.

TRACE CT,OFF,COMP=irlmnm
Stops IRLM tracing.

TRACE CT,ON,COMP=irlmnm
Starts IRLM tracing for all subtypes (DBM, SLM, XIT, and XCF).

TRACE CT,ON,COMP=irlmnm,SUB=(subname)
Starts IRLM tracing for a single subtype.

GUPI

Related reference

"z/OS IRLM commands" (DB2 Command Reference)

Related information

"IRLM names" (DB2 Data Sharing: Planning and Administration)

z/OS commands that operate on IRLM

You can use several z/OS commands to modify and monitor the IRLM connection.

GUPI

MODIFY *irlmproc,SET,PVT=nnn*
Sets the maximum amount of private virtual (PVT) storage that this IRLM can use for lock control structures.

MODIFY *irlmproc,SET,DEADLOCK=nnnn*
Sets the time for the local deadlock detection cycle.

MODIFY *irlmproc,SET,LTE=nnnn*
Sets the number of LOCK HASH entries that this IRLM can use on the next connect to the XCF LOCK structure. Use this command only for data sharing.

MODIFY *irlmproc,SET,TIMEOUT=nnnn,subsystem-name*
Sets the timeout value for the specified DB2 subsystem. Display the *subsystem-name* by using **MODIFY** *irlmproc,STATUS*.

MODIFY *irlmproc,SET,TRACE=nnn*
Sets the maximum number of trace buffers that are used for this IRLM.

MODIFY *irlmproc,STATUS,irlmnm*
Displays the status of a specific IRLM.

MODIFY *irlmproc,STATUS,ALLD*
Displays the status of all subsystems known to this IRLM in the data sharing group.

MODIFY *irlmproc,STATUS,ALLI*
Displays the status of all IRLMs known to this IRLM in the data sharing group.

MODIFY *irlmproc,STATUS,MAINT*
Displays the maintenance levels of IRLM load module CSECTs for the specified IRLM instance.

MODIFY *irlmproc*,STATUS,STOR

Displays the current and high-water allocation for private virtual (PVT) storage, as well as storage that is above the 2-GB bar.

MODIFY *irlmproc*,STATUS,TRACE

Displays information about trace types of IRLM subcomponents.

Each IMS and DB2 subsystem must use a separate instance of IRLM. 

Related reference

"z/OS IRLM commands" (DB2 Command Reference)

Starting the IRLM

The IRLM must be available when you start DB2.

 When DB2 is installed, you normally specify that the IRLM be started automatically. Then, if the IRLM is not available when DB2 is started, DB2 starts it, and periodically checks whether it is up before attempting to connect. If the attempt to start the IRLM fails, DB2 terminates.

If an automatic IRLM start has not been specified, start the IRLM before starting DB2, using the z/OS START *irlmproc* command.

When started, the IRLM issues this message to the z/OS console:

```
DXR117I irlmm INITIALIZATION COMPLETE
```

Consider starting the IRLM manually if you are having problems starting DB2 for either of these reasons:

- An IDENTIFY or CONNECT to a data sharing group fails.
- DB2 experiences a failure that involves the IRLM.

When you start the IRLM manually, you can generate a dump to collect diagnostic information because IRLM does not stop automatically. 

Stopping the IRLM

If the IRLM is started automatically by DB2, it stops automatically when DB2 is stopped. If the IRLM is not started automatically, you must stop it after DB2 stops.

 If you try to stop the IRLM while DB2 or IMS is still using it, you get the following message:

```
DXR105E irlmm STOP COMMAND REJECTED. AN IDENTIFIED SUBSYSTEM  
IS STILL ACTIVE
```

If that happens, issue the STOP *irlmproc* command again, when the subsystems are finished with the IRLM.

Alternatively, if you must stop the IRLM immediately, enter the following command to force the stop:

```
MODIFY irlmproc,ABEND,NODUMP
```

The system responds with this message:

```
DXR165I KRLM TERMINATED VIA IRLM MODIFY COMMAND.  
DXR121I KRLM END-OF-TASK CLEANUP SUCCESSFUL - HI-CSA 335K  
- HI-ACCT-CSA 0K
```

DB2 abends. An IMS subsystem that uses the IRLM does not abend and can be reconnected.

IRLM uses the z/OS Automatic Restart Manager (ARM) services. However, it de-registers from ARM for normal shutdowns. IRLM registers with ARM during initialization and provides ARM with an event exit routine. The event exit routine must be in the link list. It is part of the IRLM DXRRL183 load module. The event exit routine ensures that the IRLM name is defined to z/OS when ARM restarts IRLM on a target z/OS system that is different from the failing z/OS system. The IRLM element name that is used for the ARM registration depends on the IRLM mode. For local-mode IRLM, the element name is a concatenation of the IRLM subsystem name and the IRLM ID. For global mode IRLM, the element name is a concatenation of the IRLM data sharing group name, IRLM subsystem name, and the IRLM ID.

IRLM de-registers from ARM when one of the following events occurs:

- PURGE *irlmproc* is issued.
- MODIFY *irlmproc,ABEND,NODUMP* is issued.
- DB2 automatically stops IRLM.

The command MODIFY *irlmproc,ABEND,NODUMP* specifies that IRLM de-register from ARM before terminating, which prevents ARM from restarting IRLM. However, this command does not prevent ARM from restarting DB2, and, if you set the automatic restart manager to restart IRLM, DB2 automatically starts IRLM.

◀ GUPI

Monitoring threads

You monitor threads by using the DB2 command DISPLAY THREAD, which displays current information about the status of threads.

◀ GUPI

The DISPLAY THREAD command displays information about:

- Threads that are processing locally
- Threads that are processing distributed requests
- Stored procedures or user-defined functions that the thread is executing
- Parallel tasks

◀ GUPI

Types of threads

Threads can be active or pooled and an active thread can be an allied thread or a database access thread.

◀ GUPI

Active allied thread

A thread that is connected to DB2 from TSO, batch, IMS, CICS, CAF, or RRSAA.

Active database access thread

A thread that is connected through a network with another system and performing work on behalf of that system.

Pooled database access thread

An idle thread that is waiting for a new unit of work from a connection to another system so that it can begin. Pooled threads hold no database locks.

GUPI

Output of the DISPLAY THREAD command

The output of the command DISPLAY THREAD can also indicate that a system quiesce is in effect as a result of the ARCHIVE LOG command.

The command DISPLAY THREAD allows you to select which type of information you want to include in the display by using one or more of the following:

- Active, indoubt, postponed-abort, or pooled threads
- Allied threads that are associated with the address spaces whose connection-names are specified
- Allied threads
- Distributed threads
- Distributed threads that are associated with a specific remote location
- Detailed information about connections with remote locations
- A specific logical unit of work ID (LUWID)

The information that is returned by the DISPLAY THREAD command reflects a dynamic status. By the time the information is displayed, the status might have changed. Moreover, the information is consistent only within one address space and is not necessarily consistent across all address spaces.

To use the TYPE, LOCATION, DETAIL, and LUWID keywords, you must have SYSOPR authority or higher.

More information about how to interpret this output can be found in the topics describing the individual connections and in the description of message DSNV408I.

Related tasks

"Archiving the log" on page 489

Related reference

"-DISPLAY THREAD (DB2)" (DB2 Command Reference)

"DSNV408I" (DB2 Messages)

Displaying information about threads

Use DISPLAY THREAD TYPE(INDOUBT) to find information about allied and database access indoubt threads. This command provides information about threads where DB2 is a participant, a coordinator, or both.

GUPI

The TYPE(INDOUBT) option tells you which systems still need indoubt resolution and provides the LUWIDs that you need to recover indoubt threads. A thread that has completed phase 1 of commit and still has a connection with its coordinator is in the *prepared* state and is displayed as part of the DISPLAY THREAD active report. If a prepared thread loses its connection with its coordinator, it enters the *indoubt* state and terminates its connections to any participants at that time. Any threads that are in the prepared or indoubt state when DB2 terminates are indoubt after DB2 restart. However, if the participant system is waiting for a commit or rollback decision from the coordinator, and the connection is still active, DB2 considers the thread active. **GUPI**

If a thread is indoubt at a participant, you can determine whether a commit or abort decision was made at the coordinator by issuing the DISPLAY THREAD command at the coordinator as described previously. If an indoubt thread appears at one system and does not appear at the other system, the latter system backed out the thread, and the first system must therefore do the same.

Related concepts

“Output of the DISPLAY THREAD command” on page 432

Displaying information by location

Use the LOCATION keyword, followed by a list of location names, to display thread information for particular locations.

GUIP You can use an asterisk (*) after the THD and LOCATION keywords. For example, enter:

```
-DISPLAY THREAD(*) LOCATION(*) DETAIL
```

DB2 returns messages like these:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME      ST 1 A 2  REQ ID      AUTHID  PLAN    ASID TOKEN
SERVER    RA *          2923 DB2BP    ADMF001 DISTSERV 0036  20 3
V437-WORKSTATION=ARRAKIS, USERID=ADMF001,
APPLICATION NAME=DB2BP
V436-PGM=NULLID.SQLC27A4, SEC=201, STMT=210
V445-09707265.01BE.889C28200037=203 ACCESSING DATA FOR
( 1)2002:91E:610:1::5 4
V447-INDEX SESSID              A ST TIME
V448-( 1) 446:1300 5          W S2 9802812045091
DISPLAY ACTIVE REPORT COMPLETE
DSNV9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

Key Description

- 1** The ST (status) column contains characters that indicate the connection status of the local site. The *TR* indicates that an allied, distributed thread has been established. The *RA* indicates that a distributed thread has been established and is in receive mode. The *RD* indicates that a distributed thread is performing a remote access on behalf of another location (R) and is performing an operation that involves DCE services (D). Currently, DB2 supports the optional use of DCE services to authenticate remote users.
- 2** The A (active) column contains an asterisk that indicates that the thread is active within DB2. It is blank when the thread is inactive within DB2 (active or waiting within the application).
- 3** This LUWID is unique across all connected systems. This thread has a token of 20 (it appears in two places in the display output).
- 4** This is the location of the partner. If the RDBNAME is not known, the location column contains either an SNA LUNAME or IP address.
- 5** If the connection uses TCP/IP, the SESSID column contains "local:remote", where *local* specifies the DB2 TCP/IP port number and *remote* specifies the partner's TCP/IP port number.

GUIP

Related reference

"DSNV444I" (DB2 Messages)

"DSNV446I" (DB2 Messages)

"DSNV404I" (DB2 Messages)

Displaying information for non-DB2 locations

Because DB2 does not receive a location name from non-DB2 locations, you must enter the LUNAME or IP address of the location for which you want to display information.

GUIP The LUNAME is enclosed by the less-than (<) and greater-than (>) symbols. The IP address can be in the dotted decimal or colon hexadecimal format. For example, if you wanted to display information about a non-DB2 DBMS with the LUNAME of LUSFOS2, you would enter the following command:

```
-DISPLAY THREAD (*) LOCATION (<LUSFOS2>)
```

DB2 uses the one of the following formats in messages that display information about non-DB2 requesters:

- LUNAME notation
- Dotted decimal format
- Colon hexadecimal format

GUIP

Displaying conversation-level information about threads

Use the DETAIL keyword with the LOCATION keyword to give you information about conversation activity when distribution information is displayed for active threads.

GUIP This keyword has no effect on the display of indoubt threads.

For example, issue:

```
-DISPLAY THREAD(*) LOCATION(*) DETAIL
```

DB2 returns the following messages, which indicate that the local site application is waiting for a conversation to be allocated in DB2, and a DB2 server that is accessed by a DRDA client using TCP/IP.

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME      ST A  REQ ID          AUTHID  PLAN  ASID  TOKEN
TSO       TR *   3 SYSADM          SYSADM  DSNESPRR 002E      2
V436-PGM=DSNESPRR.DSNESM68, SEC=1, STMT=116
V444-DB2NET.LUND0.A238216C2FAE=2 ACCESSING DATA AT
( 1)USIBMSTODB22-LUND1
V447--INDEX SESSID          A ST TIME
V448--( 1) 0000000000000000 N 1 A1 2 9015816504776
TSO       RA *   11 SYSADM          SYSADM  DSNESPRR 001A      15
V445-STLDRIV.SSLU.A23555366A29=15 ACCESSING DATA FOR
( 1)123.34.101.98
V447--INDEX SESSID          A ST TIME
V448--( 1) 446:3171 3 S2 9015611253108
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

Key Description

- 1** The information on this line is part of message DSNV447I. The conversation A (active) column for the server is useful in determining when a DB2 thread is hung and whether processing is waiting in the

network stack (VTAM or TCP/IP) or in DB2. A value of W indicates that the thread is suspended in DB2 and is waiting for notification from the network that the event has completed. A value of N indicates that control of the conversation is in the network stack.

2 The information on this line is part of message DSNV448I. The A in the conversation ST (status) column for a serving site indicates that a conversation is being allocated in DB2. The 1 indicates that the thread uses DB2 private protocol access. A 2 would indicate DRDA access. An R in the status column would indicate that the conversation is receiving or waiting to receive a request or reply. An S in this column for a server indicates that the application is sending or preparing to send a request or reply.

3 The information on this line is part of message DSNV448I. The SESSID column has changed as follows. If the connection uses VTAM, the SESSID column contains a VTAM session identifier. If the connection uses TCP/IP, the SESSID column contains "local:remote", where *local* specifies the DB2 TCP/IP port number and *remote* specifies the partner's TCP/IP port number.

GUPI

Related reference

"DSNV447I" (DB2 Messages)

"DSNV448I" (DB2 Messages)

"-DISPLAY THREAD (DB2)" (DB2 Command Reference)

Displaying threads by LUWID

Use the LUWID optional keyword, which is only valid when DDF is started, to display threads by logical unit of work identifiers. The LUWIDs are assigned to the thread by the site that originated the thread.

GUPI

You can use an asterisk (*) in an LUWID as in a LOCATION name. For example, use DISPLAY THREAD TYPE(INDOUBT) LUWID(NET1.*) to display all the indoubt threads whose LUWID has a network name of NET1. The command DISPLAY THREAD TYPE(INDOUBT) LUWID(IBM.NEW*) displays all indoubt threads whose LUWID has a network name of "IBM" and whose LUNAME begins with "NEW."

Use the DETAIL keyword with the DISPLAY THREAD LUWID command to show the status of every conversation that is connected to each displayed thread and to indicate whether a conversation is using DRDA access or DB2 private protocol access.

To issue this command, enter the following command:

```
-DISPLAY THREAD(*) LUWID (luwid) DETAIL
```

DB2 returns the following message:

```
-DISPLAY THREAD(*) LUWID (luwid) DET
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME      ST A  REQ ID          AUTHID  PLAN    ASID  TOKEN
BATCH    TR      5 TC3923S0      SYSADM  TC392   0000   2
V436-PGM=*.TC3923S0, SEC=1, STMT=116
V444-DB2NET.LUNSITE0.A11A7D7B2057=2 1 ACCESSING DATA AT
( 1)USIBMSTODB22-LUNSITE1
V447--INDEX SESSID          A ST TIME
```

```
V448--( 1) 00C3F4228C5A244C S2 2 8929612225354
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

Key Description

- 1** In the preceding display output, you can see that the LUWID has been assigned a token of 2. You can use this token instead of the long version of the LUWID to cancel or display the given thread. For example:
-DISPLAY THREAD(*) LUWID(2) DET
- 2** In addition, the status column for the serving site contains a value of S2. The S means that this thread can send a request or response, and the 2 means that this is an DRDA access conversation.

GUIP

Displaying threads by type

Use the TYPE keyword with the DISPLAY THREAD command to request the type of thread to display.

GUIP

For example, you can show only active threads that are executing a stored procedure or user-defined function by issuing the following command:

```
-DISPLAY THREAD(*) TYPE(PROC)
```

GUIP

Monitoring all DBMSs in a transaction

The DETAIL keyword of the command DISPLAY THREAD allows you to monitor all of the requesting and serving DBMSs that are involved in a transaction.

GUIP

For example, you could monitor an application that runs at USIBMSTODB21 requesting information from USIBMSTODB22, which must establish conversations with secondary servers USIBMSTODB23 and USIBMSTODB24 to provide the requested information. The following figure depicts such an example. In this example, ADA refers to DRDA access, and SDA refers to DB2 private protocol access. USIBMSTODB21 is considered to be upstream from USIBMSTODB22. USIBMSTODB22 is considered to be upstream from USIBMSTODB23. Conversely, USIBMSTODB23 and USIBMSTODB24 are downstream from USIBMSTODB22 and USIBMSTODB21 respectively.

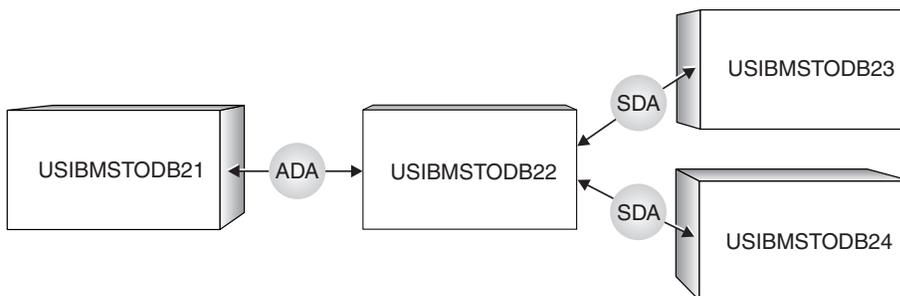


Figure 33. Example of a DB2 transaction involving four sites

The application that runs at USIBMSTODB21 is connected to a server at USIBMSTODB22, using DRDA access. If you enter the DISPLAY THREAD

command with the DETAIL keyword from USIBMSTODB21, you receive the output that the following figure depicts.

```
-DISPLAY THREAD(*) LOC(*) DET
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME      ST A   REQ ID          AUTHID PLAN      ASID  TOKEN
BATCH    TR *    6 BKH2C          SYSADM YW1019C  0009    2
V436-PGM=BKH2C.BKH2C, SEC=1, STMNT=4
V444-USIBMSY.SSLU.A23555366A29=2 ACCESSING DATA AT
( 1)USIBMSTODB22-SSURLU
V447--INDEX SESSID          A ST TIME
V448--( 1) 0000000300000004 N R2 9015611253116
DISPLAY ACTIVE REPORT COMPLETE
11:26:23 DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

Figure 34. DISPLAY THREAD DETAIL output from USIBMSTODB21

This output indicates that the application is waiting for data to be returned by the server at USIBMSTODB22.

The server at USIBMSTODB22 is running a package on behalf of the application at USIBMSTODB21, in order to access data at USIBMSTODB23 and USIBMSTODB24 by DB2 private protocol access. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB22, you receive the output that the following figure depicts.

```
|
| -DISPLAY THREAD(*) LOC(*) DET
| DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
| DSNV402I - ACTIVE THREADS -
| NAME      ST A   REQ ID          AUTHID PLAN      ASID  TOKEN
| BATCH    RA *    0 BKH2C          SYSADM YW1019C  0008    2
| V436-PGM=BKH2C.BKH2C, SEC=1, STMNT=4
| V445-STLDRIV.SSLU.A23555366A29=2 ACCESSING DATA FOR
| ( 1)USIBMSTODB21-SSLU
| V444-STLDRIV.SSLU.A23555366A29=2 ACCESSING DATA AT
| ( 2)USIBMSTODB23-OSSLU
| ( 3)USIBMSTODB24-OSSURLU
| V447--INDEX SESSID          A ST TIME
| V448--( 1) 0000000300000004 S2 9015611253108
| V448--( 2) 0000000600000002 S1 9015611253077
| V448--( 3) 0000000900000005 N R1 9015611253907
| DISPLAY ACTIVE REPORT COMPLETE
| 11:26:34 DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

Figure 35. DISPLAY THREAD DETAIL output from USIBMSTODB22

This output indicates that the server at USIBMSTODB22 is waiting for data to be returned by the secondary server at USIBMSTODB24.

The secondary server at USIBMSTODB23 is accessing data for the primary server at USIBMSTODB22. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB23, you receive the output that the following figure depicts.

```

-DISPLAY THREAD(*) LOC(*) DET
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME      ST A   REQ ID           AUTHID PLAN      ASID  TOKEN
BATCH    RA *    2 BKH2C           SYSADM YW1019C  0006   1
V445-STLDRIV.SSLU.A23555366A29=1 ACCESSING DATA FOR
( 1)USIBMSTODB22-SSURLU
V447--INDEX SESSID           A ST TIME
V448--( 1) 0000000600000002 W R1 9015611252369
DISPLAY ACTIVE REPORT COMPLETE
11:27:25 DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION

```

Figure 36. DISPLAY THREAD DETAIL output from USIBMSTODB23

This output indicates that the secondary server at USIBMSTODB23 is not currently active.

The secondary server at USIBMSTODB24 is also accessing data for the primary server at USIBMSTODB22. If you enter the DISPLAY THREAD command with the DETAIL keyword from USIBMSTODB24, you receive the output that the following figure depicts:

```

-DISPLAY THREAD(*) LOC(*) DET
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME      ST A   REQ ID           AUTHID PLAN      ASID  TOKEN
BATCH    RA *    2 BKH2C           SYSADM YW1019C  0006   1
V436-PGM=*.BKH2C, SEC=1, STMNT=1
V445-STLDRIV.SSLU.A23555366A29=1 ACCESSING DATA FOR
( 1)USIBMSTODB22-SSURLU
V447--INDEX SESSID           A ST TIME
V448--( 1) 0000000900000005 S1 9015611253075
DISPLAY ACTIVE REPORT COMPLETE
11:27:32 DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION

```

Figure 37. DISPLAY THREAD DETAIL output from USIBMSTODB24

This output indicates that the secondary server at USIBMSTODB24 is currently active.

The conversation status might not change for a long time. The conversation could be hung, or the processing could just be taking a long time. To determine whether the conversation is hung, issue DISPLAY THREAD again and compare the new timestamp to the timestamps from previous output messages. If the timestamp is changing, but the status is not changing, the job is still processing. If you need to terminate a distributed job, perhaps because it is hung and has been holding database locks for a long period of time, you can use the CANCEL DDF THREAD command if the thread is in DB2 (whether active or suspended) or the VARY NET

TERM command if the thread is within VTAM. 

Related tasks

“Canceling threads” on page 466

Controlling connections

The method that you use to control connections between DB2 and another subsystem or environment depends on the subsystem or environment that is involved.

Controlling TSO connections

z/OS provides no commands for controlling or monitoring a connection to DB2.

GUPI The connection is monitored instead by the DB2 command `DISPLAY THREAD`, which displays information about connections to DB2 (from other subsystems as well as from z/OS).

The command is generally entered from a z/OS console or an administrator's TSO session. **GUPI**

Related tasks

"Monitoring threads" on page 431

Connecting to DB2 from TSO

The z/OS operator is not involved in starting and stopping TSO connections. Those connections are made through the DSN command processor.

GUPI The DSN command processor is invoked in one of these ways:

- Explicitly, by the DSN command
- Implicitly, through DB2I (DB2 Interactive)

When a DSN session is active, you can enter DSN subcommands, DB2 commands, and TSO commands.

The DSN command can be issued in the foreground or background, when running under the TSO terminal monitor program (TMP). The full syntax of the command is:

```
DSN SYSTEM (subsystemid) RETRY (n1) TEST (n2)
```

The parameters are optional, and have the following meanings:

subsystemid

Is the subsystem ID of the DB2 subsystem to be connected.

n1 Is the number of times to attempt the connection if DB2 is not running (one attempt every 30 seconds).

n2 Is the DSN tracing system control that can be used if a problem is suspected.

For example, this invokes a DSN session, requesting five retries at 30-second intervals:

```
DSN SYSTEM (DB2) RETRY (5)
```

To make an implicit connection, invoke a DSN session by selecting any of these operations:

- SQL statements using SPUFI
- DCLGEN
- BIND/REBIND/FREE
- RUN
- DB2 commands
- Program preparation and execution

In carrying out those operations, the DB2I panels invoke CLISTs, which start the DSN session and invoke appropriate subcommands. **GUPI**

Related tasks

“Running TSO application programs” on page 369

Monitoring TSO and CAF connections

To display information about connections that use the TSO attach facility and call attach facility (CAF), issue the command `DISPLAY THREAD`.

GUPI The following table summarizes how `DISPLAY THREAD` output differs for a TSO online application, a TSO batch application, a QMF session, and a call attach facility application.

Table 100. Differences in `DISPLAY THREAD` information for different environments

Connection	Name	AUTHID	Corr-ID ¹	Plan ¹
DSN (TSO Online)	TSO	Logon ID	Logon ID	RUN .. Plan(x)
DSN (TSO Batch)	BATCH	Job USER=	Job Name	RUN .. Plan(x)
QMF	DB2CALL	Logon ID	Logon ID	'QMFvr0'
CAF	DB2CALL	Logon ID	Logon ID	OPEN parm

Notes:

1. After the application connects to DB2 but before a plan is allocated, this field is blank.

The name of the connection can have one of the following values:

Name Connection to

TSO Program that runs in TSO foreground

BATCH

Program that runs in TSO background

DB2CALL

Program that uses the call attachment facility and that runs in the same address space as a program that uses the TSO attachment facility

The correlation ID, *corr-id*, is either the foreground authorization ID or the background job name.

The following command displays information about TSO and CAF threads, including those threads that process requests to or from remote locations:

```
-DISPLAY THREAD(BATCH,TSO,DB2CALL)
```

```

DSNV401I = DISPLAY THREAD REPORT FOLLOWS -
DSNV402I = ACTIVE THREADS -
NAME      ST A   REQ ID          AUTHID  PLAN      ASID TOKEN
1 BATCH   T *   2997 TEP2           SYSADM  DSNTEP41 0019 18818
2 BATCH   RA *  1246 BINETEP2    SYSADM  DSNTEP44 0022 20556
V445-DB2NET.LUND1.AB0C8FB44C4D=20556 ACCESSING DATA FOR SAN_JOSE
3 TSO     T     12 SYSADM        SYSADM  DSNESPRR 0028 5570
4 DB2CALL T *  18472 CAFCOB2     SYSADM  CAFCOB2  001A 24979
5 BATCH   T *    1 PUPPY      SYSADM  DSNTEP51 0025 20499
6         PT *   641 PUPPY      SYSADM  DSNTEP51 002D 20500
7         PT *   592 PUPPY      SYSADM  DSNTEP51 002D 20501
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I = DSNVDT '-DIS THREAD' NORMAL COMPLETION

```

Key Description

- 1 This is a TSO batch application.
- 2 This is a TSO batch application running at a remote location and accessing tables at this location.
- 3 This is a TSO online application.
- 4 This is a call attachment facility application.
- 5 This is an originating thread for a TSO batch application.
- 6 This is a parallel thread for the originating TSO batch application thread.
- 7 This is a parallel thread for the originating TSO batch application thread.

Figure 38. DISPLAY THREAD output that shows TSO and CAF connections

GUPI

Related tasks

“Monitoring threads” on page 431

Related reference

“DSNV404I” (DB2 Messages)

Disconnecting from DB2 while under TSO

Several conditions cause the TSO connection to DB2 to end.

GUPI

The connection to DB2 ends, and the thread is terminated, when:

- You enter the END subcommand.
- You enter DSN again. (A new connection is established immediately.)
- You enter the CANCEL THREAD command (for threads that are active or suspended in DB2).
- You enter the MVS CANCEL command.
- You press the attention key (PA1).
- Any of the following operations end, or you enter END or RETURN when using any of them:
 - SQL statements using SPUFI
 - DCLGEN
 - BIND/REBIND/FREE
 - RUN

For example, the following command and subcommands establish a connection to DB2, run a program, and terminate the connection:

TSO displays:

READY

You enter:

DSN SYSTEM (DSN)

DSN displays:

DSN

You enter:

RUN PROGRAM (MYPROG)

DSN displays:

DSN

You enter:

END

TSO displays:

READY



Controlling CICS connections

Certain CICS attachment facility commands can be entered from a CICS terminal to control and monitor connections between CICS and DB2.



DSNC DISCONNECT

Terminates threads using a specific DB2 plan.

DSNC DISPLAY

Displays thread information or statistics.

DSNC MODIFY

Modifies the maximum number of threads for a transaction or group.

DSNC STOP

Disconnects CICS from DB2.

DSNC STRT

Starts the CICS attachment facility.

CICS command responses are sent to the terminal from which the corresponding command was entered, unless the DSNC DISPLAY command specifies an alternative destination. For details on specifying alternate destinations for output, see the DSNC DISPLAY command in the *DB2 Command Reference*. 

Related information

 Overview: How you can define the CICS DB2 connection

Connecting from CICS

You can start a connection to DB2 at any time after CICS initialization by using the CICS attachment facility which is a set of modules DB2 provides that are loaded into the CICS address space.

GUPI To connect to DB2, issue the following command to start the attachment facility:

```
DSNC STRT ssid
```

For *ssid*, specify a DB2 subsystem ID to override the value that is specified in the CICS INITPARM macro.

You can also start the attachment facility automatically at CICS initialization by using a program list table (PLT). **GUPI**

Restarting CICS

One function of the CICS attachment facility is to keep data in synchronization between the two systems.

GUPI If DB2 completes phase 1 but does not start phase 2 of the commit process, the units of recovery that are being committed are termed *indoubt*. An indoubt unit of recovery might occur if DB2 terminates abnormally after completing phase 1 of the commit process. CICS might commit or roll back work without DB2 knowing about it.

DB2 cannot resolve those indoubt units of recovery (that is, commit or roll back the changes made to DB2 resources) until the connection to CICS is restarted. This means that CICS should always be auto-started (START=AUTO in the DFHSIT table) obtain all necessary information for indoubt thread resolution that is available from its log. Do not perform a cold start. The START option can be specified in the DFHSIT table, as described in *CICS Transaction Server for z/OS Resource Definition Guide*.

If CICS has requests active in DB2 when a DB2 connection terminates, the corresponding CICS tasks might remain suspended even after CICS is reconnected to DB2. Purge those tasks from CICS using a CICS-supplied transaction such as:

```
CEMT SET TASK(nn) FORCE
```

See *CICS Transaction Server for z/OS CICS Supplied Transactions* for more information about transactions that CICS supplies.

If any unit of work is indoubt when the failure occurs, the CICS attachment facility automatically attempts to resolve the unit of work when CICS is reconnected to DB2. Under some circumstances, however, CICS cannot resolve indoubt units of recovery. You have to manually recover these indoubt units of recovery. **GUPI**

Related tasks

“Recovering CICS-DB2 indoubt units of recovery” on page 444

Defining CICS threads

Every CICS transaction that accesses DB2 requires a thread to service the DB2 requests. Each thread uses one z/OS subtask to execute DB2 code for the CICS application.

GUPI The DSNC STRT command starts the CICS DB2 attachment facility, which allows CICS application programs to access DB2 databases.

Threads are created at the first DB2 request from the application if one is not already available for the specific DB2 plan.

For complete information about defining CICS threads with DB2, see *CICS DB2 Guide*. 

Monitoring CICS threads

No operator intervention is required for connecting applications; CICS handles the threads dynamically. You can monitor threads using CICS attachment facility commands or DB2 commands.

 Any authorized CICS user can monitor the threads and change the connection parameters as needed. Operators can use the following CICS attachment facility commands to monitor the threads:

```
DSNC DISPLAY PLAN plan-name destination
DSNC DISPLAY TRANSACTION transaction-id destination
```

These commands display the threads that the resource or transaction is using. The following information is provided for each created thread:

- Authorization ID for the plan that is associated with the transaction (8 characters).
- PLAN/TRAN name (8 characters).
- A or I (one character).
If **A** is displayed, the thread is within a unit of work. If **I** is displayed, the thread is waiting for a unit of work, and the authorization ID is blank.

The following CICS attachment facility command is used to monitor CICS:
DSNC DISPLAY STATISTICS *destination*



Displaying CICS-DB2 indoubt units of recovery:

You can display a list of CICS-DB2 indoubt units of recovery.



To display a list of indoubt units of recovery, issue the command:
-DISPLAY THREAD (*connection-name*) TYPE (INDOUBT)

The command produces messages similar to these:

```
DSNV407I -STR INDOUBT THREADS - 480
COORDINATOR          STATUS      RESET URID          AUTHID
CICS41                INDOUBT        00019B8ADE9E      ADMF001
  V449-HAS NID= CICS41.AACC9B739F125184 AND ID=GT00LE39
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```



Related reference

"DSNV408I" (DB2 Messages)

Recovering CICS-DB2 indoubt units of recovery:

You can recover CICS-DB2 indoubt units of recovery.

GUPI To recover an indoubt unit of recovery, issue one of the following commands:

```
-RECOVER INDOUBT (connection-name) ACTION (COMMIT) ID (correlation-id)  
-RECOVER INDOUBT (connection-name) ACTION (ABORT) ID (correlation-id)
```

The default value for *connection-name* is the connection name from which you entered the command. The *correlation-id* is the correlation ID of the thread to be recovered. You can determine the correlation ID by issuing the command DISPLAY THREAD. Your choice for the ACTION parameter indicates whether to commit or roll back the associated unit of recovery.

One of the following messages might be issued after you use the RECOVER command:

```
DSNV414I - THREAD correlation-id COMMIT SCHEDULED  
DSNV415I - THREAD correlation-id ABORT SCHEDULED
```

GUPI

Related troubleshooting information

“Recovering CICS indoubt units of recovery” on page 608

Related concepts

“Multiple system consistency” on page 515

Related tasks

“Resolving indoubt units of recovery” on page 522

Displaying CICS postponed units of recovery:

You can display a list of CICS postponed units of recovery.

GUPI

To display a list of postponed units of recovery, issue the command:

```
-DISPLAY THREAD (connection-name) TYPE (POSTPONED)
```

The command produces messages similar to these:

```
DSNV431I -POSTPONED ABORT THREADS - 480  
COORDINATOR          STATUS      RESET URID          AUTHID  
CICS41                P-ABORT          00019B8ADE9E      ADMF001  
V449-HAS NID= CICS41.AACC9B739F125184 AND ID=GT00LE39  
DISPLAY POSTPONED ABORT REPORT COMPLETE  
DSN9022I -STR DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

GUPI

Related reference

“DSNV408I” (DB2 Messages)

Disconnecting CICS applications

You cannot disconnect a particular CICS transaction from DB2 without abnormally terminating the transaction.

GUPI

To disconnect a CICS application from DB2:

- The DB2 command CANCEL THREAD can be used to cancel a particular thread. CANCEL THREAD requires that you know the *token* for any thread that

you want to cancel. Enter the following command to cancel the thread that is identified by the token as indicated in the display output.

```
-CANCEL THREAD(46)
```

When you issue CANCEL THREAD for a thread, that thread is scheduled to be terminated in DB2.

- The command DSNB DISCONNECT terminates the threads allocated to a plan ID, but it does not prevent new threads from being created. This command frees DB2 resources that are shared by the CICS transactions and allows exclusive access to them for special-purpose processes such as utilities or data definition statements.

The thread is not canceled until the application releases it for reuse, either at SYNCPOINT or end-of-task.

For complete information about the use of CICS attachment commands with DB2, see *CICS DB2 Guide*. 

Disconnecting from CICS

This section describes how to do both an orderly and forced disconnection of the attachment to CICS.

Performing an orderly termination from CICS:

Orderly termination is recommended whenever possible. An orderly termination of the connection allows each CICS transaction to terminate before thread subtasks are detached. This means that no indoubt units of recovery should exist at reconnection time.

 To perform an orderly termination, use one of the following methods:

- Enter the DSNB STOP QUIESCE command. CICS and DB2 remain active. For example, the following command stops the DB2 subsystem (QUIESCE) allows the currently identified tasks to continue normal execution, and does not allow new tasks to identify themselves to DB2:

```
-STOP DB2 MODE (QUIESCE)
```

This message appears when the stop process starts and frees the entering terminal (option QUIESCE):

```
DSNB012I THE ATTACHMENT FACILITY STOP QUIESCE IS PROCEEDING
```

When the stop process ends and the connection is terminated, this message is added to the output from the CICS job:

```
DSNB025I THE ATTACHMENT FACILITY IS INACTIVE
```

- Enter the CICS command CEMT PERFORM SHUTDOWN. During program list table (PLT) processing, the CICS attachment facility is also named to shut down. DB2 remains active. For information about the CEMT PERFORM SHUTDOWN command, see *CICS for MVS/ESA CICS-Supplied Transactions*.
- Enter the DB2 command CANCEL THREAD. The thread terminates abnormally.



Performing a forced termination from CICS:

Although forced termination is not recommended, at times you might need to force the connection to end.

GUPI A forced termination of the connection can abnormally terminate CICS transactions that are connected to DB2. Therefore, indoubt units of recovery can exist at reconnect.

A forced termination occurs under the following circumstances:

- You enter the DSNC STOP FORCE command. This command waits 15 seconds before detaching the thread subtasks and in some cases can achieve an orderly termination.

This message appears when the stop process starts and frees the entering terminal (option FORCE):

```
DSNC022I THE ATTACHMENT FACILITY STOP FORCE IS PROCEEDING
```

DB2 and CICS remain active.

- You enter the CICS command CEMT PERFORM SHUTDOWN IMMEDIATE. For information about this command, see *CICS for MVS/ESA CICS-Supplied Transactions*. DB2 remains active.
- You enter the DB2 command STOP DB2 MODE (FORCE). CICS remains active.
- A DB2 abend occurs. CICS remains active.
- A CICS abend occurs. DB2 remains active.
- STOP is issued to the DB2 or CICS attachment facility. The CICS transaction overflows to the pool. The transaction issues an intermediate commit. The thread is terminated at commit time, and further DB2 access is not allowed.

When the stop process ends and the connection is terminated, this message is added to the output from the CICS job:

```
DSNC025I THE ATTACHMENT FACILITY IS INACTIVE
```

GUPI

Controlling IMS connections

You use IMS operator commands to control and monitor the connection to DB2.

/START SUBSYS

Connects the IMS control region to a DB2 subsystem.

/TRACE

Controls the IMS trace.

/DISPLAY SUBSYS

Displays connection status and thread activity.

/DISPLAY OASN SUBSYS

Displays outstanding units of recovery.

/CHANGE SUBSYS

Deletes an indoubt unit of recovery from IMS.

/STOP SUBSYS

Disconnects IMS from a DB2 subsystem.

IMS command responses are sent to the terminal from which the corresponding command was entered. Authorization to enter IMS commands is based on IMS security.

Related reference

"IMS commands" (DB2 Command Reference)

Connections to the IMS control region

IMS makes one connection to its control region from each DB2 subsystem. IMS can make the connection either automatically or in response to a command.

Connections are made in the following ways:

- Automatically during IMS cold start initialization or at warm start of IMS if a DB2 connection was active when IMS is shut down.
- In response to the `/START SUBSYS ssid` command, where *ssid* is the DB2 subsystem identifier.

The command causes the following message to be displayed at the logical terminal (LTERM):

```
DFS058  START COMMAND COMPLETED
```

The message is issued regardless of whether DB2 is active and does not imply that the connection is established.

The order of starting IMS and DB2 is not vital. If IMS is started first, when DB2 comes up, DB2 posts the control region MODIFY task, and IMS again tries to reconnect.

If DB2 is stopped by the STOP DB2 command, the `/STOP SUBSYS` command, or a DB2 abend, IMS cannot reconnect automatically. You must make the connection by using the `/START SUBSYS` command.

The following messages can be produced when IMS attempts to connect a DB2 subsystem. In each message, *imsid* is the IMS connection name.

- If DB2 is active, these messages are sent:
 - To the z/OS console:

```
DFS3613I ESS TCB INITIALIZATION COMPLETE
```
 - To the IMS master terminal:

```
DSNM001I IMS/TM imsid CONNECTED TO SUBSYSTEM ssnm
```

- If DB2 is not active, this message is sent to the IMS master terminal:

```
DSNM003I IMS/TM imsid FAILED TO CONNECT TO SUBSYSTEM ssnm  
RC=00 imsid
```

RC=00 means that a notify request has been queued. When DB2 starts, IMS is also notified.

No message goes to the z/OS console.

IMS thread attachment

For IMS connections, a *thread* is a structure that describes an IMS application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services

Execution of the first SQL statement of the program causes the IMS attachment facility to create a thread and allocate a plan, whose name is associated with the IMS application program module name. DB2 sets up control blocks for the thread and loads the plan.

Duplicate IMS correlation IDs

Under certain circumstances, two threads can have the same correlation ID.



Two threads can have the same correlation ID (*pst#.psbname*) if all of these conditions occur:

- Connections have been broken several times.
- Indoubt units of recovery were not recovered.
- Applications were subsequently scheduled in the same region.

To uniquely identify threads which have the same correlation ID (*pst#.psbname*) requires that you be able to identify and understand the network ID (NID). For connections with IMS, you should also be able to identify and understand the IMS originating sequence number (OASN).

The NID is shown in a condensed form on the messages that are issued by the DB2 DISPLAY THREAD command processor. The IMS subsystem name (*imsid*) is displayed as the *net_node*. The *net_node* is followed by the 8-byte OASN, which is displayed in hexadecimal format (16 characters), with all leading zeros omitted. The *net_node* and the OASN are separated by a period.

For example, if the *net_node* is IMSA, and the OASN is 0003CA670000006E, the NID is displayed as IMSA.3CA670000006E on the DB2 DISPLAY THREAD command output.

If two threads have the same *corr-id*, use the NID instead of *corr-id* on the RECOVER INDOUBT command. The NID uniquely identifies the work unit.

The OASN is a 4-byte number that represents the number of IMS scheduling since the last IMS cold start. The OASN is occasionally found in an 8-byte format, where the first 4 bytes contain the scheduling number, and the last 4 bytes contain the number of IMS sync points (commits) during this schedule. The OASN is part of the NID.

The NID is a 16-byte network ID that originates from IMS. The NID contains the 4-byte IMS subsystem name, followed by four bytes of blanks, followed by the 8-byte version of the OASN. In communications between IMS and DB2, the NID serves as the recovery token. 

Displaying IMS attachment facility threads

You use the DB2 command DISPLAY THREAD to display IMS attachment facility threads.

 DISPLAY THREAD output for DB2 connections to IMS differs depending on whether DB2 is connected to a DL/I batch program, a control region, a message-driven program, or a non-message-driven program. The following table summarizes these differences.

Table 101. Differences in DISPLAY THREAD information for IMS connections

Connection	Name	AUTHID ²	ID ^{1,2}	Plan ^{1,2}
DL/I batch	DDITV02 statement	JOBUSER=	Job Name	DDITV02 statement
Control region	IMSID	N/A	N/A	N/A
Message driven	IMSID	Signon ID or ltermid	PST+ PSB	RTT or program
Non-message driven	IMSID	AXBUSER or PSBNAME	PST+ PSB	RTT or program

Table 101. Differences in DISPLAY THREAD information for IMS connections (continued)

Connection	Name	AUTHID ²	ID ^{1,2}	Plan ^{1,2}
------------	------	---------------------	-------------------	---------------------

Notes:

1. After the application connects to DB2 but before sign-on processing completes, this field is blank.
2. After sign-on processing completes but before a plan is allocated, this field is blank.

The following command displays information about IMS threads, including those accessing data at remote locations:

```
-DISPLAY THREAD(imsid)
```

```

DSNV401I -STR DISPLAY THREAD REPORT FOLLOWS -
DSNV402I -STR ACTIVE THREADS -
NAME      ST A   REQ ID      AUTHID  PLAN      ASID TOKEN
1 SYS3    T *    3 0002BMP255  ADMF001  PROGHR1  0019  99
          SYS3    T *    4 0001BMP255  ADMF001  PROGHR2  0018  97
2 SYS3    N      5          SYSADM          0065  0
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I -STR DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION

```

Key Description

- 1 This is a message-driven BMP.
- 2 This thread has completed sign-on processing, but a DB2 plan has not been allocated.

Figure 39. DISPLAY THREAD output showing IMS connections



Terminating IMS attachment facility threads

When an application terminates, IMS invokes an exit routine to disconnect the application from DB2. You cannot terminate a thread without causing an abend in the IMS application with which it is associated.

To terminate an IMS application, use one of these methods:

- Terminate the application.
The IMS commands /STOP REGION *reg#* ABDUMP or /STOP REGION *reg#* CANCEL can be used to terminate an application that runs in an online environment. For an application that runs in the DL/I batch environment, the z/OS command CANCEL can be used. See *IMS Command Reference* for more information about terminating IMS applications.
- Use the DB2 command CANCEL THREAD.
CANCEL THREAD can be used to cancel a particular thread or set of threads. CANCEL THREAD requires that you know the *token* for any thread that you want to cancel. Enter the following command to cancel the thread that is identified by a token in the display output:
-CANCEL THREAD(46)
When you issue CANCEL THREAD, that thread is scheduled to be terminated in DB2.

Displaying IMS-DB2 indoubt units of recovery

You can display information about threads whose status is indoubt by using the DB2 DISPLAY THREAD command.

GUPI One function of the thread that connects DB2 to IMS is to keep data in synchronization between the two systems. If the application program requires it, a change to IMS data must also be made to DB2 data. If DB2 abends while connected to IMS, IMS might commit or back out work without DB2 being aware of it. When DB2 restarts, that work is termed *indoubt*. Typically, some decision must be made about the status of the work.

To display a list of indoubt units of recovery, issue the following command:

```
-DISPLAY THREAD (imsid) TYPE (INDOUBT)
```

The command produces messages similar to these:

```
DSNV401I -STR DISPLAY THREAD REPORT FOLLOWS -
DSNV406I -STR POSTPONED ABORTT THREADS - 920
COORDINATOR          STATUS      RESET URID           AUTHID
SYS3                  P-ABORT          00017854FF6B ADMF001
V449-HAS NID= SYS3.400000000 AND ID= 0001BMP255
BATCH                 P-ABORT          00017854A8A0 ADMF001
V449-HAS NID= DSN:0001.0 AND ID= RUNP10
BATCH                 P-ABORT          00017854AA2E ADMF001
V449-HAS NID= DSN:0002.0 AND ID= RUNP90
BATCH                 P-ABORT          0001785CD711 ADMF001
V449-HAS NID= DSN:0004.0 AND ID= RUNP12
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

GUPI

Related tasks

Chapter 17, “Restarting DB2 after termination,” on page 501

Related reference

“DSNV408I” (DB2 Messages)

Recovering IMS-DB2 indoubt units of recovery

When you determine that indoubt units of recovery exist, you recover from this situation by using the DB2 RECOVER INDOUBT command.

GUPI To recover an indoubt unit, issue one of the following commands.

In each command, *imsid* is the connection name, and *pst#.psbname* is the correlation ID that is listed by the command DISPLAY THREAD. Your choice of the ACTION parameter tells whether to commit or roll back the associated unit of recovery.

- -RECOVER INDOUBT (*imsid*) ACTION (COMMIT) ID (*pst#.psbname*)
- -RECOVER INDOUBT (*imsid*) ACTION (ABORT) ID (*pst#.psbname*)

One of the following messages might be issued after you issue the RECOVER command:

```
DSNV414I - THREAD pst#.psbname COMMIT SCHEDULED
DSNV415I - THREAD pst#.psbname ABORT SCHEDULED
```

GUPI

Related tasks

“Resolving indoubt units of recovery” on page 522

Displaying postponed IMS-DB2 units of recovery

You can display a list of postponed IMS-DB2 units of recovery.

GUIP

To display a list of postponed units of recovery, issue the following command:

```
-DISPLAY THREAD (imsid) TYPE (POSTPONED)
```

In this command, *imsid* is the connection name. The command produces messages similar to these:

```
DSNV401I -STR DISPLAY THREAD REPORT FOLLOWS -
DSNV406I -STR POSTPONED ABORTT THREADS - 920
COORDINATOR          STATUS      RESET URID          AUTHID
SYS3                  P-ABORT      00017854FF6B ADMF001
V449-HAS NID= SYS3.400000000 AND ID= 0001BMP255
BATCH                 P-ABORT      00017854A8A0 ADMF001
V449-HAS NID= DSN:0001.0 AND ID= RUNP10
BATCH                 P-ABORT      00017854AA2E ADMF001
V449-HAS NID= DSN:0002.0 AND ID= RUNP90
BATCH                 P-ABORT      0001785CD711 ADMF001
V449-HAS NID= DSN:0004.0 AND ID= RUNP12
DISPLAY POSTPONED ABORT REPORT COMPLETE
DSN9022I -STR DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

GUIP

Related tasks

Chapter 17, “Restarting DB2 after termination,” on page 501

Related reference

“DSNV406I” (DB2 Messages)

Resolving IMS residual recovery entries

You can resolve IMS residual recovery entries (RREs).

1. To display the residual recovery entry (RRE) information, issue the following command:

```
/DISPLAY OASN SUBSYS subsystem-name
```

2. To purge the RRE, issue one of these commands:

- /CHANGE SUBSYS *subsystem-name* RESET
- /CHANGE SUBSYS *subsystem-name* RESET OASN *nnnn*

Where *nnnn* is the originating application sequence number that is listed in the display. The originating application sequence number is the schedule number of the program instance, indicating its place in the sequence of invocations of that program since the last cold start of IMS. IMS cannot have two indoubt units of recovery with the same schedule number.

These commands reset the status of IMS; they do not result in any communication with DB2.

IMS residual recovery entries:

At given times, IMS builds a list of *residual recovery entries* (RREs). RREs are units of recovery about which DB2 could be in doubt.

RREs arise in several situations:

- If DB2 is not operational, IMS has RREs that cannot be resolved until DB2 is operational. Those are not a problem.
- If DB2 is operational and connected to IMS, and if IMS rolled back the work that DB2 has committed, the IMS attachment facility issues message DSNM005I. If the data in the two systems must be consistent, this is a problem situation.
- If DB2 is operational and connected to IMS, RREs can still exist, even though no messages have informed you of this problem. The only way to recognize this problem is to issue the IMS /DISPLAY OASN SUBSYS command after the DB2 connection to IMS has been established.

Related troubleshooting information

“Recovering from IMS indoubt units of recovery” on page 602

Controlling IMS dependent region connections

Controlling IMS dependent region connections involves three activities: connecting from dependent regions, monitoring connection activity, and disconnecting from dependent regions.

How IMS dependent region connections work:

The IMS attachment facility that is used in the control region is also loaded into dependent regions. A connection is made from each dependent region to DB2. This connection is used to pass SQL statements and to coordinate the commitment of DB2 and IMS work.

The following process is used by IMS to initialize and connect.

1. Read the SSM from IMS.PROCLIB.

A subsystem member can be specified on the dependent region EXEC parameter. If it is not specified, the control region SSM is used. If the region is never to connect to DB2, specify a member with no entries to avoid loading the attachment facility.

2. Load the DB2 attachment facility from *prefix.SDSNLOAD*.

For a batch message processing (BMP) program, the load is not done until the application issues its first SQL statement. At that time, IMS attempts to make the connection.

For a message processing program (MPP) region or IMS Fast Path (IFP) region, the connection is made when the IMS region is initialized, and an IMS transaction is available for scheduling in that region.

An IMS dependent region establishes two connections to DB2: a region connection and an application connection, which occurs at execution of the first SQL statement.

If DB2 is not active, or if resources are not available when the first SQL statement is issued from an application program, the action taken depends on the error option specified on the SSM user entry. The options are:

Option Action

- R** The appropriate return code is sent to the application, and the SQL code is returned.
- Q** The application abends. This is a PSTOP transaction type; the input transaction is re-queued for processing, and new transactions are queued.
- A** The application abends. This is a STOP transaction type; the input transaction is discarded, and new transactions are not queued.

The region error option can be overridden at the program level by using the resource translation table (RTT).

Related information

"IMS attachment facility macro (DSNMAPN)" (DB2 Installation Guide)

Disconnecting from IMS dependent regions:

Usually, IMS master terminal operators avoid to disconnecting a dependent region explicitly.

However, they might want to change values in the SSM member of IMS.PROCLIB. To do that, they can issue /STOP REGION, update the SSM member, and issue /START REGION.

Monitoring activity on connections from DB2

A thread is established from a dependent region when an application makes its first successful DB2 request. You can issue IMS or DB2 commands to see information about connections and the applications that currently use them

- **From DB2:**

- DISPLAY THREAD (*imsid*)

- **From IMS:**

- /SSR -DISPLAY THREAD (*imsid*)

Either command produces the following messages:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS -
NAME      ST  A  REQ  ID          AUTHID    PLAN    ASID    TOKEN
conn-name s  *  req-ct corr-id    auth-id   pname   asid    token
conn-name s  *  req-ct corr-id    auth-id   pname   asid    token
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

Related tasks

"Displaying information by location" on page 433

Related reference

"DSNV404I" (DB2 Messages)

Monitoring activity of connections from IMS

You can monitor the connection to DB2 from IMS using the /DISPLAY SUBSYS command.

In addition to showing which program is active on each dependent region connection, the display also shows the LTERM user name and gives the control region connection status. The syntax of the command is:

```
/DISPLAY SUBSYS subsystem-name
```

The connection between IMS and DB2 is shown as one of the following states:

```
CONNECTED
NOT CONNECTED
CONNECT IN PROGRESS
STOPPED
STOP IN PROGRESS
INVALID SUBSYSTEM NAME=name
SUBSYSTEM name NOT DEFINED BUT RECOVERY OUTSTANDING
```

The thread status from each dependent region is shown as one of the following states:

CONN
CONN, ACTIVE (includes LTERM of user)

The following four examples show the output that might be generated when an IMS /DISPLAY SUBSYS command is issued.

The following figure shows the output that is returned for a DSN subsystem that is not connected. The IMS attachment facility issues message DSNM003I in this example.

```

0000 15.49.57          R 45,/DIS SUBSYS NEW
0000 15.49.57          IEE600I REPLY TO 45 IS;/DIS SUBSYS END
0000 15.49.57 JOB     56 DFS000I DSNM003I IMS/TM V1 SYS3 FAILED TO CONNECT TO SUBSYSTEM DSN RC=00  SYS3
0000 15.49.57 JOB     56 DFS000I      SUBSYS  CRC REGID PROGRAM LTERM      STATUS  SYS3
0000 15.49.57 JOB     56 DFS000I      DSN      :                               NON CONN  SYS3
0000 15.49.57 JOB     56 DFS000I      *83228/154957*  SYS3
0000 15.49.57 JOB     56 *46 DFS996I *IMS READY*  SYS3

```

Figure 40. Example of output from IMS /DISPLAY SUBSYS processing

The following figure shows the output that is returned for a DSN subsystem that is connected. The IMS attachment facility issues message DSNM001I in this example.

```

0000 15.58.59          R 46,/DIS SUBSYS ALL
0000 15.58.59          IEE600I REPLY TO 46 IS;/DIS SUBSYS ALL
0000 15.59.01 JOB     56 DFS551I MESSAGE REGION MPP1      STARTED ID=0001 TIME=1551 CLASS=001,002,003,004
0000 15.59.01 JOB     56 DFS000I DSNM001I IMS/TM=V1 SYS3 CONNECTED TO SUBSYSTEM DSN  SYS3
0000 15.59.01 JOB     56 DFS000I      SUBSYS  CRC REGID PROGRAM LTERM      STATUS  SYS3
0000 15.59.01 JOB     56 DFS000I      DSN      :                               CONN    SYS3
0000 15.59.01 JOB     56 DFS000I      *83228/155900*  SYS3
0000 15.59.01 JOB     56 *47 DFS996I *IMS READY*  SYS3

```

Figure 41. Example of output from IMS /DISPLAY SUBSYS processing

The following figure shows the output that is returned for a DSN subsystem that is in a stopped status. The IMS attachment facility issues message DSNM002I in this example.

```

0000 15.59.28          R 47,/STO SUBSYS ALL
0000 15.59.28          IEE600I REPLY TO 47 IS;/STO SUBSYS ALL
0000 15.59.37 JOB     56 DFS058I 15:59:37 STOP COMMAND IN PROGRESS  SYS3
0000 15.59.37 JOB     56 *48 DFS996I *IMS READY*  SYS3
0000 15.59.44          R 48,/DIS SUBSYS ALL
0000 15.59.44          IEE600I REPLY TO 48 IS;/DIS SUBSYS ALL
0000 15.59.45 JOB     56 DFS000I DSNM002I IMS/TM V1 SYS3 DISCONNECTED FROM SUBSYSTEM DSN RC = E.  SYS3
0000 15.59.45 JOB     56 DFS000I      SUBSYS  CRC REGID PROGRAM LTERM      STATUS  SYS3
0000 15.59.45 JOB     56 DFS000I      DSN      :                               STOPPED SYS3
0000 15.59.45 JOB     56 DFS000I      *83228/155945*  SYS3
0000 15.59.45 JOB     56 *49 DFS996I *IMS READY*  SYS3

```

Figure 42. Example of output from the IMS /DISPLAY SUBSYS command

The following figure shows the output that is returned for a DSN subsystem that is connected and region 1. You can use the values from the REGID and the PROGRAM fields to correlate the output of the command to the LTERM that is involved.

```

| 0000 16.09.35 JOB 56 R 59,/DIS SUBSYS ALL
| 0000 16.09.35 JOB 56 IEE600I REPLY TO 59 IS;/DIS SUBSYS ALL
| 0000 16.09.38 JOB 56 DFS000I SUBSYS CRC REGID PROGRAM LTERM STATUS SYS3
| 0000 16.09.38 JOB 56 DFS000I DSN : CONN SYS3
| 0000 16.09.38 JOB 56 DFS000I 1 CONN SYS3
| 0000 16.09.38 JOB 56 DFS000I *83228/160938* SYS3
| 0000 16.09.38 JOB 56 *60 DFS996I *IMS READY* SYS3
| 0000 16.09.38 JOB 56

```

Figure 43. Example of output from IMS /DISPLAY SUBSYS processing for a DSN subsystem that is connected and the region ID (1) that is included.

Disconnecting from IMS

The connection between IMS and DB2 ends when either IMS or DB2 terminates. Alternatively, the IMS master terminal operator can explicitly break the connection.

To break the connection, enter this command:

```
/STOP SUBSYS subsystem-name
```

That command sends the following message to the terminal that entered it, usually the master terminal operator (MTO):

```
DFS058I STOP COMMAND IN PROGRESS
```

The /START SUBSYS *subsystem-name* command is required to re-establish the connection.

In an implicit or explicit disconnect, the following message is sent to the IMS master terminal:

```
DSNM002I IMS/TM imsid DISCONNECTED FROM SUBSYSTEM subsystem-name - RC=z
```

That message uses the following reason codes (RC):

Code Meaning

- A** IMS is terminating normally (for example, /CHE FREEZE|DUMPQ|PURGE). Connected threads complete.
- B** IMS is terminating abnormally. Connected threads are rolled back. DB2 data is backed out now; DL/I data is backed out at IMS restart.
- C** DB2 is terminating normally after a STOP DB2 MODE (QUIESCE) command. Connected threads complete.
- D** DB2 is terminating normally after a STOP DB2 MODE (FORCE) command, or DB2 is terminating abnormally. Connected threads are rolled back. DL/I data is backed out now. DB2 data is backed out now if DB2 terminated normally; otherwise, it is backed out at restart.
- E** IMS is ending the connection because of a /STOP SUBSYS *subsystem-name* command. Connected threads complete.

If an application attempts to access DB2 after the connection ended and before a thread is established, the attempt is handled according to the region error option specification (R, Q, or A).

Controlling RRS connections

You can start or restart an RRS connection at any time after RRS is started.

GUPI If RRS is not started, an IDENTIFY request fails with reason code X'00F30091'.

Application programs can use the following Resource Recovery Services attachment facility (RRSAF) functions to control connections to DB2:

IDENTIFY

Establishes the task (TCB) as a user of the named DB2 subsystem. When the first task within an address space issues a connection request, the address space is initialized as a user of DB2.

SIGNON

Provides a user ID and, optionally, one or more secondary authorization IDs that are to be associated with the connection. Invokes the sign-on exit routine. Optionally, lets a thread join a global transaction.

AUTH SIGNON

Provides a user ID, an access control environment element (ACEE), and, optionally, one or more secondary authorization IDs that are to be associated with the connection. Invokes the sign-on exit routine.

CREATE THREAD

Allocates a plan. If you provide a plan name, DB2 allocates that plan. If you provide a collection name, DB2 allocates a special plan named ?RRSAF and a package list that contains the collection name.

After CREATE THREAD completes, DB2 can execute SQL statements.

TERMINATE THREAD

Deallocates the plan.

TERMINATE IDENTIFY

Removes the task as a user of DB2. If this is the last or only task in the address space with a DB2 connection, the TERMINATE IDENTIFY command terminates the address space connection to DB2.

TRANSLATE

Returns an SQL code and printable text, in the SQLCA, that describes a DB2 error reason code.

GUPI

Related reference

"Invoking the Resource Recovery Services attachment facility" (DB2 Application Programming and SQL Guide)

Related information

"Programming your applications for concurrency" (DB2 Performance Monitoring and Tuning Guide)

Abnormal termination involving DB2 and RRS

If DB2 abnormally terminates but RRS remains active, RRS might commit or roll back work without DB2 knowledge. In a similar manner, if RRS abnormally terminates after DB2 has completed phase 1 of commit processing for an application, DB2 does not know whether to commit or roll back the work.

GUPI In either case, when DB2 restarts, that work is termed *indoubt*.

DB2 cannot resolve those indoubt units of recovery (that is, commit or roll back the changes made to DB2 resources) until DB2 restarts with RRS.

If any unit of work is indoubt when a failure occurs, DB2 and RRS automatically resolve the unit of work when DB2 restarts with RRS. 

Displaying RRS indoubt units of recovery

You can display a list of RRS indoubt units of recovery.

GUPI

To display a list of indoubt units of recovery, issue the following command:
`-DISPLAY THREAD (RRSAF) TYPE (INDOUBT)`

The command produces output similar to this:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV406I - INDOUBT THREADS -
COORDINATOR          STATUS      RESET URID          AUTHID
RRSAF                INDOUBT      00019B8ADE9E      ADMF001
  V449-HAS NID= AD64101C7EED9000000000101010000 AND ID= ST47653RRS
DISPLAY INDOUBT REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

For RRSAF connections, a network ID is the z/OS RRS unit of recovery ID (URID), which uniquely identifies a unit of work. A z/OS RRS URID is a 32-character number. 

Related reference

"DSNV408I" (DB2 Messages)

Recovering RRS indoubt units of recovery manually

You might need to manually recover an indoubt unit of recovery if the RRS log is lost. When that happens, message DSN3011I is displayed on the z/OS console.

GUPI

To recover an indoubt unit of recovery:

1. Determine the correlation ID of the thread to be recovered by issuing the `DISPLAY THREAD` command.
2. Issue one of the following commands to recover the indoubt unit:
 - `-RECOVER INDOUBT (RRSAF) ACTION (COMMIT) ID (correlation-id)`
 - `-RECOVER INDOUBT (RRSAF) ACTION (ABORT) ID (correlation-id)`

The `ACTION` parameter of the `RECOVER` command indicates whether to commit or roll back the associated unit of recovery.

If you recover a thread that is part of a global transaction, all threads in the global transaction are recovered.

The following messages might be issued when you issue the `RECOVER INDOUBT` command:

```
DSNV414I - THREAD correlation-id COMMIT SCHEDULED
DSNV415I - THREAD correlation-id ABORT SCHEDULED
```

The following DSNV418I message might also be issued:

```
DSNV418I - RECOVER INDOUBT REJECTED FOR ID=correlation-id
```

If this message is issued, use the following `NID` option of `RECOVER INDOUBT`:
`-RECOVER INDOUBT(RRSAF) ACTION(action) NID(nid)`

where *nid* is the 32-character field that is displayed in the DSNV449I message.

GUPI

Related concepts

“Multiple system consistency” on page 515

Related tasks

“Resolving indoubt units of recovery” on page 522

Displaying RRS postponed units of recovery

You can display a list of RRS postponed units of recovery.

GUPI

To display a list of postponed units of recovery, issue the following command:

```
-DISPLAY THREAD (RRSAF) TYPE (POSTPONED)
```

The command produces output similar to this:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -  
DSNV406I - POSTPONED ABORT THREADS -  
COORDINATOR          STATUS      RESET URID          AUTHID  
RRSAF                 P-ABORT          00019B8ADE9E     ADMF001  
V449-HAS NID= AD64101C7EED90000000000101010000 AND ID= ST47653RRS  
DISPLAY POSTPONED ABORT REPORT COMPLETE  
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

For RRSAF connections, a network ID is the z/OS RRS unit of recovery ID (URID), which uniquely identifies a unit of work. A z/OS RRS URID is a 32-character

number. **GUPI**

Related reference

“DSNV408I” (DB2 Messages)

Monitoring and displaying RRSAF connections

RRSAF allows an application or application monitor to disassociate a DB2 thread from a TCB and later associate the thread with the same or a different TCB within the same address space.

GUPI RRSAF uses the RRS Switch Context (CTXSWCH) service to do this. Only authorized programs can execute CTXSWCH.

DB2 stores information in an RRS CONTEXT about an RRSAF thread so that DB2 can locate the thread later. An application or application monitor can then invoke CTXSWCH to disassociate the CONTEXT from the current TCB and then associate the CONTEXT with the same TCB or a different TCB.

The following command displays information about RRSAF threads, including those that access data at remote locations:

```
-DISPLAY THREAD(RRSAF)
```

The command produces output similar to the output in the following figure:

```

DSNV401I = DISPLAY THREAD REPORT FOLLOWS -
DSNV402I = ACTIVE THREADS -
NAME      ST A   REQ ID          AUTHID  PLAN    ASID  TOKEN
1 RRSAF   T      4 RRSTEST2-111  ADMF001 ?RRSAF 0024  13
2 RRSAF   T      6 RRSCDBTEST01 USRT001 TESTDBD 0024  63
3 RRSAF   DI      3 RRSTEST2-100 USRT002 ?RRSAF 001B  99
4 RRSAF   TR      9 GT01XP05     SYSADM  TESTP05 001B  235
          V444-DB2NET.LUND0.AA8007132465=16 ACCESSING DATA AT
          V446-SAN_JOSE:LUND1
DISPLAY ACTIVE REPORT COMPLETE

```

Key Description

- 1** This is an application that used CREATE THREAD to allocate the special plan that is used by RRSAF (plan name = ?RRSAF).
- 2** This is an application that connected to DB2 and allocated a plan with the name TESTDBD.
- 3** This is an application that is currently not connected to a TCB (shown by status DI).
- 4** This is an active connection that is running plan TESTP05. The thread is accessing data at a remote site.

Figure 44. DISPLAY THREAD output showing RRSAF connections

GUIP

Disconnecting RRSAF applications from DB2

You cannot disconnect an RRSAF transaction from DB2 without abnormally terminating the transaction.

GUIP

You can use the DB2 command CANCEL THREAD to cancel a particular thread. CANCEL THREAD requires that you know the *token* for any thread that you want to cancel. Issue the command DISPLAY THREAD to obtain the token number, and then enter the following command to cancel the thread:

```
-CANCEL THREAD(token)
```

When you issue CANCEL THREAD, DB2 schedules the thread for termination.

GUIP

Controlling connections to remote systems

You can control connections to remote systems, which use distributed data, by controlling the threads. Two types of threads are involved with connecting to other systems, allied threads and database access threads.

GUIP

An *allied thread* is a thread that is connected locally to your DB2 subsystem, that is from TSO, CICS, IMS, or a stored procedures address space. A *database access thread* is a thread that is initiated by a remote DBMS to your DB2 subsystem.

GUIP

Related troubleshooting information

“Recovering from database access thread failure” on page 675

Related concepts

“DB2 commands for monitoring connections to other systems” on page 463

Related tasks

“Resolving indoubt units of recovery” on page 522

Starting DDF

You can start the distributed data facility (DDF) if you have SYSOPR authority or higher.

GUIP To start the distributed data facility (DDF), if it has not already been started, use the following command:

```
-START DDF
```

When DDF is started and is responsible for indoubt thread resolution with remote partners, message DSNL432I or DSNL433I, or both, is generated. These messages summarize the responsibility DDF has for indoubt thread resolution with remote partners.

The following messages are associated with this command:

```
DSNL003I - DDF IS STARTING
DSNL004I - DDF START COMPLETE LOCATION locname
           LU netname.luname
           GENERICLU netname.gluname
           DOMAIN domain
           TCPSPORT tcpport
           SECPORT secport
           RESPOR resport
```

If the DDF is not properly installed, the START DDF command fails, and message ,DSN9032I, - REQUESTED FUNCTION IS NOT AVAILABLE, is issued. If the DDF has started, the START DDF command fails, and message ,DSNL001I, - DDF IS ALREADY STARTED, is issued. Use the DISPLAY DDF command to display the status of DDF.

When you install DB2, you can request that the distributed data facility start automatically when DB2 starts.

GUIP

Related concepts

Chapter 18, “Maintaining consistency across multiple systems,” on page 515

Related reference

“Distributed data facility panel 1: DSNTIPR” (DB2 Installation Guide)

Suspending DDF server activity

You can use the STOP DDF MODE(SUSPEND) command to suspend DDF server threads temporarily.

GUIP Suspending DDF server threads frees all resources that are held by the server threads and lets the following operations complete:

- CREATE
- ALTER
- DROP
- GRANT
- REVOKE

When you issue STOP DDF MODE(SUSPEND), DB2 waits for all active DDF database access threads to become pooled or to terminate. Two optional keywords on this command, WAIT and CANCEL, let you control how long DB2 waits and what action DB2 takes after a specified time period. 

Related reference

"-STOP DDF (DB2)" (DB2 Command Reference)

Resuming DDF server activity

You can resume suspended DDF server activity.

 To resume suspended DDF server threads, issue the START DDF command. 

Related reference

"-STOP DDF (DB2)" (DB2 Command Reference)

Displaying information about DDF work

The command DISPLAY DDF displays information regarding the status of DDF and, in addition, any information that is displayed when DDF is started, such as the location name, the LU name, the IP address, and domain names.

 To issue the DISPLAY DDF command, you must have SYSOPR authority or higher.

Tip: You can use the optional DETAIL keyword to receive additional configuration and statistical information.

The DISPLAY DDF DETAIL command is especially useful because it reflects the presence of new inbound connections that are not reflected by other commands. For example, if DDF is in INACTIVE MODE, as denoted by a DT value of I in the message DSNL090I, and DDF is stopped with mode SUSPEND, or the maximum number of active database access threads has been reached, new inbound connections are not yet reflected in the DISPLAY THREAD report. However, the presence of these new connections is reflected in the DISPLAY DDF DETAIL report, although specific details regarding the origin of the connections, such as the client system IP address or LU name, are not available until the connections are actually associated with a database access thread.

Enter one of the following commands:

- To show only the basic information, enter:
-DISPLAY DDF
- To show additional information, enter the following command:
-DISPLAY DDF DETAIL

DB2 returns output similar to this sample.

```

DSNL080I - DSNLTDDF DISPLAY DDF REPORT FOLLOWS-
DSNL081I 1 STATUS=STARTD
DSNL082I 2 LOCATION          3 LUNAME          4 GENERICLU
DSNL083I          SVL650A          USIBMSY.SYEC650A          -NONE
DSNL084I 5 TCPSPORT=446          SECPORT=0          6 RESPRT=5001
DSNL085I 7 IPADDR=:8.110.115.106
DSNL085I 7 IPADDR=2002:91E:610:1::5
DSNL086I 8 SQL      DOMAIN=v8ec103.svl.ibm.com
DSNL086I 9 RESYNC  DOMAIN=v8ec103.svl.ibm.com
DSNL087I 10 ALIAS          PORT

```

```

DSNL088I 11 ALIASLOC1          551
DSNL088I  ALIASLOC2          552
DSNL088I  ALIASLOC3          553
DSNL088I  ALIASLOC4          554
DSNL088I  ALIASLOC5          555
DSNL088I  ALIASLOC6          556
DSNL088I  ALIASLOC7          557
DSNL088I  ALIASLOC8          558
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE

```

With the DETAIL option, the following additional information is included, before the DSNL099I line:

```

DSNL090I 12 DT=      A 13 CONDBAT= 64 14 MDBAT= 64
DSNL092I 15 ADBAT=  1 16 QUEDBAT=  0 17 INADBAT=  0 18 CONQUED=  0
DSNL093I 19 DSCDBAT= 0 20 INACONN=  0
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE

```

Key Description

- 1 The status of the distributed data facility (DDF).
STOPDQ
DDF is not started.
STARTED
DDF is started.
- 2 The location name of DDF defined in the BSDS.
- 3 The fully qualified LU name for DDF (that is, the network ID and LUNAME). If "-NONE" is present, then DDF is not started.
- 4 The fully qualified generic LU name for DDF.
- 5 The TCPSPORT of DDF.
- 6 The RESPORT of DDF.
- 7 The IPv4 and IPv6 addresses. If "-NONE" is present, then DDF is not started.
- 8 The domain that accepts inbound SQL requests from remote partners. If "-NONE" is present, then DDF is not started.
- 9 The domain that accepts inbound two-phase commit resynchronization requests. If "-NONE" is present, then DDF is not started.
- 10 Column headers for a table that contains the alias names and their associated ports.
- 11 An alias name and its associated port.
- 12 The DDF thread value:
A Indicates that DDF is configured with DDF THREADS ACTIVE.
I Indicates that DDF is configured with DDF THREADS INACTIVE.
- 13 The maximum number of inbound connections.
- 14 The maximum number of concurrent active DBATs that can execute SQL.
- 15 The current number of active database access threads.
- 16 The total number of queued database access threads. This count is cumulative and resets only when DB2 restarts.
- 17 The current number of inactive DBATs (type 1 inactive threads).
- 18 The current number of connection requests that are queued and waiting.
- 19 The current number of pooled database access threads.
- 20 The current number of inactive connections (type 2 inactive connections).

GUPI

DB2 commands for monitoring connections to other systems:

DB2 commands can give you information about distributed threads.

GUIP

DISPLAY DDF

Displays information about the status and configuration of the distributed data facility (DDF), and about the connections or threads controlled by DDF.

DISPLAY LOCATION

Displays statistics about threads and conversations between a remote DB2 subsystem and the local subsystem.

DISPLAY THREAD

Displays information about DB2, distributed subsystem connections, and parallel tasks.

GUIP

Displaying information about connections with other locations:

The command DISPLAY LOCATION displays summary information about connections with other locations and can be used to display detailed information about DB2 system conversations.

GUIP

System conversations are used either for DB2 private protocol access or for supporting functions with DRDA access. Location names, SNA LU names, or IP addresses can be specified, and the DETAIL keyword is supported. To issue the DISPLAY LOCATION command, you must have SYSOPR authority or higher. Issue the following command:

-DISPLAY LOCATION(*)

DB2 returns output similar to this sample:

```

| DSNL200I -DISPLAY LOCATION REPORT FOLLOWS-
| LOCATION          PRDID    REQSTR  SERVER  CONVS
| USIBMSTODB22      DSN09010  1      0      3
|   LUND0
| USIBMSTODB23      DSN09010  0      0      0
|   LUND1
| DRDALOC           SQL08020  3      0      3
|   ::FFFF:124:63:51:17  SQL08020  0     15     15
| DISPLAY LOCATION REPORT COMPLETE

```

Use an asterisk (*) in place of the end characters of a location name. For example, use DISPLAY LOCATION(SAN*) to display information about all active connections between your DB2 subsystem and a remote location that begins with "SAN". This includes the number of conversations and the role for each non-system conversation, requester or server.

When DB2 connects with a remote location, information about that location, including LOCATION, PRDID, and LINKNAME (LUNAME or IP address), persists in the report even if no active connections exist.

The DISPLAY LOCATION command displays the following types of information for each DBMS that has active connections, except for the local subsystem:

- The location name (or RDB_NAME) of the other connected system. If the RDBNAME is not known, the LOCATION column contains one of the following identifiers:
 - A VTAM LU name in this format: '<luname>'.

- A dotted decimal IPv4 or colon hexadecimal IPv6 address.
- The PRDID, which identifies the database product at the location in the form *nnnnvrrm*:
 - *nnn* - the database product
 - *vv* - product version
 - *rr* - product release
 - *m* - product modification level
- The number of threads at the local system that are requesting data from the remote system.
- The number of threads at the local system that are acting as a server to the remote system.
- The total number of conversations that are in use between the local system and the remote system.
- The corresponding linkname representing the LUNAME or IP address of the system if different from the LOCATION name.

DB2 does not receive a location name from non-DB2 requesting DBMSs that are connected to DB2. In this case, it displays instead the LUNAME, enclosed in less-than (<) and greater-than (>) symbols, or the IP address of the requesting system.

For example, suppose two threads are at location USIBMSTODB21. One is a distributed access thread from a non-DB2 system, and the other is an allied thread that goes from USIBMSTODB21 to the non-DB2 system, both via SNA connections. The DISPLAY LOCATION command that is issued at USIBMSTODB21 displays the following output:

```

DSNL200I -DISPLAY LOCATION REPORT FOLLOWS -
LOCATION                                PRDID   REQSTR  SERVER  CONVS
NONDB2DBMS                             1       0       1
  LUND1
<LULA>                                DSN09010  0       1       1
  LULA
DISPLAY LOCATION REPORT COMPLETE

```

The following output shows the result of a DISPLAY LOCATION(*) command when DB2 is connected to the following DRDA partners:

- DB2A is connected to this DB2, using TCP/IP for DRDA connections and SNA for DB2 private-protocol connections.
- DB2SERV is connected to this DB2, using only SNA.

```

DSNL200I -DISPLAY LOCATION REPORT FOLLOWS -
LOCATION                                PRDID   REQSTR  SERVER  CONVS
DB2A                                    DSN09010  3       4       9
  LUDB2A
DB2A                                    DSN09010  2       1       3
  ::FFFF:124:38:54:16
DB2SERV                                DSN09010  1       1       3
  LULA
DISPLAY LOCATION REPORT COMPLETE

```

The DISPLAY LOCATION command displays information for each remote location that currently is, or once was, in contact with DB2. If a location is displayed with zero conversations, one of the following conditions exists:

- Sessions currently exist with the partner location, but currently no active conversations are allocated to any of the sessions.
- Sessions no longer exist with the partner because contact with the partner has been lost.

If you use the `DETAIL` parameter, each line is followed by information about conversations that are owned by DB2 system threads, including those that are used for resynchronization of indoubt units of work. 

Canceling dynamic SQL from a client application

You can use the CLI and ODBC function `SQLCancel()` or the JDBC cancel method to cancel a remote SQL request from a client application.

 To cancel SQL statements that are running on a remote DB2 server:

1. Establish an additional remote connection from your application to the remote DB2 server.
2. From that connection, issue `SQLCancel()` or invoke the JDBC cancel method.

When you cancel an SQL statement from a client application, you do not eliminate the original connection to the remote server. The original connection remains active to process additional SQL requests. Any cursor that is associated with the canceled statement is closed, and the DB2 server returns an `SQLCODE` of `-952` to the client application when you cancel a statement by using this method.

You can cancel only dynamic SQL codes that excludes transaction-level statements (`CONNECT`, `COMMIT`, `ROLLBACK`) and bind statements from a client application.

For more information about `SQLCancel()`, see *DB2 ODBC Guide and Reference*. For more information about the JDBC cancel method, see *DB2 Application Programming Guide and Reference for Java*. 

Canceling threads

You can use the command `CANCEL THREAD` to terminate threads that are active or suspended in DB2. The command has no effect if the thread is not active or suspended in DB2.

Using `CANCEL THREAD` requires `SYSOPR` authority or higher.

 If the thread is processing in VTAM or TCP/IP, you can use VTAM or TCP/IP commands to terminate the conversations, as described in the “`-CANCEL THREAD (DB2)`” section in the *DB2 Command Reference*.

`DISPLAY THREAD` can be used to determine if a thread is hung in DB2 or VTAM. In VTAM, there is no reason to use the `CANCEL` command.

To terminate a thread, enter one of the following commands:

- Cancel a thread with a token:
`-CANCEL THREAD (token)`
- Alternatively, you can use the following version of the command with either the token or LUW ID:
`-CANCEL DDF THREAD (token or luwid)`

The *token* is a 1-character to 5-character number that identifies the thread result. When DB2 schedules the thread for termination, the following message for a distributed thread is issued:

```
DSNL010I - DDF THREAD token or luwid HAS BEEN CANCELED
```

For a non-distributed thread, you see the following message:

DSNV426I - csect THREAD token HAS BEEN CANCELED

As a result of entering CANCEL THREAD, the following messages can be displayed:

DSNL009I
DSNL010I
DSNL022I

CANCEL THREAD allows you to specify that a diagnostic dump be taken. 

Related reference

"-CANCEL THREAD (DB2)" (DB2 Command Reference)

Related information

Diagnosis Guide and Reference

Effects of the CANCEL THREAD command:

A database access thread can also be in the prepared state waiting for the commit decision from the coordinator. When you issue CANCEL THREAD for a database access thread in the prepared state, the thread is converted from active to indoubt.

 The conversation with the coordinator and all conversations with downstream participants are terminated, and message DSNL450I is returned. The resources that are held by the thread are not released until the indoubt state is resolved. This is accomplished automatically by the coordinator or by using the command RECOVER INDOUBT.

When the command is entered at the DB2 subsystem that has a database access thread servicing requests from a DB2 subsystem that owns the allied thread, the database access thread is terminated. Any active SQL request (and all later requests) from the allied thread result in a "resource not available" return code.



Related tasks

"Resolving indoubt units of recovery" on page 522

Monitoring and controlling stored procedures

Stored procedures are user-written SQL programs that run at a DB2 server.

 Stored procedures run in WLM-established address spaces. To monitor and control stored procedures in WLM-established address spaces, you might need to use WLM commands rather than DB2 commands. When you execute a WLM command on a z/OS system that is part of a Sysplex, the scope of that command is the Sysplex.



Related information

"Creating a stored procedure" (DB2 Application Programming and SQL Guide)

Displaying information about stored procedures with DB2 commands:

Use the DB2 commands `DISPLAY PROCEDURE` and `DISPLAY THREAD` to obtain information about a stored procedure while it is running. Use the z/OS command `DISPLAY WLM` to obtain information about the application environment in which a stored procedure runs.

Using the DB2 DISPLAY PROCEDURE command:

This command can display the following information about stored procedures:

- Status (started, stop-queue, stop-reject, or stop-abend)
- Number of requests that are currently running and queued
- Maximum number of threads that are running a stored procedure load module and queued
- Count of timed-out SQL CALLs

To display information about all stored procedures in all schemas that have been accessed by DB2 applications, enter:

```
-DISPLAY PROCEDURE
```

This example shows two schemas (PAYROLL and HRPROD) that have been accessed by DB2 applications. You can also display information about specific stored procedures.

```
DSNX940I csect - DISPLAY PROCEDURE REPORT FOLLOWS-
----- SCHEMA=PAYROLL
PROCEDURE      STATUS    ACTIVE  QUED  MAXQ  TIMEOUT  FAIL  WLM_ENV
PAYRPRC1
                STARTED      0      0      1      0      0  PAYROLL
PAYRPRC2
                STOPQUE      0      5      5      3      0  PAYROLL
PAYRPRC3
                STARTED      2      0      6      0      0  PAYROLL
USERPRC4
                STOPREJ      0      0      1      0      1  SANDBOX
----- SCHEMA=HRPROD
PROCEDURE      STATUS    ACTIVE  QUED  MAXQ  TIMEOUT  FAIL  WLM_ENV
HRPRC1
                STARTED      0      0      1      0      1  HRPROCS
HRPRC2
                STOPREJ      0      0      1      0      0  HRPROCS
DISPLAY PROCEDURE REPORT COMPLETE
DSN9022I = DSNX9COM '-DISPLAY PROC' NORMAL COMPLETION
```

Using the DB2 DISPLAY THREAD command:

This command tells whether:

- A thread is waiting for a stored procedure to be scheduled.
- A thread is executing within a stored procedure.

The following example of `DISPLAY THREAD` output shows a thread that is executing a stored procedure:

```
!display thread(*) det
DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ! ACTIVE THREADS -
NAME      ST A  REQ ID          AUTHID  PLAN      ASID  TOKEN
BATCH    SP   3 CALLWLM      SYSADM  PLNAPPLX 0022  5
      V436-PGM=*.MYPROG, SEC=2, STMNT=1
      V429 CALLING PROCEDURE=SYSADM .WLMSP
      PROC=V61AWLM1, ASID=0085, WLM_ENV=WLMENV1
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I ! DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

The SP status indicates that the thread is executing within the stored procedure. An SW status indicates that the thread is waiting for the stored procedure to be scheduled.

The following example of DISPLAY THREAD output shows a thread that is executing a user-defined function:

```
!display thread(*) det
DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
DSNV402I ! ACTIVE THREADS -
NAME      ST A  REQ ID          AUTHID  PLAN      ASID TOKEN
BATCH    SP    27 LI33FN1      SYSADM  DSNTEP3  0021    4
V436-PGM=*.MYPROG, SEC=2, STMT=1
V429 CALLING FUNCTION =SYSADM .FUNC1
        PROC=V61AWLM1, ASID=0085, WLM_ENV=WLMENV1
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I ! DSNVDT '-DISPLAY THD' NORMAL COMPLETION
```

GUIP You can use the `-DISPLAY THREAD(*) TYPE(PROC)` command to see active threads that are running stored procedures and user-defined functions. **GUIP**

Determining the status of an application environment:

Use the command `DISPLAY WLM` to determine the status of an application environment in which a stored procedure runs.

The output from `DISPLAY WLM` lets you determine whether a stored procedure can be scheduled in an application environment.

For example, you can issue this command to determine the status of application environment `WLMENV1`:

```
D WLM,APPLENV=WLMENV1
```

You might get results like this:

```
IWM029I 15.22.22 WLM DISPLAY
APPLICATION ENVIRONMENT NAME    STATE    STATE DATA
WLMENV1                          AVAILABLE
ATTRIBUTES: PROC=DSNWLM1 SUBSYSTEM TYPE: DB2
```

The output indicates that `WLMENV1` is available, so WLM can schedule stored procedures for execution in that environment.

Refreshing the environment for stored procedures or user-defined functions:

Depending on what has changed in a stored procedure's environment, you may need WLM to refresh any existing WLM-managed stored procedures address spaces. **GUIP**

For example:

- You might want to quiesce the environment while you make a change to the JCL for the stored procedure's address space and resume the environment when your changes are complete.
- You might need to restart an application environment because the environment is in the stopped state. This is due to a failure when starting the address space that was caused by a condition that has already been corrected.

The method that you use to perform these tasks for stored procedures depends on if you are using WLM-established address spaces. 

Refreshing the environment for WLM-established address spaces:

You can use the VARY WLM command to refresh the environment for WLM-established address spaces.

 Use the VARY WLM command to complete the following tasks:

- To refresh the Language Environment® when you need to load a new version of a stored procedure, use the z/OS command:

```
VARY WLM,APPLENV=name,REFRESH
```

In this command, *name* represents the name of a WLM application environment that is associated with a group of stored procedures. The application environment is refreshed to pick up the changed load modules for all stored procedures and user-defined functions in the particular environment.

- To stop all stored procedures address spaces that are associated with WLM application environment *name*, use the following z/OS command:

```
VARY WLM,APPLENV=name,QUIESCE
```

- To change a WLM application environment from the STOPPED or QUIESCED state to the STARTED state, use the following z/OS command, where *name* is application environment name:

```
VARY WLM,APPLENV=name,RESUME
```

You also need to use the VARY WLM command with the RESUME option when WLM puts an application environment in the STOPPED state. An application environment that runs stored procedures is put in the STOPPED state when WLM detects five abnormal terminations within 10 minutes. When an application environment is in the STOPPED state, WLM does not schedule stored procedures for execution in it.

See *z/OS MVS Planning: Workload Management* for more information about the command VARY WLM. 

Obtaining diagnostic information about stored procedures:

If the startup procedures for your stored procedures address spaces contain a DD statement for CEEDUMP, Language Environment writes a small diagnostic dump to CEEDUMP when a stored procedure terminates abnormally. The output is printed after the stored procedures address space terminates.

 You can obtain the dump information by stopping the stored procedures address space in which the stored procedure is running.

Related tasks

“Refreshing the environment for stored procedures or user-defined functions” on page 469

Monitoring DDF problems by using NetView

The NetView® program lets you have a single focal point from which to view problems in the network. DDF sends an alert to NetView when a remote location is either involved in the cause of the failure or affected by the failure.

To see the recommended action for solving a particular problem, enter the selection number, and then press Enter. This displays the Recommended Action for Selected Event panel, shown in the following figure.

```

N E T V I E W          SESSION DOMAIN: CNM01  OPER2    11/03/89 10:30:06
NPDA-45A              * RECOMMENDED ACTION FOR SELECTED EVENT *    PAGE 1 OF 1
CNM01                 AR 1                                     AS 2
+-----+             +-----+
DOMAIN                RQST --- SRVR
+-----+             +-----+
USER   CAUSED - NONE
INSTALL CAUSED - NONE
FAILURE CAUSED - SNA COMMUNICATIONS ERROR:
                      RCPRI=0008 RCSEC=0001
                      FAILURE OCCURRED ON RELATIONAL DATA BASE USIBMSTODB21
ACTIONS - I008 - PERFORM PROBLEM DETERMINATION PROCEDURE FOR REASON
          CODE 3 00D31029
          I168 - FOR RELATIONAL DATA BASE USIBMSTODB22
          REPORT THE FOLLOWING LOGICAL UNIT OF WORK IDENTIFIER
          DB2NET.LUND0.A1283FFB0476.0001
ENTER DM (DETAIL MENU) OR D (EVENT DETAIL)

```

Figure 45. Recommended action for selected event panel in NetView. In this example, the AR (USIBMSTODB21) reports the problem, which affects the AS (USIBMSTODB22).

Key Description

- 1** The system that is reporting the error. The system that is reporting the error is always on the left side of the panel. That system name appears first in the messages. Depending on who is reporting the error, either the LUNAME or the location name is used.
- 2** The system that is affected by the error. The system that is affected by the error is always displayed to the right of the system that is reporting the error. The affected system name appears second in the messages. Depending on what type of system is reporting the error, either the LUNAME or the location name is used.

If no other system is affected by the error, this system does not appear on the panel.
- 3** DB2 reason code.

For more information about using NetView, see *Tivoli NetView for z/OS User's Guide*.

Related reference

"DB2 codes" (DB2 Codes)

Related information

Diagnosis Guide and Reference

DDF alerts:

Several major events generate alerts.

- Conversation failures
- Distributed security failures
- DDF abends
- DDM protocol errors
- Database access thread abends
- Distributed allied thread abends

Alerts for DDF are displayed on NetView Hardware Monitor panels and are logged in the hardware monitor database. The following figure is an example of the Alerts-Static panel in NetView.

```

NETVIEW          SESSION DOMAIN: CNM01   OPER2   11/03/89 10:29:55
NPDA-30B          * ALERTS-STATIC *
SEL# DOMAIN RESNAME TYPE TIME  ALERT DESCRIPTION:PROBABLE CAUSE
( 1) CNM01 AS      *RQST 09:58 SOFTWARE PROGRAM ERROR:COMM/REMOTE NODE
( 2) CNM01 AR      *SRVR 09:58 SOFTWARE PROGRAM ERROR:SNA COMMUNICATIONS
( 3) CNM01 P13008  CTRL 12:11 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
( 4) CNM01 P13008  CTRL 12:11 RLSO OFF DETECTED:OUTBOUND LINE
( 5) CNM01 P13008  CTRL 12:11 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
( 6) CNM01 P13008  CTRL 12:11 LINK ERROR:INBOUND LINE                    +
( 7) CNM01 P13008  CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
( 8) CNM01 P13008  CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
( 9) CNM01 P13008  CTRL 12:10 LINK ERROR:INBOUND LINE                    +
(10) CNM01 P13008  CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
(11) CNM01 P13008  CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
(12) CNM01 P13008  CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
(13) CNM01 P13008  CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
(14) CNM01 P13008  CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
(15) CNM01 P13008  CTRL 12:10 LINK ERROR:REMOTE DCE INTERFACE CABLE      +
PRESS ENTER KEY TO VIEW ALERTS-DYNAMIC OR ENTER A TO VIEW ALERTS-HISTORY
ENTER SEL# (ACTION),OR SEL# PLUS M (MOST RECENT), P (PROBLEM), DEL (DELETE)

```

Figure 46. Alerts-static panel in NetView. DDF errors are denoted by the resource name AS (server) and AR (requester). For DB2-only connections, the resource names are RS (server) and RQ (requester).

Stopping DDF

You can stop the distributed data facility (DDF) if you have SYSOPR authority or higher.

GUIP

Use one of the following commands:

```

-STOP DDF MODE (QUIESCE)
-STOP DDF MODE (FORCE)

```

The STOP DDF command causes the following messages to appear:

```

DSNL005I - DDF IS STOPPING
DSNL006I - DDF STOP COMPLETE

```

If the distributed data facility has already been stopped, the STOP DDF command fails and message DSNL002I - DDF IS ALREADY STOPPED appears. **GUIP**

Stopping DDF using the QUIESCE option:

Use the QUIESCE option whenever possible; it is the default.

GUIP

With QUIESCE, the STOP DDF command does not complete until all VTAM or TCP/IP requests have completed. In this case, no resynchronization work is necessary when you restart DDF. If any indoubt units of work require resynchronization, the QUIESCE option produces message DSNL035I. Use the FORCE option only when you must stop DDF quickly. Restart times are longer if you use FORCE.

To stop DDF with the QUIESCE option, issue the following command:

```

-STOP DDF MODE (QUIESCE)

```

GUIP

Stopping DDF using the FORCE option:

When DDF is stopped with the FORCE option, and DDF has indoubt thread responsibilities with remote partners, one or both of messages DSNL432I and DSNL433I is generated.

GUIP

DSNL432I shows the number of threads that DDF has coordination responsibility over with remote participants who could have indoubt threads. At these participants, database resources that are unavailable because of the indoubt threads remain unavailable until DDF is started and resolution occurs.

DSNL433I shows the number of threads that are indoubt locally and need resolution from remote coordinators. At the DDF location, database resources are unavailable because the indoubt threads remain unavailable until DDF is started and resolution occurs.

To force the completion of outstanding VTAM or TCP/IP requests, use the FORCE option, which cancels the threads that are associated with distributed requests.

When the FORCE option is specified with STOP DDF, database access threads in the prepared state that are waiting for the commit or abort decision from the coordinator are logically converted to the indoubt state. The conversation with the coordinator is terminated. If the thread is also a coordinator of downstream participants, these conversations are terminated. Automatic indoubt resolution is initiated when DDF is restarted.

To stop DDF with the FORCE option, issue the following command:

```
-STOP DDF MODE (FORCE)
```

GUIP

Stopping DDF using VTAM commands:

Another way to force DDF to stop is to issue the VTAM VARY NET,INACT command. This command makes VTAM unavailable and terminates DDF. VTAM forces the completion of any outstanding VTAM requests immediately.

GUIP

To stop force DDF to stop, enter the following command:

```
VARY NET,INACT,ID=db2lu,FORCE
```

where *db2lu* is the VTAM LU name for the local DB2 system.

When DDF has stopped, the following command must be issued before START DDF can be attempted:

```
VARY NET,ACT,ID=db2lu
```

GUIP

Controlling traces

Several traces are available for problem determination.

- DB2 trace
- IMS attachment facility trace
- CICS trace
- Three TSO attachment facility traces
- CAF trace stream
- RRS trace stream
- z/OS component trace used for IRLM

Types of DB2 traces

DB2 trace allows you to trace and record subsystem data and events.

GUPI

Five different types of traces are available.

Statistics

Data that allows you to conduct DB2 capacity planning and to tune the entire set of DB2 programs.

Accounting

Data that allows you to conduct DB2 capacity planning and to tune the entire set of DB2 programs.

Performance

Data about subsystem events, which can be used to do program, resource, user, and subsystem-related tuning.

Audit Data that can be used to monitor DB2 security and access to data.

Monitor

Data that can be used to monitor DB2 security and access to data.

GUPI

Related reference

"-START TRACE (DB2)" (DB2 Command Reference)

Related information

"Interpreting DB2 trace output" (DB2 Performance Monitoring and Tuning Guide)

Diagnostic traces for attachment facilities

Several trace facilities provide diagnostic information.

- IMS provides a trace facility that shows the flow of requests across the connections from the IMS control and IMS dependent regions to DB2. The trace is recorded on the IMS log if the appropriate options are specified, and then it is printed with DFSERA10 plus a formatting exit module. For more information about this trace facility, see *IMS Utilities Reference: System*.

In addition, the IMS attachment facility of DB2 provides an internal wrap-around trace table that is always active. When certain unusual error conditions occur, these trace entries are externalized on the IMS log.

- You can use the CICS trace facility to trace the CICS attachment facility.

Use the transaction CETR to control the CICS trace facility. CETR provides a series of menus that you can use to set CICS trace options to trace the CICS attachment facility. For CICS 4.1 and later, set these values in the Component Trace Options panel:

- For CICS 4.1, specify the value 2 in the FC field.
- For later releases, specify the value 2 in the RI field.

For information about using the CETR transaction to control CICS tracing, see *CICS Transaction Server for z/OS CICS Supplied Transactions*.

- The TSO attachment facility provides three tracing mechanisms:
 - The DSN trace stream
 - The CLIST trace facility
 - The SPUFI trace stream
- The call attachment facility trace stream uses the same *ddname* as the TSO DSN trace stream, but is independent of TSO.
- The RRSAF trace stream uses the same *ddname* as the TSO DSN trace stream, but is independent of TSO. An RRSAF internal trace is included in any ABEND dump that is produced by RRSAF. This tracing facility provides a history of RRSAF usage that can aid in diagnosing errors in RRSAF.

Controlling the DB2 trace

DB2 provides commands for controlling the collection of this data.

GUIP To use the trace commands, you must have one of the following types of authority:

- SYSADM or SYSOPR authority
- Authorization to issue start and stop trace commands (the TRACE privilege)
- Authorization to issue the display trace command (the DISPLAY privilege)

The trace commands include:

START TRACE

Invokes one or more different types of trace.

DISPLAY TRACE

Displays the trace options that are in effect.

STOP TRACE

Stops any trace that was started by either the START TRACE command or as a result of the parameters that were specified during installation or migration.

MODIFY TRACE

Changes the trace events (IFCIDs) that are being traced for a specified active trace.

You can specify several parameters to further qualify the scope of a trace. You can trace specific events within a trace type as well as events within specific DB2 plans, authorization IDs, resource manager IDs, and locations. You can also control where trace data is sent.

When you install DB2, you can request that any trace type and class start automatically when DB2 starts. **GUIP**

Related reference

"Tracing parameters panel: DSNTIPN" (DB2 Installation Guide)

Diagnostic trace for the IRLM

You can control diagnostic traces for the IRLM using z/OS commands.

MODIFY *irlmproc*,SET,TRACE

Dynamically sets the maximum number of trace buffers for each trace type. IRLM uses this value only when the external component trace writer is not activated.

MODIFY *irlmproc*,STATUS,TRACE

Displays the status of traces and the number of trace buffers that are used for each trace type. Also displays indication of whether the external component trace writer is active for the trace.

START *irlmproc*,TRACE=YES

Captures traces in wrap-around IRLM buffers at IRLM startup.

TRACE CT

Starts, stops, or modifies a diagnostic trace for IRLM. The TRACE CT command acts independently of traces that are started automatically during IRLM startup.

Recommendations:

- Do not use the external component trace writer to write traces to the data set.
- Activate all traces during IRLM startup. Use the command **START** *irlmproc*,TRACE=YES to activate all traces.

Related reference

"z/OS IRLM commands" (DB2 Command Reference)

Controlling the resource limit facility (governor)

The governor allows the system administrator to limit the amount of time that is permitted for the execution of the SELECT, UPDATE, DELETE, and INSERT dynamic SQL statements.

GUIP DB2 provides these commands for controlling the governor:

START RLIMIT

Starts the governor and identifies a resource limit specification table. You can also use START RLIMIT to switch resource limit specification tables.

DISPLAY RLIMIT

Displays the current status of the governor. If the governor has been started, the output from the command also identifies the resource limit specification table.

STOP RLIMIT

Stops the governor and removes any set limits.

The limits are defined in resource limit specification tables and can vary for different users. One resource limit specification table is used for each invocation of the governor and is identified on the START RLIMIT command.

When you install DB2, you can request that the governor start automatically when DB2 starts. **GUIP**

Related reference

"Facilities for controlling resource usage" (DB2 Performance Monitoring and Tuning Guide)

"Operator functions panel: DSNTIPO" (DB2 Installation Guide)

Changing subsystem parameter values

You can modify the values of subsystem parameters dynamically even while DB2 is running.

GUIP Use the following procedure to modify values dynamically:

1. Run the installation process in UPDATE mode, specifying any new parameter values. This process produces a new DSNTIJUZ job with the new values; it also saves these values in the file that is specified as the output member name on panel DSNTIPA1.
2. Assemble and link-edit the new DSNTIJUZ job, and then submit the job to create the new load module with the new subsystem parameter values.
3. Issue the SET SYSPARM command to change the subsystem parameters dynamically:

```
SET SYSPARM LOAD(load-module-name)
```

where *load-module-name* is the same as the output member name in step 1.

If you want to specify the load module name that is used during DB2 startup, you can issue the following command:

```
SET SYSPARM RELOAD
```

GUIP

Related reference

"Main panel: DSNTIPA1" (DB2 Installation Guide)

"-SET SYSPARM (DB2)" (DB2 Command Reference)

Chapter 16. Managing the log and the bootstrap data set

The DB2 log registers data changes and significant events as they occur. The bootstrap data set (BSDS) contains information about the data sets that contain the log. You can perform a variety of tasks to ensure that DB2 logging satisfies the needs of your environment.

DB2 writes each log record to a disk data set called the *active log*. When the active log is full, DB2 copies its contents to a disk or tape data set called the *archive log*. That process is called *offloading*. This section describes:

“How database changes are made”

“How the initial DB2 logging environment is established” on page 481

“Management of the bootstrap data set” on page 498

“Discarding archive log records” on page 494

For information about the physical and logical records that make up the log, see “Reading log records.” That topic also contains information about how to write a program to read log records.

How database changes are made

Before you can fully understand how logging works, you need to be familiar with how database changes are made to ensure consistency.

This section discusses units of recovery and rollbacks.

Units of recovery and points of consistency

A *unit of recovery* is the work, done by a single DB2 DBMS for an application, that changes DB2 data from one point of consistency to another. A *point of consistency* (also *sync point* or *commit point*) is a time when all recoverable data that an application program accesses is consistent with other data.

A unit of recovery begins with the first change to the data after the beginning of the job or following the last point of consistency. The unit of recovery ends at a later point of consistency. An example of units of recovery within an application program is shown in the following figure.

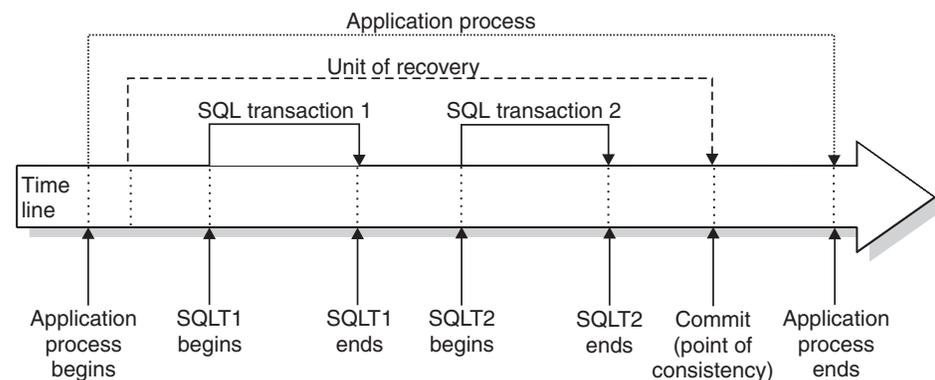


Figure 47. A unit of recovery within an application process

In this example, the application process makes changes to databases at SQL transaction 1 and 2. The application process can include a number of units of recovery or just one, but any complete unit of recovery ends with a commit point.

For example, a bank transaction might transfer funds from account A to account B. First, the program subtracts the amount from account A. Next, it adds the amount to account B. After subtracting the amount from account A, the two accounts are inconsistent. These accounts are inconsistent until the amount is added to account B. When both steps are complete, the program can announce a point of consistency and thereby make the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. The SQL COMMIT statement causes a point of consistency during program execution under TSO. A sync point causes a point of consistency in CICS and IMS programs.

Related concepts

“Multiple system consistency” on page 515

How DB2 rolls back work

If failure occurs within a unit of recovery, DB2 rolls back (backs out) any changes to data, returning the data to its state at the start of the unit of recovery; that is, DB2 undoes the work.

For a partition-by-growth table space, if a new partition was added in the unit of recovery, any uncommitted updates can be backed out, but the physical partition is not deleted.

The events are shown in the following figure.

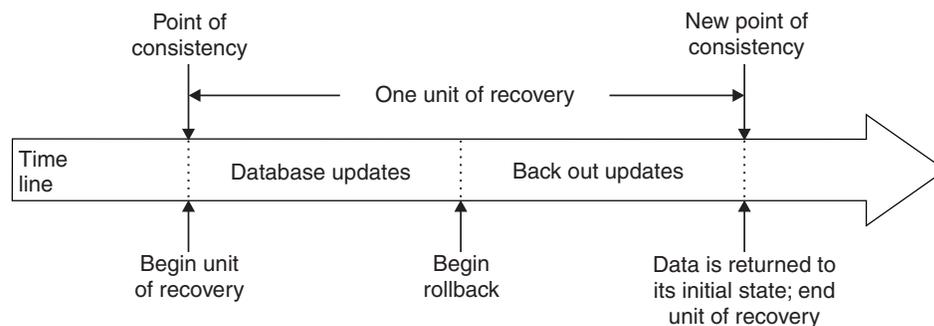


Figure 48. Unit of recovery (rollback)

The possible events that trigger "Begin rollback" in this figure include:

- SQL ROLLBACK statement
- Deadlock (reported as SQLCODE -911)
- Timeout (reported as SQLSTATE 40001)

The effects of inserts, updates, and deletes to large object (LOB) values are backed out along with all the other changes that were made during the unit of work that is being rolled back, even if the LOB values that were changed reside in a LOB table space that has the LOG NO attribute.

GUPI An operator or an application can issue the CANCEL THREAD command with the NOBACKOUT option to cancel long-running threads without backing out

data changes. DB2 backs out changes to catalog and directory tables regardless of the NOBACKOUT option. As a result, DB2 does not read the log records and does not write or apply the compensation log records. After CANCEL THREAD NOBACKOUT processing, DB2 marks all objects that are associated with the thread as refresh-pending (REFP) and puts the objects in a logical page list (LPL). For information about how to reset the REFP status, see *DB2 Utility Guide and Reference*. 

The NOBACKOUT request might fail for either of the following two reasons:

- DB2 does not completely back out updates of the catalog or directory (message DSNI032I with reason 00C900CC).
- The thread is part of a global transaction (message DSNV439I).

How the initial DB2 logging environment is established

The initial DB2 logging environment is established during installations of DB2.

Installation panels enable you to specify options, such as whether to have dual active logs (strongly recommended), what media to use for archive log volumes, and how many log buffers to have.

Related information

"System resource data set names panel: DSNTIPH" (DB2 Installation Guide)

How DB2 creates log records

Log records typically go through a standard cycle.

1. DB2 registers changes to data and significant events in recovery log records.
2. DB2 processes recovery log records and breaks them into segments if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM control intervals (CIs). Each log record is identified by a continuously increasing RBA in the range 0 to $2^{48}-1$, where 2^{48} represents 2 to the 48th power. (In a data sharing environment, a log record sequence number (LRSN) is used to identify log records. See *DB2 Data Sharing: Planning and Administration* for more information.)
4. The CIs are written to a set of predefined disk *active log data sets*, which are used sequentially and recycled.
5. As each active log data set becomes full, its contents are automatically *offloaded* to a new *archive log data set*.

If you change or create data that is compressed, the data logged is also compressed. Changes to compressed rows that result from inserts, updates, and deletes are also logged as compressed data. Updates to compressed indexes are not logged as compressed data.

How DB2 writes the active log

DB2 writes the log buffers to an active log data set when they become full, when the write threshold is reached, or, more often, when the DB2 subsystem forces the log buffer to be written.

When DB2 forces the log buffer to be written (such as at commit time), the same control interval can be written several times to the same location.

Be sure to set your ZPARMs are set so that there are enough log buffers to avoid the need to wait for a buffer to become available (DSN6LOGP OUTBUFF

parameter). Switching log data sets may also cause a temporary performance impact when the switch takes place and the associated recovery checkpoint is taken. This can be minimized by ensuring that the active log data sets are large enough to avoid frequent switching. In addition, some events can cause log buffers to be written before the ZPARM-defined threshold has been reached. These events include, but are not limited to:

- Phase 1 commit
- Abort processing
- GBP-dependent index split
- Mass delete in a data-sharing environment
- Use of GBPCACHE NO
- All log buffers being filled

Consider the probable frequency of these events when you determine how often to commit changes.

When DB2 is initialized, the active log data sets that are named in the BSDS are dynamically allocated for exclusive use by DB2 and remain allocated exclusively to DB2 (the data sets were allocated as DISP=OLD) until DB2 terminates. Those active log data sets cannot be replaced, nor can new ones be added, without terminating and restarting DB2. The size and number of log data sets is indicated by what was specified by installation panel DSNTIPL. The use of dual active logs increases availability as well as the reliability of recovery by eliminating a single point of failure.

How DB2 writes (offloads) the archive log

The process of copying active logs to archive logs is called *offloading*.

The relationship of offloading to other logging events is shown schematically in the following figure.

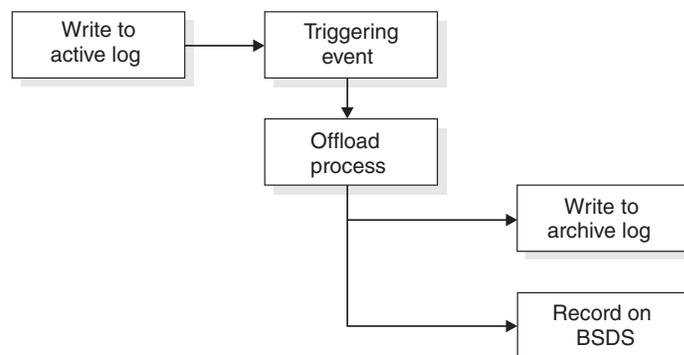


Figure 49. The offloading process

During the process, DB2 determines which data set to offload. Using the last log relative byte address (RBA) that was offloaded, as registered in the BSDS, DB2 calculates the log RBA at which to start. DB2 also determines the log RBA at which to end, from the RBA of the last log record in the data set, and registers that RBA in the BSDS.

When all active logs become full, the DB2 subsystem runs an offload and halts processing until the offload is completed. If the offload processing fails when the active logs are full, DB2 cannot continue doing any work that requires writing to the log.

Related troubleshooting information

“Recovering from active log failures” on page 613

What triggers an offload

An offload of an active log to an archive log can be triggered by several events.

The most common situations that trigger an offload include:

- An active log data set is full.
- DB2 starts, but an active log data set is full.
- The ARCHIVE LOG command is issued.

An offload can be also triggered by two uncommon events:

- An error occurs while writing to an active log data set. The data set is truncated before the point of failure, and the record that failed to write becomes the first record of the next data set. An offload is triggered for the truncated data set as in a normal end-of-file condition. With dual active logs, both copies are truncated so the two copies remain synchronized.
- The last unarchived active log data set becomes full. Message DSNJ110E is issued, stating the percentage of its capacity in use; IFCID trace record 0330 is also issued if statistics class 3 is active. If all active logs become full, DB2 issues the following message and stops processing until offloading occurs.

DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS

Role of the operator in the offload process

When an active log is ready to be offloaded, a request can be sent to the z/OS console operator to mount a tape or prepare a disk unit.

The value of the field WRITE TO OPER of the DSNTIPA installation panel determines whether the request is received. If the value is YES, the request is preceded by a WTOR (message number DSNJ008E) informing the operator to prepare an archive log data set for allocating.

The operator need not respond to message DSNJ008E immediately. However, delaying the response delays the offload process. It does not affect DB2 performance unless the operator delays response for so long that DB2 uses all the active logs.

The operator can respond by canceling the offload. In that case, if the allocation is for the first copy of dual archive data sets, the offload is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single-copy mode, but for the one data set only.

When DB2 switches active logs and finds that the offload task has been active since the last log switch, it issues the following message to notify the operator of a possible outstanding tape mount or some other problem that prevents the offload of the previous active log data set.

DSNJ017E - *csect-name* WARNING - OFFLOAD TASK HAS BEEN ACTIVE SINCE
date-time AND MAY HAVE STALLED

DB2 continues processing. The operator can cancel and then restart the offload.

Messages that are returned during offloading

During the offload process, DB2 sends a series of messages to the z/OS console. Most of these messages include information about the RBA ranges in the various log data sets.

- The following message appears during DB2 initialization when the current active log data set is found, and after a data set switch. During initialization, the STARTRBA value in the message does not refer to the beginning of the data set, but to the position in the log where logging is to begin.

```
DSNJ001I - csect-name CURRENT COPY n ACTIVE LOG DATA SET IS
          DSNAME=..., STARTRBA=..., ENDRBA=...
```

- The following message appears when an active data set is full:

```
DSNJ002I - FULL ACTIVE LOG DATA SET DSNAME=...,
          STARTRBA=..., ENDRBA=...
```

- One of the following message appears when offload reaches end-of-volume or end-of-data-set in an archive log data set:

The non-data sharing version of this message is:

```
DSNJ003I - FULL ARCHIVE LOG VOLUME DSNAME=..., STARTRBA=..., ENDRBA=...,
          STARTTIME=..., ENDTIME=..., UNIT=..., COPYnVOL=...,
          VOLSPAN=..., CATLG=...
```

The data sharing version of this message is:

```
DSNJ003I - FULL ARCHIVE LOG VOLUME DSNAME=..., STARTRBA=..., ENDRBA=...,
          STARTLRSN=..., ENDLRSN=..., UNIT=..., COPYnVOL=...,
          VOLSPAN=..., CATLG=...
```

- The following message appears when one data set of the next pair of active logs is not available because of a delay in offloading, and logging continues on one copy only:

```
DSNJ004I - ACTIVE LOG COPY n INACTIVE, LOG IN SINGLE MODE,
          ENDRBA=...
```

- The following message appears when dual active logging resumes after logging has been performed on one copy only:

```
DSNJ005I - ACTIVE LOG COPY n IS ACTIVE, LOG IN DUAL MODE,
          STARTRBA=...
```

- The following message indicates that the offload task has ended:

```
DSNJ139I LOG OFFLOAD TASK ENDED
```

Effects of interruptions and errors on the offload process

DB2 is able to handle some types of interruptions during the offloading process.

Here is how DB2 handles the following interruptions in the offloading process:

- The command STOP DB2 does not take effect until offloading is finished.
- A DB2 failure during offload causes offload to begin again from the previous start RBA when DB2 is restarted.
- Offload handling of read I/O errors on the active log is described under “Recovering from active log failures” on page 613, or write I/O errors on the archive log, under “Recovering from archive log failures” on page 618.
- An unknown problem that causes the offload task to hang means that DB2 cannot continue processing the log. This problem might be resolved by retrying the offload, which you can do by using the option CANCEL OFFLOAD of the command ARCHIVE LOG.

Archive log data sets

Archive log data sets can be placed on standard label tapes or disks and can be managed by DFSMSHsm (Data Facility Hierarchical Storage Manager). Archive logs are always written by QSAM.

Prior to DB2 Version 9, archive logs on tape are read by BSAM; those on disk are read by BDAM. From DB2 Version 9 forward, they are always read using BSAM. The block size of an archive log data set is a multiple of 4 KB.

Output archive log data sets are dynamically allocated, with names chosen by DB2. The data set name prefix, block size, unit name, and disk sizes that are needed for allocation are specified when DB2 is installed, and recorded in the DSNZPxxx module. You can also choose, at installation time, to have DB2 add a date and time to the archive log data set name.

Restrictions: For archive log data sets and volumes:

- You cannot specify specific volumes for new archive logs. If allocation errors occur, offloading is postponed until the next time loading is triggered.
- Do not use partitioned data set extended (PDSE) for archive log data. PDSEs are not supported for archive logs.

Related reference

"Archive log data set parameters panel: DSNTIPA" (DB2 Installation Guide)

"System resource data set names panel: DSNTIPH" (DB2 Installation Guide)

How dual archive logging works

Each log control interval (CI) retrieved from the active log is written to two archive log data sets. The log records that are contained on a pair of dual archive log data sets are identical, but ends-of-volumes are not synchronized for multivolume data sets.

Archiving to disk offers faster recoverability but is more expensive than archiving to tape. If you use dual logging, on installation panel DSNTIPA enables you to specify that the primary copy of the archive log go to disk and the secondary copy go to tape.

Dual archive logging increases recovery speed without using as much disk. The second tape is intended as a backup, or it can be sent to a remote site in preparation for disaster recovery. To make recovering from the COPY2 archive tape faster at the remote site, use the installation parameter ARC2FRST to specify that COPY2 archive log should be read first. Otherwise, DB2 always attempts to read the primary copy of the archive log data set first.

Tips for archiving

You can archive to tape or disk.

Tips for archiving to tape:

If you choose to archive to tape, certain tips can help you avoid problems.

If the unit name reflects a tape device, DB2 can extend to a maximum of twenty volumes. DB2 passes a file sequence number of 1 on the catalog request for the first file on the next volume. Although a file sequence number of 1 might appear to be an error in the integrated catalog facility catalog, be aware that this situation causes no problems in DB2 processing.

If you choose to offload to tape, consider adjusting the size of your active log data sets so that each data set contains the amount of space that can be stored on a nearly full tape volume. That adjustment minimizes tape handling and volume mounts, and it maximizes the use of tape resources. However, such an adjustment is not always necessary.

If you want the active log data set to fit on one tape volume, consider placing a copy of the BSDS on the same tape volume as the copy of the active log data set. Adjust the size of the active log data set downward to offset the space that is required for the BSDS.

Tips for archiving to disk:

If you choose to archive to disk, certain tips can help you avoid problems.

All archive log data sets that are allocated on disk must be cataloged. If you choose to archive to disk, the field CATALOG DATA of installation panel DSNTIPA must contain YES. If this field contains NO, and you decide to place archive log data sets on disk, you receive message DSNJ072E each time an archive log data set is allocated, although the DB2 subsystem still catalogs the data set.

If you use disk storage, ensure that the primary and secondary space quantities and block size and allocation unit are large enough so that the disk archive log data set does not attempt to extend beyond 15 volumes. The possibility of unwanted z/OS B37 or E37 abends during the offload process is thereby minimized. Primary space allocation is set with the PRIMARY QUANTITY field of the DSNTIPA installation panel. If the archive log is not allocated with the DSNTYPE=LARGE attribute, then the primary space quantity must be less than 64KB tracks because of the DFSMS Direct Access Device Space Management limit of 64KB tracks on a single volume when allocating a sequential disk data set without the DSNTYPE=LARGE attribute.

Tips for archiving with DFSMS:

You can use DFSMS (Data Facility Storage Management Subsystem) to manage archive log data sets.

When archiving to disk, DB2 uses the number of online storage volumes for the specified unit to determine a count of candidate volumes, up to a maximum of 15 volumes. If you are using SMS to direct archive log data set allocation, override this candidate volume count by specifying YES for the field SINGLE VOLUME on installation panel DSNTIPA. This enables SMS to manage the allocation volume count appropriately when creating multi-volume disk archive log data sets.

Because SMS requires disk data sets to be cataloged, ensure that the field CATALOG DATA on installation panel DSNTIPA contains YES. Even if it does not, message DSNJ072E is returned, and DB2 forces the data set to be cataloged.

Prior to DB2 Version 9: DB2 uses the basic direct access method (BDAM) to read archive logs from disk. DFSMS does not support reading of extended sequential format data sets using BDAM. Extended sequential format data sets may be striped, compressed, or both. Therefore, do not direct DFSMS to assign extended sequential format attributes to your archive log data sets.

DB2 Version 9 and later: DB2 uses the basic sequential access method (BSAM) to read archive logs from disk and BSAM supports the use of Extended Format (EF)

| data sets. However, the use of EF data sets for archive logs is not supported until
| DB2 Version 9 new function mode is activated. To ensure that you do not
| encounter any compatibility or coexistence problems, you should not create EF
| archive log data sets until all members of a data sharing group are migrated to
| new function mode.

Ensure that DFSMS does not alter the LRECL, BLKSIZE, or RECFM of the archive log data sets. Altering these attributes could result in read errors when DB2 attempts to access the log data.

Attention: DB2 does not issue an error or a warning if you write or alter archive data to an unreadable format. For example, if DB2 successfully writes archive log data to an extended format data set, DB2 issues an error message only when you attempt to read that data, not when the data is written.

Automatic archive log deletion

You can use a disk or tape management system to delete archive log data sets or tapes automatically.

The length of the retention period (in days), which is passed to the management system in the JCL parameter RETPD, is determined by the RETENTION PERIOD field on the DSNTIPA installation panel.

The default for the retention period keeps archive logs forever. Any other retention period must be long enough to contain as many recovery cycles as you plan for. For example, if your operating procedures call for a full image copy every sixty days of the least frequently-copied table space, and you want to keep two complete image copy cycles on hand at all times, you need an archive log retention period of at least 120 days. For more than two cycles, you need a correspondingly longer retention period.

If archive log data sets or tapes are deleted automatically, the operation does not update the archive log data set inventory in the BSDS. If you want, you can update the BSDS with the change log inventory utility. The update is not really necessary; it wastes space in the BSDS to record old archive logs, but it does no other harm because the archive log data set inventory wraps and automatically deletes the oldest entries.

Related concepts

“Recommendations for changing the BSDS log inventory” on page 499

Related reference

“Archive log data set parameters panel: DSNTIPA” (DB2 Installation Guide)

How DB2 retrieves log records

Normal DB2 operation and recovery tasks rely on the availability of log records. DB2 retrieves log records from different sources, depending on the situation.

Log records are retrieved by DB2 through the following events:

- A log record is requested using its RBA.
- DB2 searches for the log record in the following locations in the order in which they are presented:
 1. The log buffers.

2. The active logs. The bootstrap data set registers which log RBAs apply to each active or archive log data set. If the record is in an active log, DB2 dynamically acquires a buffer, reads one or more CIs, and returns one record for each request.
3. The archive logs. DB2 determines which archive volume contains the CIs, dynamically allocates the archive volume, acquires a buffer, and reads the CIs.

Managing the log

You can control and monitor log activity by using several DB2 commands and a utility.

Quiescing activity before offloading

You can use the MODE(QUIESCE) option of the ARCHIVE LOG command to ensure that activity has stopped before the log is archived.

GUPI With this option, DB2 work is quiesced after a commit point, and the resulting point of consistency is captured in the current active log before it is offloaded. Unlike the QUIESCE utility, ARCHIVE LOG MODE(QUIESCE) does not force all changed buffers to be written to disk and does not record the log RBA in SYSIBM.SYSCOPY. It does record the log RBA in the bootstrap data set.

Consider using MODE(QUIESCE) when planning for offsite recovery. Using MODE(QUIESCE) creates a system-wide point of consistency, which can minimize the number of data inconsistencies when the archive log is used with the most current image copy during recovery.

In a data sharing group, ARCHIVE LOG MODE(QUIESCE) might result in a delay before activity on all members has stopped. If this delay is unacceptable to you, consider using ARCHIVE LOG SCOPE(GROUP) instead. This command causes truncation and offload of the logs for each active member of a data sharing group. Although the resulting archive log data sets do not reflect a point of consistency, all the archive logs are made at nearly the same time and have similar LRSN values in their last log records. When you use this set of archive logs to recover the data sharing group, you can use the ENDLRSN option in the CRESTART statement of the change log inventory utility (DSNJU003) to truncate all the logs in the group to the same point in time.

The MODE(QUIESCE) option suspends all new update activity on DB2 up to the maximum period of time that is specified on the installation panel DSNTIPA. If the time needed to quiesce is less than the time that is specified, the command completes successfully; otherwise, the command fails when the time period expires. This time amount can be overridden when you issue the command, by using the TIME option:

```
-ARCHIVE LOG MODE(QUIESCE) TIME(60)
```

The preceding command allows for a quiesce period of up to 60 seconds before archive log processing occurs.

Important: Use of this option during prime time, or when time is critical, can cause a significant disruption in DB2 availability for all jobs and users that use DB2 resources.

By default, the command is processed asynchronously from the time you submit the command. (To process the command synchronously with other DB2 commands, use the WAIT(YES) option with QUIESCE; the z/OS console is then locked from DB2 command input for the entire QUIESCE period.)

During the quiesce period:

- Jobs and users on DB2 are allowed to go through commit processing, but they are suspended if they try to update any DB2 resource after the commit.
- Jobs and users that only read data can be affected, because they can be waiting for locks that are held by jobs or users that were suspended.
- New tasks can start, but they are not allowed to update data.

As shown in the following example, the DISPLAY THREAD output issues message DSNV400I to indicate that a quiesce is in effect:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV400I - ARCHIVE LOG QUIESCE CURRENTLY ACTIVE
DSNV402I - ACTIVE THREADS -
NAME      ST A  REQ ID          AUTHID  PLAN    ASID  TOKEN
BATCH    T *   20 TEPJOB        SYSADM  DSNTEP3 0012   12
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
```

When all updates are quiesced, the quiesce history record in the BSDS is updated with the date and time that the active log data sets were truncated, and with the last-written RBA in the current active log data sets. DB2 truncates the current active log data sets, switches to the next available active log data sets, and issues message DSNJ311E, stating that offload started.

If updates cannot be quiesced before the quiesce period expires, DB2 issues message DSNJ317I, and archive log processing terminates. The current active log data sets are not truncated and not switched to the next available log data sets, and offload is not started.

Regardless of whether the quiesce is successful, all suspended users and jobs are then resumed, and DB2 issues message DSNJ312I, stating that the quiesce is ended and update activity is resumed.

If ARCHIVE LOG is issued when the current active log is the last available active log data set, the command is not processed, and DB2 issues this message:

```
DSNJ319I - csect-name CURRENT ACTIVE LOG DATA SET IS THE LAST
          AVAILABLE ACTIVE LOG DATA SET.  ARCHIVE LOG PROCESSING WILL
          BE TERMINATED.
```

If ARCHIVE LOG is issued when another ARCHIVE LOG command is already in progress, the new command is not processed, and DB2 issues this message:

```
DSNJ318I - ARCHIVE LOG COMMAND ALREADY IN PROGRESS.
```

GUPI

Related reference

"Archive log data set parameters panel: DSNTIPA" (DB2 Installation Guide)

Archiving the log

If you are a properly authorized operator, you can archive the current DB2 active log data sets whenever required by issuing the ARCHIVE LOG command. Using

ARCHIVE LOG can help with diagnosis by allowing you to quickly offload the active log to the archive log where you can use DSN1LOGP to further analyze the problem.

You must have either SYSADM authority or have been granted the ARCHIVE privilege.

GUIP

To archive the log, enter the following command:

```
-ARCHIVE LOG
```

When you issue the preceding command, DB2 truncates the current active log data sets, runs an asynchronous offload, and updates the BSDS with a record of the offload. The RBA that is recorded in the BSDS is the beginning of the last complete log record that is written in the active log data set that is being truncated.

Example

You can use the ARCHIVE LOG command as follows to capture a point of consistency for the MSTR01 and XUSR17 databases:

```
-STOP DATABASE (MSTR01,XUSR17)
-ARCHIVE LOG
-START DATABASE (MSTR01,XUSR17)
```

In this simple example, the STOP command stops activity for the databases before archiving the log.

Canceling log offloads

In some cases, the offload of an active log might be suspended when something goes wrong with the offload process, such as a problem with allocation or tape mounting. If the active logs cannot be offloaded, the DB2 active log data sets become full and DB2 stops logging.

To cancel (and retry) an offload, issue this command:

```
-ARCHIVE LOG CANCEL OFFLOAD
```

When you enter the command, DB2 restarts the offload, beginning with the oldest active log data set and proceeding through all active log data sets that need offloading. If the offload fails again, you must fix the problem that is causing the failure before the command can work. **GUIP**

Dynamically changing the checkpoint frequency

Use the LOGLOAD or CHKTIME option of the SET LOG command to dynamically change the checkpoint frequency without recycling DB2.

GUIP

The LOGLOAD value specifies the number of log records that DB2 writes between checkpoints. The CHKTIME value specifies the number of minutes between checkpoints. Either value affects the restart time for DB2.

For example, during prime shift, your DB2 shop might have a low logging rate but require that DB2 restart quickly if it terminates abnormally. To meet this restart requirement, you can decrease the LOGLOAD value to force a higher checkpoint

frequency. In addition, during off-shift hours, the logging rate might increase as batch updates are processed, but the restart time for DB2 might not be as critical. In that case, you can increase the LOGLOAD value which lowers the checkpoint frequency.

You can also use either the LOGLOAD option or the CHKTIME option to initiate an immediate system checkpoint:

```
-SET LOG LOGLOAD(0)
-SET LOG CHKTIME(0)
```

The CHKFREQ value that is altered by the SET LOG command persists only while DB2 is active. On restart, DB2 uses the CHKFREQ value in the DB2 subsystem parameter load module. 

Related reference

"-SET LOG (DB2)" (DB2 Command Reference)

Setting limits for archive log tape units

Use the DB2 command SET ARCHIVE to set the upper limit for the number of and the deallocation time of tape units for the archive log.

 This command overrules the values that are specified during installation or in a previous invocation of the SET ARCHIVE command. The changes that are initiated by SET ARCHIVE are temporary; at restart, DB2 uses the values that are set during installation. 

Related reference

"-SET ARCHIVE (DB2)" (DB2 Command Reference)

Monitoring the system checkpoint

DB2 schedules a system checkpoint every time it switches active log data sets, regardless of the currently defined checkpoint frequency.

If DB2 switches active logs and finds that there has not been a system checkpoint since the last log switch, it issues the following message to notify the operator that the system checkpoint processor might not be functioning.

```
DSNJ016E - csect-name WARNING - SYSTEM CHECKPOINT PROCESSOR MAY
HAVE STALLED. LAST CHECKPOINT WAS TAKEN date-time
```

DB2 continues processing. This situation can result in a very long restart if logging continues without a system checkpoint. If DB2 continues logging beyond the defined checkpoint frequency, quiesce activity and terminate DB2 to minimize the restart time.

You can issue the DISPLAY LOG command or run the print log map utility (DSNJU004) to display the most recent checkpoint.

Related tasks

"Displaying log information"

Displaying log information

Use the DISPLAY LOG command to display the current checkpoint frequency. The checkpoint frequency can be either the number of log records or the minutes between checkpoints.

GUPI You can obtain additional information about log data sets and checkpoints from the print log map utility (DSNJU004). **GUPI**

Related reference

"-DISPLAY LOG (DB2)" (DB2 Command Reference)

"-SET LOG (DB2)" (DB2 Command Reference)

"DSNJU004 (print log map)" (DB2 Utility Guide and Reference)

Resetting the log RBA

Each DB2 subsystem writes its own recovery logs. The log records are sequenced with a Relative Byte Address (RBA), which you need to reset before the RBA reaches its maximum value.

To determine when to reset the log RBA, use one of the following two methods:

- Apply the PTF for APAR PK27611. After you apply this APAR, message DSNJ032I is issued at the active log switch when the RBA threshold is reached. If the RBA exceeds x'F8000000000', the message is issued with the keyword WARNING and processing continues. If the RBA exceeds x'FFFF00000000', the message is issued with the keyword CRITICAL, and DB2 is stopped. To resolve any outstanding units of work, DB2 will restart automatically in restart-light mode. Then, DB2 will stop again. In this situation, you need to restart DB2 in ACCESS(MAINT) mode, and you must reset the log RBA value.
- Calculate how much space is left in the log. You can use the print log map (DSNJU004) utility to obtain the highest written RBA value in the log. Subtract this RBA from x'FFFFFFFFFFFF' to determine how much space is left in the log. If APAR PK27611 is applied, you need to use the RBA value of x'FFFF00000000' for this calculation.

You can use the output for the print log map utility to determine how many archive logs are created on an average day. This number multiplied by the RBA range of the archive log data sets (ENDRBA minus STARTRBA) provides the average number of bytes that are logged per day. Divide this value into the space remaining in the log to determine approximately how much time is left before the end of the log RBA range is reached. If there is less than one year remaining before the end of the log RBA range is reached, start planning to reset the log RBA value. If less than three months remain before the end of the log RBA range is reached, you need to take immediate action to reset the log RBA value.

Log RBA range

The log RBA is an ever-increasing 6 byte hex value that starts as 0 (zero) when the DB2 subsystem is first installed and increases to a maximum value of x'FFFFFFFFFFFF' (2 to the 48th).

The rate at which the log RBA value increases through this range depends on the logging rate of the DB2 subsystem. In cases where a heavy logging rate is sustained over a period of years, the log RBA value can begin to approach the end of the range.

Before the DB2 subsystem reaches the end of the log RBA range, you need to reset the log RBA value. The process that you complete to reset the log RBA value depends on whether the DB2 subsystem is the member of a data sharing group or in a non-data sharing environment.

Related tasks

“Resetting the log RBA value in a data sharing environment”

“Resetting the log RBA value in a non-data sharing environment”

Resetting the log RBA value in a data sharing environment

Before the member of a data sharing group reaches the end of the log RBA range, you need to reset the log RBA value for that member.

To reset the log RBA value in a data sharing environment:

1. Issue the STOP DB2 command to quiesce the member that is approaching the end of the log RBA range.
2. Restart this member in ACCESS(MAINT) mode.
3. Issue the -DISPLAY THREAD command. Ensure that there are no INDOUBT or POSTPONED ABORT units of recovery.
4. Issue the -DISPLAY DATABASE(*) SPACENAM(*) RESTRICT command. Ensure that all restricted states are removed.
5. Quiesce the member again by issuing the -STOP DB2 command.
6. Optional: Start a new member to take over the work of the member that is quiesced. If this is an acceptable solution, you can leave the original member stopped indefinitely.
7. To bring the original member back into the data sharing group, you must cold start the member with a STARTRBA of 0 (zero). To cold start the member:
 - a. Before restarting the member, apply the PTF for APAR PK22872.
 - b. Make a full image copy of all data. To make a full image copy, the member might need to remain quiesced for a period of time. The length of time depends on the size of the databases. After all of the data has been image copied, you no longer need the member logs for recovery.
 - c. Cold start this member back to the RBA value of 0 (zero). This step removes all log data from the BSDS, and you can use the member again. This step requires utility DSNJU003 with the following options:

```
CRESTART CREATE,STARTRBA=0,ENDRBA=0
```

Related concepts

“Log RBA range” on page 492

Resetting the log RBA value in a non-data sharing environment

Before the DB2 subsystem in a non-data sharing environment reaches the end of the log RBA range, you need to reset the log RBA value for that subsystem.

To reset the log RBA value in a non-data sharing environment:

1. Issue the STOP DB2 command to quiesce the subsystem that is approaching the end of the log RBA range.
2. Restart DB2 in ACCESS(MAINT) mode.
3. Issue the -DISPLAY THREAD command. Ensure that there are no INDOUBT or POSTPONED ABORT units of recovery.
4. Issue the -DISPLAY UTILITY command. Ensure there are no active or stopped utilities.
5. Issue the -DISPLAY DATABASE(*) SPACENAM(*) RESTRICT command. Ensure that all restricted states are removed.

6. Quiesce the DB2 subsystem again by issuing the -STOP DB2 command.
7. Use IDCAMS to delete and redefine the table spaces SYSUTILX, SYSCOPY, and SYSLGRNX and their corresponding indexes. Then, re-initialize these page sets. For information about how to re-initialize these page sets, see member DSNTIJID in library SDSNSAMP.
8. Determine whether you can use the COPY utility, and complete the appropriate task as follows:
 - If the PTF for APAR PK28576 is applied, or can be applied, and you can use the COPY utility to make copies of all data, apply the PTF for APAR PK28576. This APAR enables the COPY utility to reset the RBA values in pages as they are copied. See the documentation for the PTF for information about enabling the COPY utility function that is provided by this APAR.
 - If you cannot use the COPY utility, do the following:
 - a. Use DSN1COPY to copy every table space and partition to another data set. If indexes are not included in this process, you must rebuild the indexes after restarting DB2.
 - b. Use DSN1COPY with the RESET parameter to copy each table space or partition to its original data set. This process resets the RBA values in all of the page sets. If indexes are not included in this process, you must rebuild the indexes after restarting DB2.
9. Cold start this subsystem back to the RBA value of 0 (zero) by using ACCESS(MAINT). This step removes all log data from the BSDS. This step requires utility DSNJU003 with the following options:
CRESTART CREATE, STARTRBA=0, ENDRBA=0
10. If you did not reset the indexes by using DSN1COPY as specified in step 8, rebuild all indexes. Start by rebuilding the catalog and directory indexes, and then rebuild the indexes on user data.
11. Take new, full image copies of all data. If you applied the PTF for APAR PK28576, run the COPY utility with option SHRLEVEL REFERENCE to automatically reset the RBA values in all of the page sets. This step can be performed in parallel with the previous step after the catalog and directory indexes are rebuilt.
12. Stop DB2. If applicable, disable the reset RBA function in the COPY utility, and restart DB2 for normal access.

Related concepts

“Log RBA range” on page 492

Canceling and restarting an offload

If the offload task remains stalled, the active logs eventually become full and DB2 stops database update activity.

 Issue the ARCHIVE LOG CANCEL OFFLOAD command to cancel and restart the offload task. 

Displaying the status of an offload

To view the status of the offload task, issue the DISPLAY LOG command.

Discarding archive log records

You must keep enough log records to recover units of work and databases.

To recover units of recovery, you need log records at least until all current actions are completed. If DB2 terminates abnormally, restart requires all log records since the previous checkpoint or the beginning of the oldest UR that was active at the abend, whichever is first on the log.

To tell whether all units of recovery are complete, read the status counts in the DB2 restart messages. If all counts are zero, no unit-of-recovery actions are pending. If indoubt units of recovery remain, identify and recover them by the methods described in Chapter 15, “Monitoring and controlling DB2 and its connections,” on page 413.

To recover databases, you need log records and image copies of table spaces. How long you keep log records depends, on how often you make those image copies. If you do not already know what records you want to keep, see Chapter 19, “Backing up and recovering your data,” on page 531 for suggestions about recovery cycles.

Locating archive log data sets

You must keep all the logs since the most recent checkpoint of DB2, so that DB2 can restart. You also must keep all the logs for two or more complete image copy cycles of your least-frequently copied table space.

What, then, can you discard?

You need an answer in terms of the log RBA ranges of the archive data sets. The earliest log record that you want is identified by a log RBA. You can discard any archive log data sets that contain only records with log RBAs less than that.

GUIP To locate archive log data sets:

1. Resolve indoubt units of recovery. If DB2 is running with TSO, continue with Step 2. If DB2 is running with IMS, CICS, or distributed data, the following procedure applies:
 - a. The period between one startup and the next must be free of any indoubt units of recovery. Ensure that no DB2 activity is going on until you finish this procedure. (Consider planning this procedure for a non-prime shift, for minimum impact on users.) To determine whether indoubt units exist, issue the following DB2 command:

```
DISPLAY THREAD TYPE(INDOUBT)
```

If you find no indoubt units of recovery, skip to Step 2.
 - b. If one or more indoubt units of recovery exist, take one of the following actions:
 - Start IMS or CICS, causing that subsystem to resolve the indoubt units of recovery. If the thread is a distributed indoubt unit of recovery, restart the distributed data facility (DDF) to resolve the unit of work. If DDF does not start or cannot resolve the unit of work, issue the following command to resolve the unit of work:

```
RECOVER INDOUBT
```
 - Issue the following DB2 command:

```
RECOVER INDOUBT
```
 - c. Re-issue the command `DISPLAY THREAD TYPE(INDOUBT)` to ensure that the indoubt units have been recovered. When none remain, continue with 2 on page 496.

2. Find the startup log RBA. Keep at least all log records with log RBAs greater than the one given in this message, issued at restart:

```
DSNR003I RESTART...PRIOR CHECKPOINT RBA=XXXXXXXXXXXX
```

If you suspended DB2 activity while performing step 1, you can restart it now.

3. Find the minimum log RBA needed. Suppose that you have determined to keep some number of complete image copy cycles of your least-frequently copied table space. You now need to find the log RBA of the earliest full image copy that you want to keep.
 - a. If you have any table spaces that were so recently created that no full image copies of them have ever been taken, take full image copies of them. If you do not take image copies of them, and you discard the archive logs that log their creation, DB2 can never recover them.

GUPI The following SQL statement lists table spaces that have no full image copy:

```
SELECT X.DBNAME, X.NAME, X.CREATOR, X.NTABLES, X.PARTITIONS
FROM SYSIBM.SYSTABLESPACE X
WHERE NOT EXISTS (SELECT * FROM SYSIBM.SYSCOPY Y
                  WHERE X.NAME = Y.TSNAME
                  AND X.DBNAME = Y.DBNAME
                  AND Y.ICTYPE = 'F')
ORDER BY 1, 3, 2;
```

GUPI

- b. Issue the following SQL statement to find START_RBA values:

GUPI

```
SELECT DBNAME, TSNAME, DSNUM, ICTYPE, ICDATE, HEX(START_RBA)
FROM SYSIBM.SYSCOPY
ORDER BY DBNAME, TSNAME, DSNUM, ICDATE;
```

GUPI

The statement lists all databases and table spaces within them, in ascending order by date.

- c. Find the START_RBA for the earliest full image copy (ICTYPE=F) that you intend to keep. If your least-frequently copied table space is partitioned, and you take full image copies by partition, use the earliest date for all the partitions.

If you are going to discard records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX, note the date of the earliest image copy that you want to keep.

4. Copy catalog and directory tables. Take full image copies of the DB2 table spaces that are listed in the following table to ensure that copies of these table spaces are included in the range of log records that you will keep.

Table 102. Catalog and directory tables to copy

Database name	Table space names	
DSNDB01	DBD01	SYSUTILX
	SCT02	SYSLGRNX
	SPT01	

Table 102. Catalog and directory tables to copy (continued)

Database name	Table space names	
DSNDB06	SYSCOPY	SYSPLAN
	SYSDBASE	SYSSTATS
	SYSDBAUT	SYSSTR
	SYSGPAUT	SYSUSER
	SYSGROUP	SYSVIEWS
	SYSPKAGE	

5. Locate and discard archive log volumes. Now that you know the minimum LOGRBA, from step 3, suppose that you want to find archive log volumes that contain only log records earlier than that. Proceed as follows:
 - a. Execute the print log map utility to print the contents of the BSDS.
 - b. Find the sections of the output titled "ARCHIVE LOG COPY n DATA SETS". (If you use dual logging, two sections exist.) The columns labeled STARTRBA and ENDRBA show the range of log RBAs that are contained in each volume. Find the volumes (two, for dual logging) whose ranges include the minimum log RBA that you found in step 3; these are the earliest volumes that you need to keep.

If no volumes have an appropriate range, one of these cases applies:

- The minimum LOGRBA has not yet been archived, and you can discard all archive log volumes.
- The list of archive log volumes in the BSDS wrapped around when the number of volumes exceeded the number that is allowed by the RECORDING MAX field of installation panel DSNTIPA. If the BSDS does not register an archive log volume, it can never be used for recovery. Therefore, consider adding information about existing volumes to the BSDS.

Also, consider increasing the value of MAXARCH.

- c. Delete any archive log data set or volume (both copies, for dual logging) whose ENDRBA value is less than the STARTRBA value of the earliest volume that you want to keep.

Because BSDS entries wrap around, the first few entries in the BSDS archive log section might be more recent than the entries at the bottom. Look at the combination of date and time to compare age. Do not assume that you can discard all entries above the entry for the archive log that contains the minimum LOGRBA.

- d. Delete the data sets. If the archives are on tape, scratch the tapes; if they are on disks, run a z/OS utility to delete each data set. Then, if you want the BSDS to list only existing archive volumes, use the change log inventory utility to delete entries for the discarded volumes. 

Related tasks

"Resolving indoubt units of recovery" on page 522

Related reference

"DSNJU004 (print log map)" (DB2 Utility Guide and Reference)

"Archive log data set parameters panel: DSNTIPA" (DB2 Installation Guide)

Management of the bootstrap data set

The bootstrap data set (BSDS) is a VSAM key-sequenced data set that contains information about the log data sets and the records those data sets include. The BSDS also contains information about buffer pool attributes.

The BSDS is defined with access method services when DB2 is installed and is allocated by a DD statement in the DB2 startup procedure. It is deallocated when DB2 terminates.

The active logs are first registered in the BSDS by job DSNTIJD, during DB2 installation. They cannot be replaced, nor can new ones be added, without terminating and restarting DB2.

Archive log data sets are dynamically allocated. When one is allocated, the data set name is registered in the BSDS in separate entries for each volume on which the archive log resides. The list of archive log data sets expands as archives are added, and the list wraps around when a user-determined number of entries is reached. The maximum number of archive log data sets are 1000 for single archive logging and 2000 for dual archive logging.

You can manage the inventory of archive log data sets with the change log inventory utility (DSNJU003).

A wide variety of tape management systems exist, along with the opportunity for external manual overrides of retention periods. Because of that, DB2 does not have an automated method to delete the archive log data sets from the BSDS inventory of archive log data sets. Thus, the information about an archive log data set can be in the BSDS long after the archive log data set is scratched by a tape management system following the expiration of the retention period of the data set.

Conversely, the maximum number of archive log data sets might be exceeded, and the data from the BSDS might be dropped long before the data set reaches its expiration date.

If you specified at installation that archive log data sets are to be cataloged when allocated, the BSDS points to the integrated catalog facility catalog for the information that is needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocation.

Related reference

"DSNJCNVB" (DB2 Utility Guide and Reference)

Restoring dual mode BSDS

Normally, DB2 keeps duplicate copies of the BSDS. If an I/O error occurs, DB2 deallocates the failing copy and continues with a single BSDS. However, you can restore the dual mode.

To restore dual mode:

1. Use access method services to rename or delete the failing BSDS.
2. Define a new BSDS with the same name as the deleted BSDS.
3. Issue the DB2 command RECOVER BSDS to make a copy of the good BSDS in the newly allocated data set.

BSDS copies with archive log data sets

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first file on the first output volume. If the archive log is on disk, the BSDS copy is a separate file, which could reside on a separate volume.

Recommendation: For better offload performance and space utilization, use a block size of 28672 for tape or 24576 for disk. (The default value is 24576.) If required, you can change this value in the BLOCK SIZE field on installation panel DSN TIP A. Also adjust the PRIMARY QUANTITY and SECONDARY QUANTITY fields to reflect any changes in block size.

The data set names of the BSDS copy and the archive log are the same, except that the first character of the last data set name qualifier in the BSDS name is B instead of A, as in the following example:

Archive log name

DSNCAT.ARCHLOG1.A0000001

BSDS copy name

DSNCAT.ARCHLOG1.B0000001

If a read error occurs while copying the BSDS, the copy is not created. Message DSNJ125I is issued, and the offload to the new archive log data set continues without the BSDS copy.

The utility DSNJU004, print log map, lists the information that is stored in the BSDS.

Related reference

"DSNJU004 (print log map)" (DB2 Utility Guide and Reference)

Recommendations for changing the BSDS log inventory

You do not need to take special steps to keep the BSDS updated with records of logging events. DB2 does that automatically.

However, you might want to change the BSDS if you:

- Add more active log data sets.
- Copy active log data sets to newly allocated data sets, as when providing larger active log allocations.
- Move log data sets to other devices.
- Recover a damaged BSDS.
- Discard outdated archive log data sets.
- Create or cancel control records for conditional restart.
- Add or change the DDF communication record.

You can change the BSDS by running the DB2 batch change log inventory (DSNJU003) utility. Do not run this utility when DB2 is active. Run it only when DB2 is inactive, or inconsistent results might result.

You can copy an active log data set using the access method services IDCAMS REPRO statement. The copy can be performed when only DB2 is down, because DB2 allocates the active log data sets as exclusive (DISP=OLD) at DB2 startup. For

more information about the REPRO statement, see *DFSMS/MVS: Access Method Services for the Integrated Catalog* and *z/OS DFSMS Access Method Services for Catalogs*.

Related reference

"DSNJU003 (change log inventory)" (DB2 Utility Guide and Reference)

Chapter 17. Restarting DB2 after termination

This section explains what to expect when DB2 terminates normally or abnormally, and how to start it again.

These concepts are important background for backup and recovery and for maintaining consistency. There are additional considerations for a DB2 subsystem that must be kept consistent with some other system.

The term *object*, used in any discussion of restarting DB2 after termination, refers to any database, table space, or index space.

Related concepts

Chapter 18, “Maintaining consistency across multiple systems,” on page 515

Related tasks

Chapter 19, “Backing up and recovering your data,” on page 531

Methods of restarting

DB2 can restart in several different ways. Some options are based on how DB2 terminated or what your environment is.

Types of termination

DB2 terminates normally in response to the command STOP DB2. If DB2 stops for any other reason, the termination is considered abnormal.

Normal termination

In a normal termination, DB2 stops all activity in an orderly way.

GUPI You can use either STOP DB2 MODE (QUIESCE) or STOP DB2 MODE (FORCE). The effects each command are compared in the following table.

Table 103. Termination using QUIESCE and FORCE

Thread type	QUIESCE	FORCE
Active threads	Run to completion	Roll back
New threads	Permitted	Not permitted
New connections	Not permitted	Not permitted

You can use either command to prevent new applications from connecting to DB2.

When you issue the STOP DB2 MODE(QUIESCE) command, current threads can run to completion, and new threads can be allocated to an application that is running.

With IMS and CICS, STOP DB2 MODE(QUIESCE) allows a current thread to run only to the end of the unit of recovery, unless either of the following conditions are true:

- Open, held cursors exist.
- Special registers are not in their original state.

Before DB2 can stop, all held cursors must be closed and all special registers must be in their original state, or the transaction must complete.

With CICS, QUIESCE mode stops the CICS attachment facility, so an active task might not necessarily run to completion.

For example, assume that a CICS transaction opens no cursors that are declared WITH HOLD and modifies no special registers, as follows:

```
EXEC SQL
      ← -STOP DB2 MODE(QUIESCE) issued here
:
:
SYNCPPOINT
:
EXEC SQL ← This receives an AETA abend
```

The thread is allowed to run only through the first SYNCPPOINT.

When you issue the command STOP DB2 MODE(FORCE), no new threads are allocated, and work on existing threads is rolled back.

A data object might be left in an inconsistent state, even after a shutdown with mode QUIESCE, if it was made unavailable by the command STOP DATABASE, or if DB2 recognized a problem with the object. MODE (QUIESCE) does not wait for asynchronous tasks that are not associated with any thread to complete before it stops DB2. This can result in data commands such as STOP DATABASE and START DATABASE having outstanding units of recovery when DB2 stops. These outstanding units of recovery become inflight units of recovery when DB2 is restarted; then they are returned to their original states. 

Abnormal terminations (abends)

An abnormal termination, or *abend*, is said to happen when DB2 does not terminate in an orderly way.

An abend can leave data in an inconsistent state for any of the following reasons:

- Units of recovery might be interrupted before reaching a point of consistency.
- Committed data might not be written to external media.
- Uncommitted data might be written to external media.

Normal restart and recovery

DB2 uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when restarting. The BSDS identifies the active and archive log data sets, the location of the most recent DB2 checkpoint on the log, and the high-level qualifier of the integrated catalog facility catalog name.

After DB2 is initialized, the restart process goes through four phases, which are described in the following sections:

- “Phase 1: Log initialization” on page 503
- “Phase 2: Current status rebuild” on page 504
- “Phase 3: Forward log recovery” on page 505
- “Phase 4: Backward log recovery” on page 506

The terms *inflight*, *indoubt*, *in-commit*, and *in-abort* refer to statuses of a unit of work that is coordinated between DB2 and another system, such as CICS, IMS, or a remote DBMS. For definitions of those terms, see “Consistency after termination or failure” on page 520.

At the end of the fourth phase recovery, a checkpoint is taken and committed changes are reflected in the data.

Application programs that do not commit often enough cause long-running units of recovery (URs). These long-running URs might be inflight after a DB2 failure. Inflight URs can extend DB2 restart time. You can restart DB2 more quickly by postponing the backout of long-running URs. Installation options LIMIT BACKOUT and BACKOUT DURATION establish what work to delay during restart.

If your DB2 subsystem has the UR checkpoint count option enabled, DB2 generates console message DSNR035I and trace records for IFCID 0313 to inform you about long-running URs. The UR checkpoint count option is enabled at installation time, through field UR CHECK FREQ on panel DSNTIPL.

If your DB2 subsystem has the UR log threshold option enabled, DB2 generates console message DSNB260I when an inflight UR writes more than the installation-defined number of log records. DB2 also generates trace records for IFCID 0313 to inform you about these long-running URs. The UR log threshold option is established at installation time, through field UR LOG WRITE CHECK on panel DSNTIPL.

Restart of large object (LOB) table spaces is like restart of other table spaces. LOB table spaces that are defined with LOG NO do not log LOB data, but they log enough control information (and follow a force-at-commit policy) so that they can restart without loss of data integrity.

After DB2 has gone through a group or normal restart that involves group buffer pool (GBP) failure, group buffer pool recovery pending (GRECP) can be automatically initiated for all objects except the object that is explicitly deferred during restart (ZPARM defer), or the object that is associated with the indoubt or postponed-abort UR.

Related reference

"Active log data set parameters: DSNTIPL" (DB2 Installation Guide)

Phase 1: Log initialization

During the first restart phase, DB2 attempts to locate the last log RBA that was written before termination. Logging continues at the next log RBA after that.

In phase 1:

1. DB2 compares the high-level qualifier of the integrated catalog facility catalog name, in the BSDS, with the corresponding qualifier of the name in the current subsystem parameter module (DSNZPxxx).

- If they are equal, processing continues with step 2.
- If they are not equal, DB2 terminates with this message:

```
DSNJ130I ICF CATALOG NAME IN BSDS
        DOES NOT AGREE WITH DSNZPARM.
        BSDS CATALOG NAME=aaaaa,
        DSNZPARM CATALOG NAME=bbbbbb
```

Without the check, the next DB2 session could conceivably update an entirely different catalog and set of table spaces. If the check fails, you probably have the wrong parameter module. Start DB2 with the command START DB2 PARM(*module-name*), and name the correct module.

2. DB2 checks the consistency of the timestamps in the BSDS.

- If both copies of the BSDS are current, DB2 tests whether the two timestamps are equal.
 - If they are equal, processing continues with step 3.
 - If they are not equal, DB2 issues message DSNJ120I and terminates. That can happen when the two copies of the BSDS are maintained on separate disk volumes (as recommended) and one of the volumes is restored while DB2 is stopped. DB2 detects the situation at restart.

To recover, copy the BSDS with the latest timestamp to the BSDS on the restored volume. Also recover any active log data sets on the restored volume, by copying the dual copy of the active log data sets onto the restored volume.

- If one copy of the BSDS was deallocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, DB2 might not detect the restoration. In that case, log records that are not noted in the BSDS would be unknown to the system.
3. DB2 finds in the BSDS the log RBA of the last log record that was written before termination.

The highest RBA field (as shown in the output of the print log map utility) is updated only when the following events occur:

 - When DB2 is stopped normally (STOP DB2).
 - When active log writing is switched from one data set to another.
 - When DB2 has reached the end of the log output buffer. The size of this buffer is determined by the OUTPUT BUFFER field of installation panel DSNTIPL.
 4. DB2 scans the log forward, beginning at the log RBA of the most recent log record, up to the last control interval (CI) that was written before termination.
 5. DB2 prepares to continue writing log records at the next CI on the log.
 6. DB2 issues message DSNJ099I, which identifies the log RBA at which logging continues for the current DB2 session. That message signals the end of the log initialization phase of restart.

Related troubleshooting information

“Recovering from BSDS failures” on page 621

Related reference

“Active log data set parameters: DSNTIPL” (DB2 Installation Guide)

Phase 2: Current status rebuild

During the second restart phase, DB2 determines the statuses of objects at the time of termination. By the end of the phase, DB2 has determined whether any units of recovery were interrupted by the termination.

In phase 2:

1. DB2 checks the BSDS to find the log RBA of the last complete checkpoint before termination.
2. DB2 processes the RESTART or DEFER option of the parameter module of the START DB2 command if any exist. The default is always RESTART ALL.
3. DB2 reads every log record from that checkpoint up to the end of the log (which was located during phase 1), and identifies:
 - All exception conditions that exist for each database and all image copy information that is related to the DSNDB01.SYSUTILX, DSNDB01.DBD01, and DSNDB06.SYSCOPY table spaces.

- All objects that are open at the time of termination.
- How far back in the log to go to reconstruct data pages that were not written to disk.

The number of log records that are written between one checkpoint and the next is set when DB2 is installed.

You can temporarily modify the checkpoint frequency by using the command SET LOG. The value that you specify persists while DB2 is active; on restart, DB2 uses the value that is specified in the CHECKPOINT_FREQ field of installation panel DSNTIPL.

4. DB2 issues message DSNR004I, which summarizes the activity that is required at restart for outstanding units of recovery.
5. DB2 issues message DSNR007I if any outstanding units of recovery are discovered. The message includes, for each outstanding unit of recovery, its connection type, connection ID, correlation ID, authorization ID, plan name, status, log RBA of the beginning of the unit of recovery (URID), and the date and time of its creation.

During phase 2, no database changes are made, nor are any units of recovery completed. DB2 determines what processing is required by phase 3, forward log recovery, before access to databases is allowed.

Related reference

"Active log data set parameters: DSNTIPL" (DB2 Installation Guide)

"-SET LOG (DB2)" (DB2 Command Reference)

Phase 3: Forward log recovery

During the third restart phase, DB2 completes the processing for all committed changes and database write operations.

DB2 processing during phase 3 includes:

- Making all database changes for each indoubt unit of recovery and locking the data to prevent access to it after restart
- Making database changes for inflight and in-abort units of recovery
- In the case of group restarts in which locks have been lost, acquiring locks to prevent access to data that is in use by those units of recovery

DB2 can use the fast log apply process to enhance performance of the forward recovery phase. See the Log Apply Storage field on panel DSNTIPL in *DB2 Installation Guide* for details about defining storage for the sort that is used in fast log apply processing. DB2 executes these steps:

1. DB2 detects whether a page set that is being recovered is at the same level ID as it was when the page set was last closed. If it is not, DB2 issues message DSNB232I and places the pages for that object on the logical page list (LPL). DB2 does not restart that object. In this case, you must recover from the down-level page set by using one of the methods described in "Recovering from a down-level page set problem" on page 662.
2. DB2 scans the log forward, beginning at the lowest (earliest) log RBA that is either required for completion of database writes or that is associated with the "Begin Unit of Recovery" of units of recovery that require locks.

That log RBA is determined during phase 2. REDO log records for all units of recovery are processed in this phase.

3. DB2 uses the log RBA of the earliest potential REDO log record for each object (determined during phase 2). All earlier changes to the object have been written to disk; therefore, DB2 ignores their log records.
4. DB2 reads the data or index page for each remaining REDO log record. The page header registers the log RBA of the record of the last change to the page.
 - If the log RBA of the page header is greater than or equal to that of the current log record, the logged change has already been made and written to disk, and the log record is ignored.
 - If the log RBA in the page header is less than that of the current log record, the change has not been made; DB2 makes the change to the page in the buffer pool.
5. DB2 writes pages to disk as the need for buffers demands it.
6. DB2 marks the completion of each unit of recovery that is processed. If restart processing terminates later, those units of recovery do not reappear in status lists.
7. DB2 stops scanning at the current end of the log.
8. DB2 writes to disk all modified buffers that are not yet written.
9. DB2 issues message DSNR005I, which summarizes the number of remaining in-commit or indoubt units of recovery. No in-commit units of recovery should exist because all processing for these units should have completed. The number of indoubt units of recovery should be equal to the number that is specified in the previous DSNR004I restart message.
10. DB2 issues message DSNR007I, which identifies any outstanding unit of recovery that still must be processed.

If DB2 encounters a problem while applying log records to an object during phase 3, the affected pages are placed in the logical page list. Message DSNI001I is issued once per page set or partition, and message DSNB250E is issued once per page. Restart processing continues.

DB2 issues status message DSNR031I periodically during this phase.

Phase 4: Backward log recovery

During the fourth restart phase, DB2 changes performed for inflight or in-abort units of recovery are reversed.

In phase 4:

1. DB2 scans the log backward, starting at the current end. The scan continues until the earliest "Begin Unit of Recovery" record for any outstanding inflight or in-abort unit of recovery.

If you specified limited backout processing, the scan might stop prematurely (however, DB2 verifies that all catalog and directory changes are completely backed out). In this case, the scan completes after DB2 receives the RECOVER POSTPONED command. You can have DB2 automatically issue this command after restart by specifying the AUTO option for limited backout processing, or you can issue this command manually.
2. DB2 reads the data or index page for each remaining undo log record. The page header registers the log RBA of the record of the last change to the page.
 - If the log RBA of the page header is greater than or equal to that of the current log record, the logged change has already been made and written to disk, so DB2 reverses it.
 - If the log RBA in the page header is less than that of the current log record, the change has not been made and DB2 ignores it.

3. DB2 writes redo compensation information in the log for each undo log record, as it does when backing out a unit of recovery. The redo records describe the reversal of the change and facilitate media recovery. They are written under all circumstances, even when:
 - The disk version of the data did not need to be reversed.
 - The page set has pages on the LPL.
 - An I/O error occurred on the disk version of the data.
 - The disk version of the data could not be allocated or opened.
 - The page set is deferred at restart using the DEFER subsystem parameter.
4. DB2 writes pages to disk as the need for buffers demands it.
5. DB2 writes to disk all modified buffers that have not yet been written.
6. DB2 issues message DSNR006I, which summarizes the number of remaining inflight, in-abort, and postponed-abort units of recovery. The number of inflight and in-abort units of recovery should be zero; the number of postponed-abort units of recovery might not be zero.
7. DB2 marks the completion of each completed unit of recovery in the log so that, if restart processing terminates, the unit of recovery is not processed again at the next restart.
8. If necessary, DB2 reacquires write claims for the objects on behalf of the indoubt and postponed-abort units of recovery.
9. DB2 takes a checkpoint after all database writes have been completed.

If DB2 encounters a problem while applying a log record to an object during phase 4, the affected pages are placed in the logical page list. Message DSNI001I is issued once per page set or partition, and message DSNB250E is issued once per page. Restart processing continues.

DB2 issues status message DSNR031I periodically during this phase.

Automatic restart

If you run DB2 in a Sysplex, you can have the automatic restart function of z/OS automatically restart DB2 or IRLM after a failure.

When DB2 or IRLM stops abnormally, z/OS determines whether z/OS failed too, and where DB2 or IRLM should be restarted. It then restarts DB2 or IRLM.

You must have DB2 installed with a command prefix scope of S to take advantage of automatic restart.

Restart in a data sharing environment

In a data sharing environment, restart processing is expanded to handle the coordination of data recovery across more than one DB2 subsystem.

When certain critical resources are lost, restart includes additional processing to recovery and rebuild those resources. This process is called *group restart*.

Related information

"Group restart phases" (DB2 Data Sharing: Planning and Administration)

Restart implications for table spaces that are not logged

Even if all tables that are modified by a given transaction reside in table spaces that are not logged, a unit of recovery is still established before any of those updates are performed. Undo processing continues to read the log in the backward

direction looking for undo log records that need to be applied until it detects the beginning of this unit of recovery as recorded on the log.

Therefore you still need to perform frequent commits to limit the distance that undo processing might need to go backward on the log to find the beginning of the unit of recovery. If a transaction that is not logged does not do frequent commits, it is subject to being reported as a long-running unit of recovery in message DSNR035I. This means that if such a unit of recovery persists for a duration that qualifies as a long-running unit of recovery, message DSNR035I is issued to identify it.

If, during restart, you need to undo work that has not been logged because of the NOT LOGGED attribute, the table space loses its data integrity, and therefore must be recovered. Recovery can be accomplished by using the RECOVER utility or by reinserting the appropriate data. For example, a summary table can be re-created by one or more INSERT statements; a materialized query table can be rebuilt by using a REFRESH TABLE SQL statement.

To mark the need for recovery, the table space or partition is marked with RECOVER-pending status. To prevent any access to the corrupt data, the table space or partition is placed in the LPL. When undo processing places a table space or partition in the logical page list (LPL) and marks it with RECOVER-pending status, it also places all of the updated indexes on all tables in the table space in the LPL. The corresponding partitions of data-partitioned secondary indexes (DPSIs) are placed in the LPL, which prevents other processes that use index-only access from seeing data whose integrity is in doubt. These indexes are also marked with REBUILD-pending status.

After restart, when DB2 is operational, if undo processing is needed for a unit of recovery in which modifications were made to the table space that was not logged, the entire table space or partition is placed in the LPL, and the table space is marked with RECOVER-pending status. This can happen, for example, as a result of a rollback, abort, trigger error, or duplicate key or referential constraint violation. The LPL ensures that no concurrently running agent can see any of the data whose integrity is in doubt.

Avoid duplicate key or referential constraint violations in table spaces that are not logged because the result is an unavailable table space that requires manual action.

When a table space or partition is placed in the LPL because undo processing is needed for a table space that is not logged, either at restart time or during rollback processing, automatic LPL recovery is not initiated, and a START DATABASE command identifying this table space has no effect on its LPL status.

Related tasks

“Altering table spaces” on page 89

Conditional restart

A *conditional restart* is a DB2 restart that is directed by a user-defined conditional restart control record (CRCR).

If you want to skip some portion of the log processing during DB2 restart, you can use a conditional restart. However, if a conditional restart skips any database change log records, data in the associated objects becomes inconsistent, and any attempt to process them for normal operations might cause unpredictable results.

The only operations that can safely be performed on the objects are recovery to a prior point of consistency, total replacement, or dropping.

In unusual cases, you might choose to make inconsistent objects available for use without recovering them. For example, the only inconsistent object might be a table space that is dropped as soon as DB2 is restarted, or the DB2 subsystem might be used only for testing application programs that are still under development. In cases like those, where data consistency is not critical, normal recovery operations can be partially or fully bypassed by using conditional restart control records in the BSDS.

Related reference

"DSNJU003 (change log inventory)" (DB2 Utility Guide and Reference)

Related information

"Restarting a member with conditions" (DB2 Data Sharing: Planning and Administration)

Restart considerations for identity columns

Cold starts and conditional restarts that skip forward recovery can cause additional data inconsistency within identity columns and sequence objects. After such restarts, DB2 might assign duplicate identity column values and create gaps in identity column sequences.

Terminating DB2 normally

Whenever possible, ensure that DB2 terminates normally.

GUIP

To terminate DB2, issue either of the following commands:

- -STOP DB2 MODE (QUIESCE)
- -STOP DB2 MODE (FORCE)

During shutdown, use the command DISPLAY THREAD to check the shutdown progress. If shutdown is taking too long, you can issue STOP DB2 MODE (FORCE), but rolling back work can take as long as or longer than the completion of QUIESCE.

When stopping in either mode, the following steps occur:

1. Connections end.
2. DB2 ceases to accept commands.
3. DB2 disconnects from the IRLM.
4. The shutdown checkpoint is taken and the BSDS is updated. **GUIP**

Restarting automatically

You control how automatic restart works by using automatic restart policies. When the automatic restart function is active, the default action is to restart the subsystems when they fail. If this default action is not what you want, then you must create a policy that defines the action that you want taken.

To create a policy, you need the element names of the DB2 and IRLM subsystems:

- For a non-data-sharing DB2, the element name is 'DB2\$' concatenated by the subsystem name (DB2\$DB2A, for example). To specify that a DB2 subsystem is not to be restarted after a failure, include RESTART_ATTEMPTS(0) in the policy for that DB2 element.
- For local mode IRLM, the element name is a concatenation of the IRLM subsystem name and the IRLM ID. For global mode IRLM, the element name is a concatenation of the IRLM data sharing group name, the IRLM subsystem name, and the IRLM ID.

For instructions on defining automatic restart policies, see *z/OS MVS Setting Up a Sysplex*.

Deferring restart processing

You have several options for deferring an object's restart processing.

- To vary the device (or volume) on which the objects reside offline:
If the data sets that contain an object are not available, and the object requires recovery during restart, DB2 flags it as stopped and requiring deferred restart. DB2 then restarts without it.

- To delay the backout of a long-running UR:

On installation panel DSNTIPL, you can use the following options:

- LIMIT BACKOUT defined as YES or AUTO indicates that some backout processing is to be postponed when restarting DB2. Issue the RECOVER POSTPONED command to complete the backout processing when the YES option is selected. DB2 does the backout work automatically after DB2 is running and receiving new work when the AUTO option is selected.
- BACKOUT DURATION indicates the number of log records, specified as a multiplier, that are to be read during the backward log scan phase of restart.

The amount of backout processing that is to be postponed is determined by:

- The frequency of checkpoints
- The BACKOUT DURATION installation option
- The characteristics of the inflight and in-abort activity when DB2 stopped

Selecting a limited backout affects log processing during restart. The backward processing of the log proceeds until the oldest inflight or in-abort UR with activity against the catalog or directory is backed out, and until the requested number of log records have been processed.

- To name the object with DEFER when installing DB2:

On installation panel DSNTIPS, you can use the following options:

- DEFER ALL defers restart log apply processing for all objects, including DB2 catalog and directory objects.
- DEFER *list_of_objects* defers restart processing only for objects in the list.

Alternatively, you can specify RESTART *list_of_objects*, which limits restart processing to the list of objects in the list.

DEFER does not affect processing of the log during restart. Therefore, even if you specify DEFER ALL, DB2 still processes the full range of the log for both the forward and backward log recovery phases of restart. However, logged operations are not applied to the data set.

Deferral of restart

Usually, restarting DB2 activates restart processing for objects that were available when DB2 terminated (in other words, not stopped with the STOP DATABASE command). Restart processing applies or backs out log records for objects that have unresolved work.

Restart processing is controlled by specifications on installation panel DSNTIPS, and the default is RESTART ALL.

If some specific object is causing problems, one option is to defer its restart processing by starting DB2 without allowing that object to go through restart processing. When you defer restart of an object, DB2 puts pages that are necessary for restart of the object in the logical page list (LPL). Only those pages are inaccessible; the rest of the object can still be accessed after restart.

There are restrictions to DB2's activation of restart processing for available objects. When DEFER ALL is specified at a site that is designated as RECOVERYSITE in DSNZPxxx, all pages for an object that is deferred are placed in the LPL (as a page range, not as a large list of individual pages). The following conditions apply:

- If DB2 cannot open and read the DBD01 table space it will not put DB2 into ACCESS(MAINT), and DSNX204I is not be issued. Instead either DSNT500I or DSNT501I 'resource unavailable' is issued.
- For a deferred restart scenario that needs to recover all DB2 objects after DB2 is up, it is recommend that you set the ZPARM DEFER ALL and start DB2 with the ACCESS(MAINT) option.
- If DEFER ALL is specified, DSNX204I is not issued.
- With DEFER ALL, DB2 will not open any data sets, including SYSLGRNX and DSNRTSTS, during any phase of restart, and will not attempt to apply any log records.

DB2 can also defer restart processing for particular objects. DB2 puts pages in the LPL for any object (or specific pages of an object) with certain problems, such as an open or I/O error during restart. When an error is encountered while accessing an index object during restart, the entire index is put in the LPL, not just the individual pages.

Performing conditional restart

Normal recovery operations can be partially or fully bypassed by using conditional restart control records in the BSDS.

The procedure for conditional restart is:

1. Optional: When considering a conditional restart, it is often useful to run the DSN1LOGP utility and review a summary report of the information contained in the log.
2. While DB2 is stopped, run the change log inventory utility by using the CRESTART control statement to create a new conditional restart control record.
3. Restart DB2. The recovery operations that take place are governed by the current conditional restart control record.
4. Optional: For data sharing environments, use the LIGHT(YES) parameter on the START DB2 command to quickly recover retained locks on a DB2 member. For more information, see *DB2 Data Sharing: Planning and Administration*.

Related concepts

“Messages at start” on page 365

Options for recovery operations after conditional restart

The recovery operations that take place during restart are controlled by the currently active conditional restart control record. An active control record is created or deactivated by running the change log inventory utility with the CRESTART control statement.

You can choose to:

- Retain a specific portion of the log for future DB2 processing
- Read the log forward to recover indoubt and uncommitted units of recovery
- Read the log backward to back out uncommitted and in-abort units of recovery
- Do a cold start, not processing any log records

A conditional restart record that specifies left truncation of the log causes any postponed-abort units of recovery that began earlier than the truncation RBA to end without resolution. The combination of unresolved postponed-abort units of recovery can cause more records than requested by the BACKODUR system parameter to be processed. The left truncation RBA takes precedence over BACKODUR in this case.

Be careful about doing a conditional restart that discards log records. If the discarded log records contain information from an image copy of the DB2 directory, a future execution of the RECOVER utility on the directory fails.

Conditional restart records

In addition to information describing the active and archive logs, the BSDS contains two queues of records that are associated with conditional restart.

The two queues are:

- A wrap-around queue of conditional restart control records. Each element in the queue records the choices that you made when you created the record and the progress of the restart operation that it controls. When the operation is complete, the use count is set at 1 for the record, and the record is not used again.
- A queue of checkpoint descriptions. Because a conditional restart can specify use of a particular log record range, the recovery process cannot automatically use the most recent checkpoint. The recovery process must find the latest checkpoint within the specified range, and that checkpoint queue is then used for that purpose.

You can use the utility DSN1LOGP to read information about checkpoints and conditional restart control records.

Related reference

“DSN1LOGP” (DB2 Utility Guide and Reference)

Resolving postponed units of recovery

You can postpone some of the backout work that is associated with long-running units of work during system restart by using the LIMIT BACKOUT installation option. By delaying such backout work, the DB2 subsystem can be restarted more quickly.

If you specify `LIMIT BACKOUT = YES`, you must use the `RECOVER POSTPONED` command to resolve postponed units of recovery.

Use the `RECOVER POSTPONED` command to complete postponed backout processing on all units of recovery; you cannot specify a single unit of work for resolution. This command might take several hours to complete depending on the content of the long-running job. In some circumstances, you can elect to use the `CANCEL` option of the `RECOVER POSTPONED` command. This option leaves the objects in an inconsistent state (`REFP`) that you must resolve before using the objects. However, you might choose the `CANCEL` option for the following reasons:

- You determine that the complete recovery of the postponed units of recovery will take more time to complete than you have available. You also determine it is faster to either recover the objects to a prior point in time or run the `LOAD` utility with the `REPLACE` option.
- You want to replace the existing data in the object with new data.
- You decide to drop the object. To drop the object successfully, complete the following steps:
 1. Issue the `RECOVER POSTPONED` command with the `CANCEL` option.
 2. Issue the `DROP TABLESPACE` statement.
- You do not have the DB2 logs to successfully recover the postponed units of recovery.

Related reference

"Active log data set parameters: `DSNTIPL`" (DB2 Installation Guide)

RECOVER POSTPONED command

Output from the `RECOVER POSTPONED` command consists of informational messages.

In the following figure, backout processing was performed against two table space partitions and two index partitions:

```
DSNV435I ! RESOLUTION OF POSTPONED ABORT URS HAS BEEN SCHEDULED
DSN9022I ! DSNVRP 'RECOVER POSTPONED' NORMAL COMPLETION
DSNR047I ! DSNRBMON POSTPONED ABORT BACKOUT
PROCESSING LOG RECORD AT RBA 000002055000 TO RBA 000001E6A20E
DSNR047I ! DSNRBMON POSTPONED ABORT BACKOUT
PROCESSING LOG RECORD AT RBA 000002049000 TO RBA 000001E6A20E
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
FOR PAGESET DSND04 .I PART 00000004.
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
FOR PAGESET DSND04 .PT PART 00000004.
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
FOR PAGESET DSND04 .I PART 00000002.
DSNI024I ! DSNIARPL BACKOUT PROCESSING HAS COMPLETED
FOR PAGESET DSND04 .PT PART 00000002.
```

Figure 50. Example of output from RECOVER POSTPONED processing

If a required page cannot be accessed during `RECOVER POSTPONED` processing, or if any other error is encountered while attempting to apply a log record, the page set or partition is deferred and processing continues. DB2 writes a compensation log record to reflect those deferrals and places the page in the logical page list. Some errors that are encountered during recovery of indexes cause the entire page set to be placed in the logical page list. Some errors halt the construction of the compensation log and mark the page set as `RECP`.

Recovering from an error during RECOVER POSTPONED processing

When an error prevents logging of a compensation log record, DB2 terminates abnormally.

If DB2 terminates abnormally:

1. Fix the error.
2. Restart DB2.
3. Re-issue the RECOVER POSTPONED command if automatic backout processing has not been specified.

If the RECOVER POSTPONED processing lasts for an extended period, the output includes DSNR047I messages to help you monitor backout processing. These messages show the current RBA that is being processed and the target RBA.

Chapter 18. Maintaining consistency across multiple systems

Data consistency issues arise when DB2 acts in conjunction with other systems, such as IMS, CICS, or remote database management systems (DBMS).

Multiple system consistency

DB2 can work with other DBMSs, including IMS, CICS, and other types of remote DBMSs through the distributed data facility (DDF). DB2 can also work with other DB2 subsystems through the DDF.

If data in more than one subsystem is to be consistent, all update operations at all subsystems for a single logical unit of work must either be committed or backed out.

Two-phase commit process

In a distributed system, the actions of a logical unit of work might occur at more than one system. When these actions update recoverable resources, the commit process ensures that either all the effects of the logical unit of work persist or that none of the effects persist, despite component, system, or communications failures.

DB2 uses a two-phase commit process in communicating between subsystems. That process is under the control of one of the subsystems, called the *coordinator*. The other systems involved are the *participants*. DB2, CICS, or WebSphere® Application Server are always the coordinator when they interact with DB2. In transactions that include those systems, DB2 is always a participant. DB2 is always the coordinator when it interacts with TSO, and DB2 completely controls the commit process in these interactions. In interactions with other DBMSs, including other DB2 subsystems, your local DB2 can be either the coordinator or a participant.

The following figure illustrates the two-phase commit process. Events in the coordinator (IMS, CICS, or DB2) are shown on the upper line, events in the participant on the lower line.

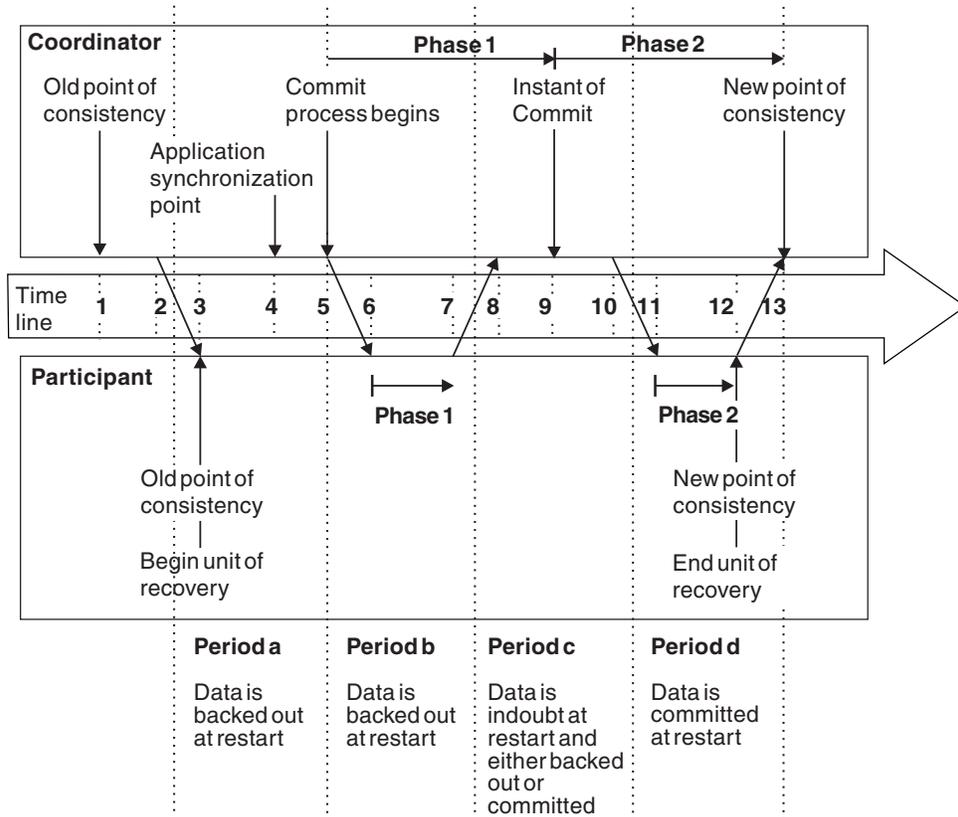


Figure 51. Time line illustrating a commit that is coordinated with another subsystem

The numbers below are keyed to the timeline in the figure. The resultant state of the update operations at the participant are shown between the two lines.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls the participant to update some data, by executing an SQL statement.
3. This starts a unit of recovery in the participant.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. IMS can do that by using a DL/I CHKP call, a fast path SYNC call, a GET UNIQUE call to the I/O PCB, or a normal application termination. CICS uses a SYNCPOINT command or a normal application termination. A DB2 application starts commit processing by an SQL COMMIT statement or by normal termination. Phase 1 of commit processing begins.
6. The coordinator informs the participant that it is to prepare for commit. The participant begins phase 1 processing.
7. The participant successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.
9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes because both have completed phase 1 and could recover from any failure. The coordinator records on its log the instant of commit—the irrevocable decision of the two subsystems to make the changes.

The coordinator now begins phase 2 of the processing—the actual commitment.

10. The coordinator notifies the participant that it can begin phase 2.
11. The participant logs the start of phase 2.
12. Phase 2 is successfully completed, which establishes a new point of consistency for the participant. The participant then notifies the coordinator that it is finished with phase 2.
13. The coordinator finishes its phase 2 processing. The data that is controlled by both subsystems is now consistent and available to other applications.

On some occasions, the coordinator invokes the participant when no participant resource has been altered since the completion of the last commit process. This can happen, for example, when a SYNCPOINT is issued after performance of a series of SELECT statements or when end-of-task is reached immediately after a SYNCPOINT is issued. When this occurs, the participant performs both phases of the two-phase commit during the first commit phase and records that the user or job is read-only at the participant.

Commit coordinator and multiple participants

The principles and methods for maintaining consistency across more than two systems are similar to those that are used to ensure consistency across two systems. The main difference involves the role of a system as coordinator or participant when a unit of work spans multiple systems.

The coordinator of a unit of work that involves two or more other DBMSs must ensure that all systems remain consistent. After the first phase of the two-phase commit process, the DB2 coordinator waits for the other participants to indicate that they can commit the unit of work. If all systems are able, the DB2 coordinator sends the commit decision and each system commits the unit of work.

If even one system indicates that it cannot commit, the DB2 coordinator communicates the decision to roll back the unit of work at all systems. This process ensures that data among multiple DBMSs remains consistent. When DB2 is the participant, it follows the decision of the coordinator, whether the coordinator is another DB2 or another DBMS.

DB2 is always the participant when interacting with IMS, CICS, or WebSphere Application Server systems. However, DB2 can also serve as the coordinator for other DBMSs or for other DB2 subsystems in the same unit of work. For example, if DB2 receives a request from a coordinating system that also requires data manipulation on another system, DB2 propagates the unit of work to the other system and serves as the coordinator for that system.

In the following figure, DB2A is the participant for an IMS transaction, but DB2 becomes the coordinator for the two database servers (AS1 and AS2), for DB2B, and for its respective DB2 servers (DB2C, DB2D, and DB2E).

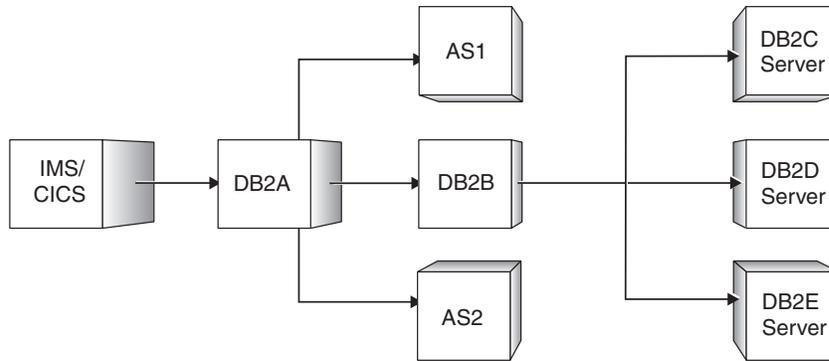


Figure 52. Illustration of multi-site unit of work

If the connection between DB2A and the coordinating IMS system fails, the connection becomes an indoubt thread. However, DB2A connections to the other systems are still waiting and are not considered indoubt. Automatic recovery occurs to resolve the indoubt thread. When the thread is recovered, the unit of work commits or rolls back, and this action is propagated to the other systems that are involved in the unit of work.

Illustration of multi-site update

This illustration shows a multi-site update that involves one coordinator and two participants.

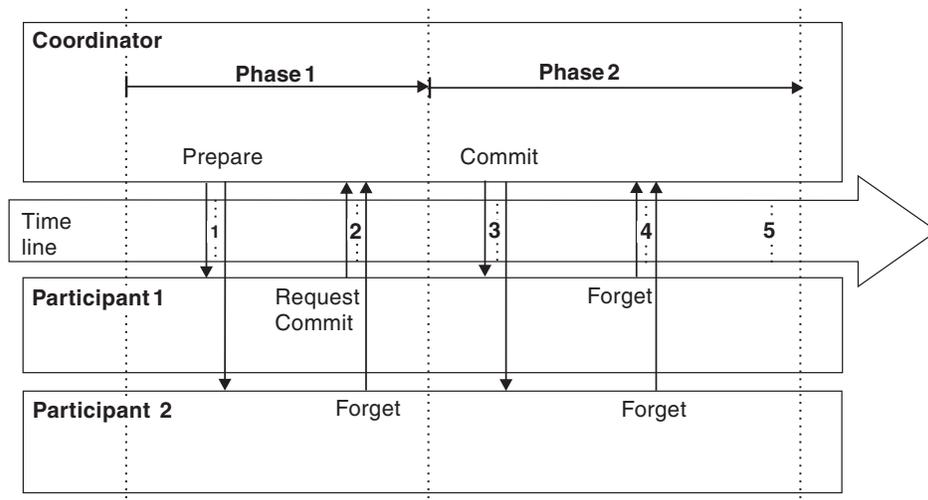


Figure 53. Illustration of multi-site update

The following process describes each action that the figure illustrates.

Phase 1

1. When an application commits a logical unit of work, it signals the DB2 coordinator. The coordinator starts the commit process by sending messages to the participants to determine whether they can commit.
2. A participant (*Participant 1*) that is willing to let the logical unit of work be committed, and which has updated recoverable resources, writes a log record. It then sends a request-commit message to the coordinator

and waits for the final decision (commit or roll back) from the coordinator. The logical unit of work at the participant is now in the prepared state.

If a participant (*Participant 2*) has not updated recoverable resources, it sends a forget message to the coordinator, releases its locks, and forgets about the logical unit of work. A read-only participant writes no log records. The disposition (commit or rollback) of the logical unit of work is irrelevant to the participant.

If a participant wants to have the logical unit of work rolled back, it writes a log record and sends a message to the coordinator. Because a message to roll back acts like a veto, the participant in this case knows that the logical unit of work is to be rolled back by the coordinator. The participant does not need any more information from the coordinator and therefore rolls back the logical unit of work, releases its locks, and forgets about the logical unit of work. (This case is not illustrated in the figure.)

Phase 2

1. After the coordinator receives request-commit or forget messages from all its participants, it starts the second phase of the commit process. If at least one of the responses is request-commit, the coordinator writes a log record and sends committed messages to all the participants who responded to the prepare message with request-commit. If neither the participants nor the coordinator have updated any recoverable resources, no second phase occurs, and no log records are written by the coordinator.

2. Each participant, after receiving a committed message, writes a log record, sends a response to the coordinator, and then commits the logical unit of work.

If any participant responds with a roll back message, the coordinator writes a log record and sends a roll back message to all participants. Each participant, after receiving a roll back message writes a log record, sends an acknowledgment to the coordinator, and then rolls back the logical unit of work. (This case is not illustrated in the figure.)

3. The coordinator, after receiving the responses from all the participants that were sent a message in the second phase, writes an 'end' record and forgets the logical unit of work.

Important: If you try to resolve any indoubt threads manually, you need to know whether the participants committed or rolled back their units of work. With this information, you can make an appropriate decision regarding processing at your site.

Termination for multiple systems

Termination for multiple systems is like termination for single systems, but with some additional considerations.

- Using STOP DB2 MODE(FORCE) could create indoubt units of recovery for threads that are between commit processing phases. These indoubt threads are resolved after you reconnect to the coordinator.
- Data that is updated by an indoubt unit of recovery is locked and unavailable for use by others. The unit could be indoubt when DB2 was stopped, or it could be indoubt from an earlier termination that is not yet resolved.
- A DB2 system failure can leave a unit of recovery in an indoubt state if the failure occurs between phase 1 and phase 2 of the commit process.

Consistency after termination or failure

If a DB2 failure occurs while DB2 is acting as a coordinator, it has the appropriate information to determine commit or roll back decisions after restart. However, if a DB2 failure occurs while DB2 acts as the participant, it must determine after restart whether to commit or to roll back units of recovery that were active at the time of the failure.

For certain units of recovery, DB2 has enough information to make the decision. For others, it does not, and DB2 must get information from the coordinator when the connection is re-established.

The status of a unit of recovery after a termination or failure depends upon the moment at which the incident occurred. The following illustration helps to illustrate the following possible statuses.

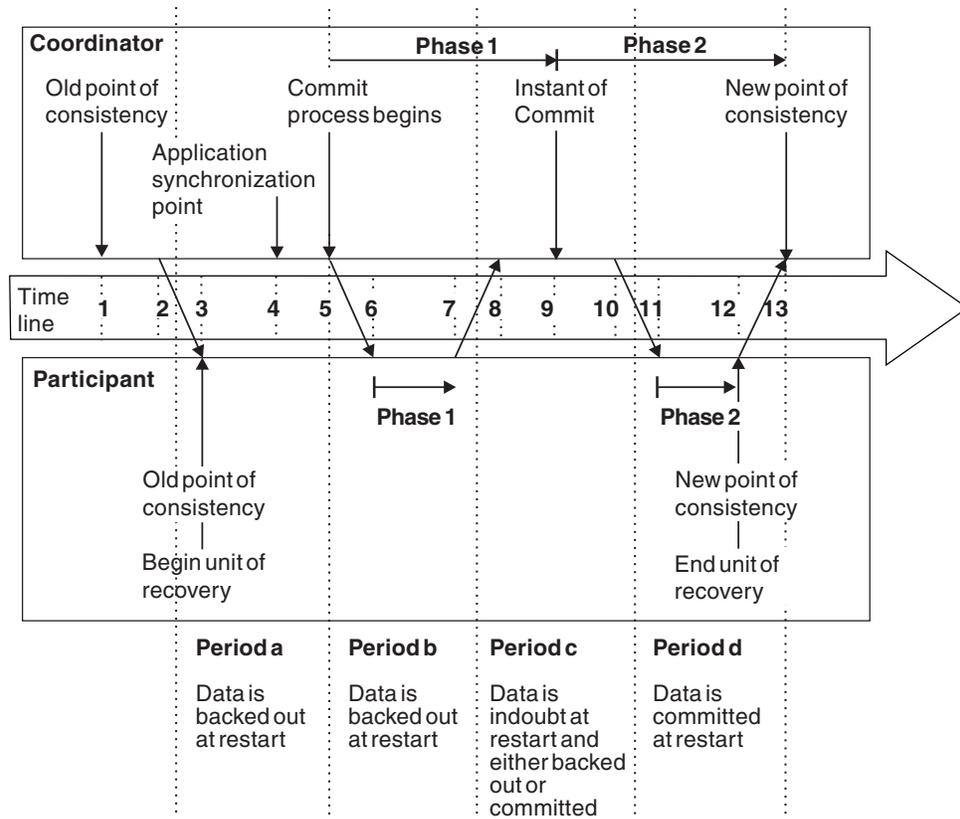


Figure 54. Time line illustrating a commit that is coordinated with another subsystem

Status Description and Processing

Inflight

The participant or coordinator failed before finishing phase 1 (period *a* or *b*); during restart, both systems back out the updates.

Indoubt

The participant failed after finishing phase 1 and before starting phase 2 (period *c*); only the coordinator knows whether the failure happened before or after the commit (point 9). If it happened before, the participant must back out its changes; if it happened afterward, it must make its changes and commit them. After restart, the participant waits for information from the coordinator before processing this unit of recovery.

In-commit

The participant failed after it began its own phase 2 processing (period *d*); it makes committed changes.

In-abort

The participant or coordinator failed after a unit of recovery began to be rolled back but before the process was complete (not shown in the figure). The operational system rolls back the changes; the failed system continues to back out the changes after restart.

postponed-abort

If the LIMIT BACKOUT installation option is set to YES or AUTO, any backout not completed during restart is postponed. The status of the incomplete URs is changed from inflight or in-abort to postponed-abort.

Normal restart and recovery for multiple systems

When DB2 acts together with another system, the recovery log contains information about units of recovery that are inflight, indoubt, in-abort, postponed-abort, or in-commit.

The phases of restart and recovery deal with that information as follows:

Phase 1: Log initialization

This phase proceeds as described in “Phase 1: Log initialization” on page 503.

Phase 2: Current status rebuild

While reading the log, DB2 identifies:

- The coordinator and all participants for every unit of recovery.
- All units of recovery that are outstanding and their statuses (indoubt, in-commit, in-abort, or inflight, as described under “Consistency after termination or failure” on page 520).

Phase 3: Forward log recovery

DB2 makes all database changes for each indoubt unit of recovery and locks the data to prevent access to it after restart. Later, when an indoubt unit of recovery is resolved, processing is completed in one of these ways:

- For the ABORT option of the RECOVER INDOUBT command, DB2 reads and processes the log, reversing all changes.
- For the COMMIT option of the RECOVER INDOUBT command, DB2 reads the log but does not process the records because all changes have been made.

At the end of this phase, indoubt activity is reflected in the database as though the decision was made to commit the activity, but the activity is not yet committed. The data is locked and cannot be used until DB2 recognizes and acts on the indoubt decision. (For a description of indoubt units of recovery, see “Resolving indoubt units of recovery” on page 522.)

Phase 4: Backward log recovery

This phase reverses changes that are performed for inflight or in-abort units of recovery. At the end of this phase, interrupted inflight and in-abort changes are removed from the database (the data is consistent and can be used) or removal of the changes is postponed (the data is inconsistent and unavailable).

If removal of the changes is postponed, the units of recovery become known as *postponed-abort* units of recovery. The data with pending backout

work is in a restrictive state (restart pending) which makes the data unavailable. The data becomes available at completion of backout work or at cold start or conditional restart of DB2.

If the LIMIT BACKOUT system parameter is AUTO, completion of the backout work begins automatically by DB2 when the system accepts new work. If the LIMIT BACKOUT system parameter is YES, completion of the backout work begins when the RECOVER POSTPONED command is issued.

Multiple-system restart with conditions

In some circumstances, you might need to perform a multiple-system conditional restart.

If conditional restart is performed when DB2 is acting together with other systems, the following actions occur:

1. All information about another coordinator and other participants that are known to DB2 is displayed by messages DSNL438I and DSNL439I.
2. This information is purged. Therefore the RECOVER INDOUBT command must be used at the local DB2 when the local location is a participant, and at another DB2 subsystem when the local location is the coordinator.
3. Indoubt database access threads continue to appear as indoubt, and no resynchronization with either a coordinator or a participant is allowed.

Related tasks

“Resolving inconsistencies resulting from a conditional restart” on page 655

Heuristic decisions about whether to commit or abort an indoubt thread

From the DB2 point of view, a decision to commit or roll back an indoubt unit of recovery by any means but the normal resynchronization process is a *heuristic decision*. If you commit or roll back a unit of work and your decision is different from the other system’s decision, data inconsistency occurs. This type of damage is called *heuristic damage*.

If this situation should occur, and your system then updates any data that is involved with the previous unit of work, your data is corrupted and is extremely difficult to correct.

In order to make a correct decision, you must be absolutely sure that the action you take on indoubt units of recovery is the same as the action that the coordinator takes. Validate your decision with the administrator of the other systems that are involved with the logical unit of work.

Resolving indoubt units of recovery

If DB2 loses its connection to another system, it normally attempts to recover all inconsistent objects after restart. The information that is needed to resolve indoubt units of recovery must come from the coordinating system.

Check the console for message DSNR036I for unresolved units of recovery encountered during a checkpoint. This message might occur to remind operators of existing indoubt threads.

Important: If the TCP/IP address that is associated with a DRDA server is subject to change, the domain name of each DRDA server must be defined in the CDB. This allows DB2 to recover from situations where the server's IP address changes prior to successful resynchronization.

Related reference

"DSNR036I" (DB2 Messages)

Resolution of IMS indoubt units of recovery

The resolution of indoubt units of recovery in IMS has no effect on DL/I resources. Because IMS is in control of recovery coordination, DL/I resources are never indoubt.

When IMS restarts, it automatically commits or backs out incomplete DL/I work, based on whether the commit decision was recorded on the IMS log. The existence of indoubt units of recovery does not imply that DL/I records are locked until DB2 connects.

During the current status rebuild phase of DB2 restart, the DB2 participant makes a list of indoubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

When indoubt units of recovery are recovered, the following steps occur:

1. IMS either passes an RRE to the IMS attachment facility to resolve the entry or informs the attachment facility of a cold start. The attachment facility passes the required information to DB2.
2. If DB2 recognizes that an entry is marked by DB2 for commit and by IMS for roll back, it issues message DSNM005I. DB2 issues this message for inconsistencies of this type between DB2 and IMS.
3. The IMS attachment facility passes a return code to IMS, indicating that it should either destroy the RRE (if it was resolved) or keep it (if it was not resolved). The procedure is repeated for each RRE.
4. Finally, if DB2 has any remaining indoubt units of recovery, the attachment facility issues message DSNM004I.

The IMS attachment facility writes all the records that are involved in indoubt processing to the IMS log tape as type X'5501FE'.

For all resolved units of recovery, DB2 updates databases as necessary and releases the corresponding locks. For threads that access offline databases, the resolution is logged and acted on when the database is started.

DB2 maintains locks on indoubt work that was not resolved. This can create a backlog for the system if important locks are being held. You can use the DISPLAY DATABASE LOCKS command to find out which tables and table spaces are locked by indoubt units of recovery. The connection remains active so that you can clean up the IMS RREs. You can then recover the indoubt threads.

Resolve all indoubt units of work unless software or operating problems occur, such as with an IMS cold start. Resolution of indoubt units of recovery from IMS can cause delays in SQL processing. Indoubt resolution by the IMS control region takes place at two times:

- At the start of the connection to DB2, during which resolution is done synchronously

- When a program fails, during which the resolution is done asynchronously

In the first case, SQL processing is prevented in all dependent regions until the indoubt resolution is completed. IMS does not allow connections between IMS dependent regions and DB2 before the indoubt units of recovery are resolved.

Related tasks

“Controlling IMS connections” on page 447

Resolution of CICS indoubt units of recovery

The resolution of indoubt units of recovery has no effect on CICS resources.

CICS is in control of recovery coordination and, when it restarts, CICS automatically commits or backs out each unit of recovery, depending on whether an end-of-unit-of-work log record exists. The existence of indoubt work does not lock CICS resources until DB2 connects.

A process to resolve indoubt units of recovery is initiated during startup of the attachment facility. During this process:

- The attachment facility receives a list of indoubt units of recovery for this connection ID from the DB2 participant and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own list. CICS determines from its own list what action it took for the indoubt unit of recovery.
- For each entry in the list, CICS creates a task for the attachment facility, specifying the final commit or abort direction for the unit of recovery.
- If DB2 does not have any indoubt unit of recovery, a dummy list is passed. CICS then purges unresolved units of recovery from previous connections, if any exist.

If the units of recovery cannot be resolved because of conditions described in messages DSNC001I, DSNC034I, DSNC035I, or DSNC036I, CICS enables the connection to DB2. For other conditions, it sends message DSNC016I and terminates the connection.

For all resolved units of recovery, DB2 updates databases as necessary and releases the corresponding locks. For threads that access offline databases, the resolution is logged and acted on when the database is started. Unresolved units of work can remain after restart; you can then resolve them.

Related troubleshooting information

“Recovering CICS indoubt units of recovery” on page 608

Resolution of RRS indoubt units of recovery

Sometimes a DB2 unit of recovery (for a thread that uses RRSAF) or an RRS unit of recovery (for a stored procedure) enters the indoubt state.

This is a state where a failure occurs when the participant (DB2 for a thread that uses RRSAF or RRS for a stored procedure) has completed phase 1 of commit processing and is waiting for the decision from the commit coordinator. This failure could be a DB2 abnormal termination, an RRS abnormal termination, or both.

Normally, automatic resolution of indoubt units of recovery occurs when DB2 and RRS re-establish communication with each other. If something prevents this, you can manually resolve an indoubt unit of recovery. This process is not recommended because it might lead to inconsistencies in recoverable resources.

The following errors make manual recovery necessary:

- An RRS cold start where the RRS log is lost.
If DB2 is a participant and has one or more indoubt threads, these indoubt threads must be manually resolved in order to commit or abort the database changes and to release database locks. If DB2 is a coordinator for an RRS unit of recovery, DB2 knows the commit or abort decision but cannot communicate this information to the RRS-compliant resource manager that has an indoubt unit of recovery.
- If DB2 performs a conditional restart and loses information from its log, inconsistent DB2 managed data might exist.
- In a Sysplex, if DB2 is restarted on a z/OS system where RRS is not installed, DB2 might have indoubt threads.
This is a user error because RRS must be started on all processors in a Sysplex on which RRS work is to be performed.

Both DB2 and RRS can display information about indoubt units of recovery. Both also provide techniques for manually resolving these indoubt units of recovery.

In DB2, the `DISPLAY THREAD` command provides information about indoubt DB2 thread. The display output includes RRS unit of recovery IDs for those DB2 threads that have RRS either as a coordinator or as a participant. If DB2 is a participant, you can use the RRS unit of recovery ID that is displayed to determine the outcome of the RRS unit of recovery. If DB2 is the coordinator, you can determine the outcome of the unit of recovery from the `DISPLAY THREAD` output.

In DB2, the `RECOVER INDOUBT` command lets you manually resolve a DB2 indoubt thread. You can use `RECOVER INDOUBT` to commit or roll back a unit of recovery after you determine what the correct decision is.

RRS provides an ISPF interface that provides a similar capability.

Resolving WebSphere Application Server indoubt units of recovery

A *global transaction* is a unit of work that involves operations on multiple resource managers, such as DB2. All of the operations that comprise a global transaction are managed by a transaction manager, such as WebSphere Application Server. When the transaction manager receives transactionally demarcated requests from an application, it translates the requests into a series of transaction control commands, which it directs to the participating resource managers.

Example: An application requests updates to multiple databases. The transaction manager translates these update requests into transaction control commands that are passed to several resource managers. Each resource manager then performs its own set of operations on a database. When the application issues a `COMMIT`, the transaction manager coordinates the commit of the distributed transaction across all participating resource managers by using the two-phase commit protocol. If any resource manager is unable to complete its portion of the global transaction, the transaction manager directs all participating resource managers to roll back the operations that they performed on behalf of the global transaction.

If a communication failure occurs between the first phase (prepare) and the second phase (commit decision) of a commit, an indoubt transaction is created on the resource manager that experienced the failure. When an indoubt transaction is created, a message like this is displayed on the console of the resource manager:

```

00- 17.24.36 STC00051 DSNL405I = THREAD
- G91E1E35.GFA7.00F962CC4611.0001=217
- PLACED IN INDOUBT STATE BECAUSE OF
- COMMUNICATION FAILURE WITH COORDINATOR 9.30.30.53.
- INFORMATION RECORDED IN TRACE RECORD WITH IFCID=209
- AND IFCID SEQUENCE NUMBER=00000001

```

After a failure, WebSphere Application Server is responsible for resolving indoubt transactions and for handling any failure recovery. To perform these functions, the server must be restarted and the recovery process initiated by an operator. You can also manually resolve indoubt transactions with the RECOVER INDOUBT command.

Recommendation: Let WebSphere Application Server resolve the indoubt transactions. Manually recover indoubt transactions only as a last resort to start DB2 and to release locks.

GUPI To manually resolve indoubt transactions:

1. Issue the command `-DISPLAY THREAD(*) T(I) DETAIL` to display indoubt threads from the resource manager console.

This command produces output like this example:

```

00- 17.01.03          =dis thd(*) t(i) detail
- 17.01.04 STC00051 DSNV401I = DISPLAY THREAD REPORT FOLLOWS -
- 17.01.04 STC00051 DSNV406I = INDOUBT THREADS -
- COORDINATOR          STATUS          RESET URID          AUTHID
1 - 9.30.30.53:4007          INDOUBT      0000111F049A          ADMF002
- V437-WORKSTATION=jaijeetsv1, USERID=admf002,
- APPLICATION NAME=db2jccmain
2 - V440-XID=7C7146CE 00000014 00000021 F6EF6F8B
- F36873BE A37AC6BC 256F295D 04BE7EE0
- 8DFEF680 D1A6EFD5 8C0E6343 67679239
- CC15A350 65BFB8EA 670CEBF4 E85938E0
- 06
2 - V467-HAS LUWID G91E1E35.GFA7.00F962CC4611.0001=217
- V466-THREAD HAS BEEN INDOUBT FOR 00:00:17
- DISPLAY INDOUBT REPORT COMPLETE

```

Key	Description
-----	-------------

- | | |
|----------|---|
| 1 | Note that in this example, only one indoubt thread exists. |
| 2 | A transaction is identified by a transaction identifier, called an XID. The first 4 bytes of the XID (in this case, 7C7146CE) identify the transaction manager. Each XID is associated with a logical unit of work ID (LUWID) at the DB2 server. Note the LUWID that is associated with each transaction, for use in the recovery step. |

2. Query the transaction manager to determine whether a commit or abort decision was made for each transaction.
3. Based on the decision that is recorded by the transaction manager, recover each indoubt thread from the resource manager console by either committing or aborting the transaction. Specify the LUWID from the DISPLAY THREAD command in step 1. Use either of the following commands:
 - `-RECOVER INDOUBT ACTION(COMMIT) LUWID(217)`
 - `-RECOVER INDOUBT ACTION(ABORT) LUWID(217)`

Either command produces output like this example:

```

00- 17.30.21          =RECOVER INDOUBT ACTION(COMMIT) LUWID(217)
- 17.30.22 STC00051  DSNV414I  = THREAD
- LUWID=G91E1E35.GFA7.00F962CC4611.0001=217 COMMIT
- SCHEDULED
- 17.30.22 STC00051  DSN9022I  = DSNVRI '-RECOVER INDOUBT' NORMAL COMPLETION

```

4. Display indoubt threads again from the resource manager console by issuing the `-DISPLAY THREAD(*) T(I) DETAIL` command.

This command produces output like this example:

```

00- 17.03.37          =dis thd(*) t(i) detail
- 17.03.37 STC00051  DSNV401I  = DISPLAY THREAD REPORT FOLLOWS -
- 17.03.37 STC00051  DSNV406I  = INDOUBT THREADS -
- COORDINATOR          STATUS      RESET URID          AUTHID
1 - 9.30.30.53:4007      COMMITTED-H  0000111F049A  ADMF002
- V437-WORKSTATION=jaijeetsvl, USERID=admf002,
- APPLICATION NAME=db2jccmain
- V440-XID=7C7146CE 00000014 00000021 F6EF6F8B
- F36873BE A37AC6BC 256F295D 04BE7EE0
- 8DFEF680 D1A6EFD5 8C0E6343 67679239
- CC15A350 65BFB8EA 670CEBF4 E85938E0
- 06
- V467-HAS LUWID G91E1E35.GFA7.00F962CC4611.0001=217
- DISPLAY INDOUBT REPORT COMPLETE

```

Key Description

- 1** Notice that the transaction now appears as a heuristically committed transaction (COMMITTED=H).

5. If the transaction manager does not recover the indoubt transactions in a timely manner, reset the transactions from the resource manager console to purge the indoubt thread information. Specify the IP address and port from the `DISPLAY THREAD` command in step 1 by issuing the `-RESET INDOUBT IPADDR(9.30.30.53:4007)FORCE` command.

This command produces output like this example:

```

00- 17.37.13          =RESET INDOUBT IPADDR(9.30.30.53:4393)FORCE
- 17.37.13 STC00051  DSNL455I  = DB2 HAS NO INFORMATION RELATED TO
- IPADDR 9.30.30.53:4007
- 17.37.13 STC00051  DSNL454I  = QUALIFYING INDOUBT INFORMATION FOR
- IPADDR 9.30.30.53:4007 HAS BEEN PURGED

```



Resolving remote DBMS indoubt units of recovery

When communicating with a remote DBMS, indoubt units of recovery can result from failure with either the participant or coordinator or with the communication link between them even if the systems themselves have not failed.

Normally, if your subsystem fails while communicating with a remote system, you should wait until both systems and their communication link become operational. Your system then automatically recovers its indoubt units of recovery and continues normal operation. When DB2 restarts while any unit of recovery is indoubt, the data that is required for that unit of recovery remains locked until the unit of recovery is resolved.

If automatic recovery is not possible, DB2 alerts you to any indoubt units of recovery that you need to resolve. If releasing locked resources and bypassing the normal recovery process is imperative, you can resolve indoubt situations manually.

Important: In a manual recovery situation, you must determine whether the coordinator decided to commit or abort and ensure that the same decision is made at the participant. In the recovery process, DB2 attempts to automatically resynchronize with its participants. If you decide incorrectly what the recovery action of the coordinator is, data is inconsistent at the coordinator and participant.

If you choose to resolve units of recovery manually, you must:

- Commit changes that were made by logical units of work that were committed by the other system.
- Roll back changes that were made by logical units of work that were rolled back by the other system.

Determining the coordinator's commit or abort decision

You can use any of several methods to ascertain the status of indoubt units at other systems.

- Use a NetView program. Write a program that analyzes NetView alerts for each involved system, and returns the results through the NetView system.
- Use an automated z/OS console to ascertain the status of the indoubt threads at the other involved systems.
- **GUIP** Use the command `DISPLAY THREAD TYPE(INDOUBT) LUWID(luwid)`. If the coordinator DB2 system is started and no DB2 cold start was performed, you can issue a `DISPLAY THREAD TYPE(INDOUBT)` command. If the decision was to commit, the display thread indoubt report includes the LUWID of the indoubt thread. If the decision was to abort, the thread is not displayed. **GUIP**
- Read the recovery log by using `DSN1LOGP`. If the coordinator DB2 cannot be started, `DSN1LOGP` can determine the commit decision. If the coordinator DB2 performed a cold start (or any type of conditional restart), the system log should contain messages `DSNL438I` or `DSNL439I`, which describe the status of the unit of recovery (LUWID).

Recovering indoubt threads

After you determine whether you need to commit or roll back an indoubt thread, recover it with the `RECOVER INDOUBT` command.

GUIP This command does not erase the indoubt status of the thread. It still appears as an indoubt thread until the systems go through the normal resynchronization process. An indoubt thread can be identified by its LUWID, LUNAME, or IP address. You can also use the LUWID token with the command.

Use the `ACTION(ABORT|COMMIT)` option of the `RECOVER INDOUBT` command to commit or roll back a logical unit of work. If your system is the coordinator of one or more other systems that are involved with the logical unit of work, your action propagates to the other system that are associated with the LUW.

Example: Assume that you need to recover two indoubt threads. The first has `LUWID=DB2NET.LUNSITE0.A11A7D7B2057.0002`, and the second has a token of 442. To commit the LUWs, enter:

```
-RECOVER INDOUBT ACTION(COMMIT) LUWID(DB2NET.LUNSITE0.A11A7D7B2057.0002,442)
```

GUIP

Related concepts

“Scenarios for resolving problems with indoubt threads” on page 708

Resetting the status of an indoubt thread

After manual recovery of an indoubt thread, allow the systems to resynchronize automatically; this resets the status of the indoubt thread. However, if heuristic damage or a protocol error occurs, use the RESET INDOUBT command to delete indoubt thread information for a thread whose reset status is set to yes.

DB2 maintains this information until normal automatic recovery. You can purge information about threads where DB2 is either the coordinator or participant. If the thread is an allied thread that is connected to IMS or CICS, the command applies only to coordinator information about downstream participants. Information that is purged does not appear in the next display thread report and is erased from the DB2 logs.

Example:  **GUPI** Assume that you need to purge information about two indoubt threads. The first has an LUWID=DB2NET.LUNSITE0.A11A7D7B2057.0002 and a resync port number of 123, and the second has a token of 442. To purge the information, enter:

```
-RESET INDOUBT LUWID(DB2NET.LUNSITE0.A11A7D7B2057.0002:123,442)
```

 **GUPI**

You can also use a LUNAME or IP address with the RESET INDOUBT command. You can use the keyword IPADDR in place of LUNAME or LUW keywords, when the partner uses TCP/IP instead of SNA. The resync port number of the parameter is required when using the IP address. The DISPLAY THREAD output lists the resync port number. This allows you to specify a location, instead of a particular thread. You can reset all the threads that are associated with that location by using the (*) option.

Chapter 19. Backing up and recovering your data

DB2 provides means for recovering data to its current state or to an earlier state. The units of data that can be recovered are table spaces, indexes, index spaces, partitions, data sets, and the entire system.

For all commands and utility statements, the complete syntax and parameter descriptions can be found in *DB2 Command Reference* and *DB2 Utility Guide and Reference*.

In this information, the term *page set* can be a table space, index space, or any combination of these.

Plans for backup and recovery

Development of backup and recovery procedures at your site is critical in order to avoid costly and time-consuming losses of data.

You should develop procedures to:

- Create a point of consistency
- Recover system and data objects to a point of consistency
- Back up the DB2 catalog and directory and your data
- Recover the DB2 catalog and directory and your data
- Recover from out-of-space conditions
- Recover from a hardware or power failure
- Recover from a z/OS component failure
- Recover from a disaster off-site

If you convert to new function mode and if the installation decides to have archive logs striped or compressed, any recovery actions that involve those archive logs must be done on a DB2 Version 9.1 system that is running in new function mode.

Recommendation: To improve recovery capability in the event of a disk failure, use dual active logging, and place the copies of the active log data sets and the bootstrap data sets on different disk volumes.

The principal tools for DB2 recovery are the QUIESCE, REPORT, COPY, RECOVER, MERGECOPY, BACKUP SYSTEM, and RESTORE SYSTEM utilities. This section also gives an overview of these utilities to help you with your backup and recovery planning.

Related concepts

“How the initial DB2 logging environment is established” on page 481

Related reference

“Implications of moving data sets after a system-level backup” on page 559

Plans for recovery of distributed data

In a distributed data environment, regardless of where a unit of work originates, each unit of work is processed as a whole at the target system. Therefore, a unit of work is recovered as a whole at the target system.

At a DB2 server, the entire unit is either committed or rolled back. It is not processed if it violates referential constraints that are defined within the target system. Whatever changes it makes to data are logged. A set of related table spaces can be quiesced at the same point in the log, and image copies can be made of them simultaneously. If that is done, and if the log is intact, any data can be recovered after a failure and be internally consistent.

However, DB2 provides no special means to coordinate recovery between different subsystems even if an application accesses data in several systems. To guarantee consistency of data between systems, applications should be written so that all related updates are done within one unit of work.

Point-in-time recovery (to the last image copy or to a relative byte address (RBA)) presents other challenges. You cannot control a utility in one subsystem from another subsystem. In practice, you cannot quiesce two sets of table spaces, or make image copies of them, in two different subsystems at exactly the same instant. Neither can you recover them to exactly the same instant, because two different logs are involved, and a RBA does not mean the same thing for both of them.

In planning, the best approach is to consider carefully what the QUIESCE, COPY, and RECOVER utilities do for you, and then plan not to place data that must be closely coordinated on separate subsystems. After that, recovery planning is a matter of agreement among database administrators at separate locations.

DB2 is responsible for recovering DB2 data only; it does not recover non-DB2 data. Non-DB2 systems do not always provide equivalent recovery capabilities.

Plans for extended recovery facility toleration

DB2 can be used in an extended recovery facility (XRF) recovery environment with CICS or IMS.

All DB2-owned data sets (executable code, the DB2 catalog, and user databases) must be on a disk that is shared between the primary and alternate XRF processors. In the event of an XRF recovery, DB2 must be stopped on the primary processor and started on the alternate. For CICS, that can be done automatically, by using the facilities provided by CICS, or manually, by the system operator. For IMS, that is a manual operation and must be done after the coordinating IMS system has completed the processor switch. In that way, any work that includes SQL can be moved to the alternate processor with the remaining non-SQL work. Other DB2 work (interactive or batch SQL and DB2 utilities) must be completed or terminated before DB2 can be switched to the alternate processor. Consider the effect of this potential interruption carefully when planning your XRF recovery scenarios.

Plan carefully to prevent DB2 from being started on the alternate processor until the DB2 system on the active, failing processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. The use of global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of DB2 on the two systems. The bootstrap data set (BSDS) must be included as a protected resource, and the primary and alternate XRF processors must be included in the GRS ring.

Plans for recovery of indexes

You can use the REBUILD INDEX utility or the RECOVER utility to recover DB2 indexes to currency.

The REBUILD INDEX utility reconstructs the indexes by reading the appropriate rows in the table space, extracting the index keys, sorting the keys, and then loading the index keys into the index. The RECOVER utility recovers indexes by restoring an image copy or system-level backup and then applying the log. It can also recover indexes to a prior point in time.

You can use the REBUILD INDEX utility to recover any index, and you do not need to prepare image copies or system-level backups of those indexes.

To use the RECOVER utility to recover indexes, you must include the following actions in your normal database operation:

- **GUPI** Create or alter indexes by using the SQL statement ALTER INDEX with the option COPY YES before you backup and recover them using image copies or system-level backups. **GUPI**
- Create image copies of all indexes that you plan to recover or take system-level backups by using the BACKUP SYSTEM utility.

The COPY utility makes full image copies or concurrent copies of indexes. Incremental copies of indexes are not supported. If full image copies of the index are taken at timely intervals, recovering a large index might be faster than rebuilding the index.

Tip: You can recover indexes and table spaces in a single list when you use the RECOVER utility. If you use a single list of objects in one RECOVER utility control statement, the logs for all of the indexes and table spaces are processed in one pass.

Preparation for recovery: a scenario

The RECOVER utility supports the recovery of table spaces or index spaces.

DB2 can recover a page set by using an image copy or system-level backup, the recovery log, or both. The DB2 recovery log contains a record of all changes that are made to the page set. If DB2 fails, it can recover the page set by restoring the image copy or system-level backup and applying the log changes to it from the point of the image copy or system-level backup.

The DB2 catalog and directory page sets must be copied at least as frequently as the most critical user page sets. Moreover, you are responsible for periodically copying the tables in the communications database (CDB), the application registration table, the object registration table, and the resource limit facility (governor), or for maintaining the information that is necessary to re-create them. Plan your backup strategy accordingly.

The following backup scenario suggests how DB2 utilities might be used when taking object level backups with the COPY utility:

Imagine that you are the database administrator for DBASE1. Table space TSPACE1 in DBASE1 has been available all week. On Friday, a disk write operation for TSPACE1 fails. You need to recover the table space to the last consistent point

before the failure occurred. You can do that because you have regularly followed a cycle of preparations for recovery. The most recent cycle began on Monday morning:

Monday morning

You start the DBASE1 database and make a full image copy of TSPACE1 and all indexes immediately. That gives you a starting point from which to recover. Use the COPY utility with the SHRLEVEL CHANGE option to improve availability.

Tuesday morning

You run the COPY utility again. This time you make an incremental image copy to record only the changes that have been made since the last full image copy that you took on Monday. You also make a full index copy.

TSPACE1 can be accessed and updated while the image copy is being made. For maximum efficiency, however, you schedule the image copies when online use is minimal.

Wednesday morning

You make another incremental image copy, and then create a full image copy by using the MERGECOPY utility to merge the incremental image copy with the full image copy.

Thursday and Friday mornings

You make another incremental image copy and a full index copy each morning.

Friday afternoon

An unsuccessful write operation occurs and you need to recover the table space. You run the RECOVER utility. The utility restores the table space from the full image copy that was made by MERGECOPY on Wednesday and the incremental image copies that were made on Thursday and Friday, and includes all changes that are made to the recovery log since Friday morning.

Later Friday afternoon

The RECOVER utility issues a message announcing that it has successfully recovered TSPACE1 to the current point in time.

This scenario is somewhat simplistic. You might not have taken daily incremental image copies on just the table space that failed. You might not ordinarily recover an entire table space. However, it illustrates this important point: with proper preparation, recovery from a failure is greatly simplified.

Related reference

"RECOVER" (DB2 Utility Guide and Reference)

"COPY" (DB2 Utility Guide and Reference)

Events that occur during recovery

During recovery, several events occur, such as reading the log and running utilities that rely on image copies of the data.

Figure 56 on page 536 shows an overview of the recovery process. To recover a page set, the RECOVER utility typically uses these items:

- A backup that is a full image copy or a system-level backup.

- For table spaces only, when the COPY utility is used, any later incremental image copies; each incremental image copy summarizes all the changes that were made to the table space from the time that the previous image copy was made.
- All log records for the page set that were created since the image copy. If the log has been damaged or discarded, or if data has been changed erroneously and then committed, you can recover to a particular point in time by limiting the range of log records that are to be applied by the RECOVER utility.

In the example shown in Figure 56 on page 536:

- Where the COPY utility is used, the RECOVER utility uses information in the SYSIBM.SYSCOPY catalog table to:
 - Restore the page set with the data in the full image copy (appearing, in Figure 56 on page 536, at X'38000').
 - For table spaces only, apply all the changes that are summarized in any later incremental image copies. (Two copies appear in Figure 56 on page 536: X'80020' and X'C0040'.)
 - Apply all changes to the page set that are registered in the log, beginning at the log RBA of the most recent image copy. (In Figure 56 on page 536, X'C0040', that address is also stored in the SYSIBM.SYSCOPY catalog table.)

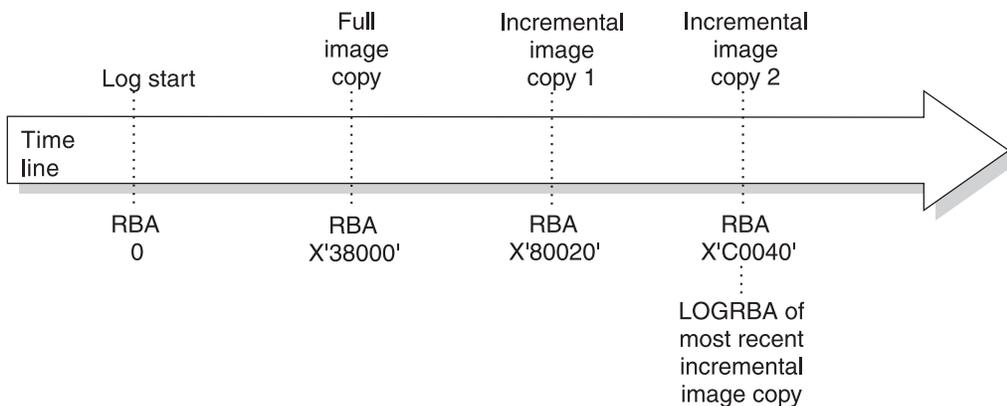


Figure 55. Overview of DB2 recovery from COPY utility. The figure shows one complete cycle of image copies. The SYSIBM.SYSCOPY catalog table can record many complete cycles.

- Where the BACKUP SYSTEM utility is used, the RECOVER utility uses information in the BSDS and in the SYSIBM.SYSCOPY catalog table to:
 - Restore the page set from the most recent backup, in this case, it is a system-level backup (appearing, in Figure 56 on page 536 at X'80000').
 - Apply all changes to the page set that are registered in the log, beginning at the log RBA of the most recent system-level backup.

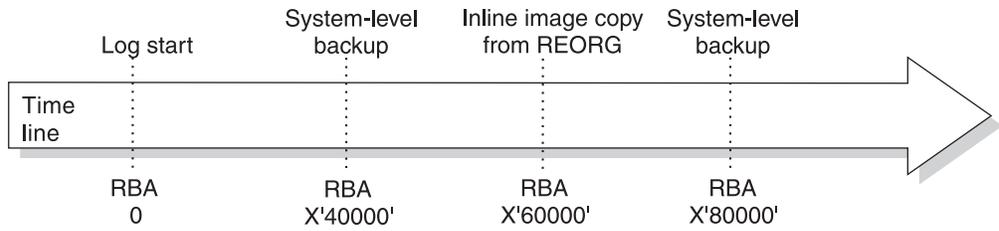


Figure 56. Overview of DB2 recovery from `BACKUP SYSTEM` utility. The figure shows one complete cycle of image copies. The `SYSIBM.SYSCOPY` catalog table can record many complete cycles.

Complete recovery cycles

In planning for recovery, you determine how often to take image copies or system-level backups and how many complete cycles to keep. Those values tell how long you must keep log records and image copies or system-level backups for database recovery.

In deciding how often to take image copies or system-level backups, consider the time needed to recover a table space. The time is determined by all of the following factors:

- The amount of log to traverse
- The time that it takes an operator to mount and remove archive tape volumes
- The time that it takes to read the part of the log that is needed for recovery
- The time that is needed to reprocess changed pages

In general, the more often you make image copies or system-level backups, the less time recovery takes, but the more time is spent making copies. If you use `LOG NO` without the `COPYDDN` keyword when you run the `LOAD` or `REORG` utilities, DB2 places the table space in `COPY pending` status. You must remove the `COPY pending` status of the table space by making an image copy before making further changes to the data. However, if you run `REORG` or `LOAD REPLACE` with the `COPYDDN` keyword, DB2 creates a full image copy of a table space during execution of the utility, so DB2 does not place the table space in `COPY pending` status. Inline copies of indexes during `LOAD` and `REORG` are not supported.

If you use `LOG YES` and log all updates for table spaces, an image copy of the table space is not required for data integrity. However, taking an image copy makes the recovery process more efficient. The process is even more efficient if you use `MERGECOPY` to merge incremental image copies with the latest full image copy. You can schedule the `MERGECOPY` operation at your own convenience, whereas the need for a recovery can arise unexpectedly. The `MERGECOPY` operation does not apply to indexes.

Even if the `BACKUP SYSTEM` is used, it is important to take full image copies (inline copies) during `REORG` and `LOAD` to avoid the copy pending state on the table space.

Recommendation: Copy your indexes after the associated utility has run. Indexes are placed in informational `COPY pending (ICOPY)` status after running `LOAD`

TABLESPACE, REORG TABLESPACE, REBUILD INDEX, or REORG INDEX utilities. Only structural modifications of the index are logged when these utilities are run, so the log does not have enough information to recover the index.

Use the CHANGELIMIT option of the COPY utility to let DB2 determine when an image copy should be performed on a table space and whether a full or incremental copy should be taken. Use the CHANGELIMIT and REPORTONLY options together to let DB2 recommend what types of image copies to make. When you specify both CHANGELIMIT and REPORTONLY, DB2 makes no image copies. The CHANGELIMIT option does not apply to indexes.

In determining how many complete copy and log cycles to keep, you are guarding against damage to a volume that contains an important image copy or a log data set. A retention period of at least two full cycles is recommended. For enhanced security, keep records for three or more copy cycles.

A recovery cycle example when using image copies

This example illustrates log management for a user group.

Table 104 suggests how often a user group with 10 locally defined table spaces (one table per table space) might take image copies, based on frequency of updating. Their least-frequently copied table is EMPALS, which contains employee salary data. If the group chooses to keep two complete image copy cycles on hand, each time EMPALS is copied, records prior to its previous copy or copies, made two months ago, can be deleted. They will always have on hand between two months and four months of log records.

In the example, the user's most critical tables are copied daily. The DB2 catalog and directory are also copied daily.

Table 104. DB2 log management example

Table space name	Content	Update activity	Full image copy period
ORDERINF	Invoice line: part and quantity ordered	Heavy	Daily
SALESINF	Invoice description	Heavy	Daily
SALESQTA	Quota information for each sales person	Moderate	Weekly
SALESDESC	Customer descriptions	Moderate	Weekly
PARTSINV	Parts inventory	Moderate	Weekly
PARTSINF	Parts suppliers	Light	Monthly
PARTS	Parts descriptions	Light	Monthly
SALESCOM	Commission rates	Light	Monthly
EMPLOYEE	Employee descriptive data	Light	Monthly
EMPSALS	Employee salaries	Light	Bimonthly

If you are using the BACKUP SYSTEM utility, you should schedule the frequency of system-level backups based on your most critical data.

If you do a full recovery, you do not need to recover the indexes unless they are damaged. If you recover to a prior point in time, you do need to recover the indexes. See “Plans for recovery of indexes” on page 533 for information about indexes.

How DFSMSHsm affects your recovery environment

The Data Facility Hierarchical Storage Manager (DFSMSHsm) can automatically manage space and data availability among storage devices in your system. If you use it, you need to know that it automatically moves data to and from the DB2 databases.

Restriction: Because DFSMSHsm can migrate data sets to different disk volumes, you cannot use DFSMSHsm in conjunction with the BACKUP SYSTEM utility. The RECOVER utility requires that the data sets reside on the volumes where they had been at the time of the system-level backup.

DFSMSHsm manages your disk space efficiently by moving data sets that have not been used recently to less-expensive storage. It also makes your data available for recovery by automatically copying new or changed data sets to tape or disk. It can delete data sets or move them to another device. Its operations occur daily, at a specified time, and they allow for keeping a data set for a predetermined period before deleting or moving it.

All DFSMSHsm operations can also be performed manually. *z/OS DFSMSHsm Managing Your Own Data* tells how to use the DFSMSHsm commands.

DFSMSHsm:

- Uses cataloged data sets
- Operates on user tables, image copies, and logs
- Supports VSAM data sets

If a volume has a DB2 storage group specified, the volume should be recalled only to like devices of the same VOLSER as defined by CREATE or ALTER STOGROUP.

DB2 can recall user page sets that have been migrated. Whether DFSMSHsm recall occurs automatically is determined by the values of the RECALL DATABASE and RECALL DELAY fields of installation panel DSNTIPO. If the value of the RECALL DATABASE field is NO, automatic recall is not performed and the page set is considered an unavailable resource. It must be recalled explicitly before it can be used by DB2. If the value of the RECALL DATABASE field is YES, DFSMSHsm is invoked to recall the page sets automatically. The program waits for the recall for the amount of time that is specified by the RECALL DELAY parameter. If the recall is not completed within that time, the program receives an error message indicating that the page set is unavailable but that recall was initiated.

The deletion of DFSMSHsm migrated data sets and the DB2 log retention period must be coordinated with use of the MODIFY utility. If not, you could need recovery image copies or logs that have been deleted.

Related tasks

“Discarding archive log records” on page 494

Tips for maximizing data availability during backup and recovery

You need to develop a plan for backup and recovery, and you need to become familiar enough with that plan so that when an outage occurs, you can get back in operation as quickly as possible.

Consider the following factors when you develop and implement your plan:

- “Decide on the level of availability you need”
- “Practice for recovery”
- “Minimize preventable outages” on page 540
- “Determine the required backup frequency” on page 540
- “Minimize the elapsed time of RECOVER jobs” on page 540
- “Minimize the elapsed time for COPY jobs” on page 540
- “Determine the right characteristics for your logs” on page 540
- “Minimize DB2 restart time” on page 541

Related tasks

“Deferring restart processing” on page 510

“Dynamically changing the checkpoint frequency” on page 490

Related reference

“-SET LOG (DB2)” (DB2 Command Reference)

Related information

“Determining the maximum number of open data sets” (DB2 Performance Monitoring and Tuning Guide)

“Setting the size of active log data sets” (DB2 Performance Monitoring and Tuning Guide)

Decide on the level of availability you need

Start by determining the primary types of outages you are likely to experience. Then, for each of those types of outages, decide on the maximum amount of time that you can spend on recovery. Consider the trade-off between cost and availability. Recovery plans for continuous availability are very costly, so you need to think about what percentage of the time your systems really need to be available.

The availability of data is affected by the availability of related objects. For example, if there is an availability issue with one object in a related set, the availability of the others may be impacted as well. The related object set includes base table spaces and indexes, objects related by referential constraints, LOB table space and indexes, and XML table spaces and indexes.

Practice for recovery

You cannot know whether a backup and recovery plan is workable unless you practice it. In addition, the pressure of a recovery situation can cause mistakes. The best way to minimize mistakes is to practice your recovery scenario until you know it well. The best time to practice is outside of regular working hours, when fewer key applications are running.

Minimize preventable outages

One aspect of your backup and recovery plan should be eliminating the need to recover whenever possible. One way to do that is to prevent outages caused by errors in DB2. Be sure to check available maintenance often, and apply fixes for problems that are likely to cause outages.

Determine the required backup frequency

Use your recovery criteria to decide how often to make copies of your databases.

For example, if you use image copies and if the maximum acceptable recovery time after you lose a volume of data is two hours, your volumes typically hold about 4 GB of data, and you can read about 2 GB of data per hour, you should make copies after every 4 GB of data that is written. You can use the COPY option SHRLEVEL CHANGE or DFSMSdss concurrent copy to make copies while transactions and batch jobs are running. You should also make a copy after running jobs that make large numbers of changes. In addition to copying your table spaces, you should also consider copying your indexes.

You can take system-level backups using the BACKUP SYSTEM utility. Because the FlashCopy technology is used, the entire system is backed up very quickly with virtually no data unavailability.

You can make additional backup image copies from a primary image copy by using the COPYTOCOPY utility. This capability is especially useful when the backup image is copied to a remote site that is to be used as a disaster recovery site for the local site. Applications can run concurrently with the COPYTOCOPY utility. Only utilities that write to the SYSCOPY catalog table cannot run concurrently with COPYTOCOPY.

Minimize the elapsed time of RECOVER jobs

When recovering system-level backups from disk, the RECOVER utility restores data sets serially by the main task. When recovering system-level backups from tape, the RECOVER utility creates multiple subtasks to restore the image copies and system-level backups for the objects.

If you are using system-level backups, be sure to have recent system-level backups on disk to reduce the recovery time.

For point-in-time recoveries, recovering to quiesce points and SHRLEVEL REFERENCE copies can be faster than recovering to other points in time. If you are recovering to a non quiesce point, the following factors can have an impact on performance:

- The duration of URs that were active at the point of recovery.
- The number of DB2 members that have active URs to be rolled back.

Minimize the elapsed time for COPY jobs

You can use the COPY utility to make image copies of a list of objects in parallel. Image copies can be made to either disk or tape.

Determine the right characteristics for your logs

- If you have enough disk space, use more and larger active logs. Recovery from active logs is quicker than from archive logs.
- To speed recovery from archive logs, consider archiving to disk.

- If you archive to tape, be sure that you have enough tape drives so that DB2 does not have to wait for an available drive on which to mount an archive tape during recovery.
- Make the buffer pools and the log buffers large enough to be efficient.

Minimize DB2 restart time

Many recovery processes involve restart of DB2. You need to minimize the time that DB2 shutdown and startup take.

You can limit the backout activity during DB2 system restart. You can postpone the backout of long-running units of recovery until after the DB2 system is operational. Use the installation options LIMIT BACKOUT and BACKOUT DURATION to determine what backout work will be delayed during restart processing.

These are some major factors that influence the speed of DB2 shutdown:

- Number of open DB2 data sets

During shutdown, DB2 must close and deallocate all data sets it uses if the fast shutdown feature has been disabled. The default is to use the fast shutdown feature. Contact IBM Software Support for information about enabling and disabling the fast shutdown feature. The maximum number of concurrently open data sets is determined by the DB2 subsystem parameter DSMAX. Closing and deallocation of data sets generally takes .1 to .3 seconds per data set.

Be aware that z/OS global resource serialization (GRS) can increase the time to close DB2 data sets. If your DB2 data sets are not shared among more than one z/OS system, set the GRS RESMIL parameter value to OFF, or place the DB2 data sets in the SYSTEMS exclusion RNL. See *z/OS MVS Planning: Global Resource Serialization* for details.

- Active threads

GUPI DB2 cannot shut down until all threads have terminated. Issue the DB2 command DISPLAY THREAD to determine if any threads are active while DB2 is shutting down. If possible, cancel those threads. **GUPI**

- Processing of SMF data

At DB2 shutdown, z/OS does SMF processing for all DB2 data sets that were opened since DB2 startup. You can reduce the time that this processing takes by setting the z/OS parameter DDCONS(NO).

These major factors influence the speed of DB2 startup:

- DB2 checkpoint interval

The DB2 checkpoint interval indicates the number of log records that DB2 writes between successive checkpoints. This value is controlled by the DB2 subsystem parameter CHKFREQ. The default of 50000 results in the fastest DB2 startup time in most cases.

GUPI You can use the LOGLOAD or CHKTIME option of the SET LOG command to modify the CHKFREQ value dynamically without recycling DB2.

The value that you specify depends on your restart requirements. **GUPI**

- Long-running units of work

DB2 rolls back uncommitted work during startup. The amount of time for this activity is approximately double the time that the unit of work was running before DB2 shut down. For example, if a unit of work runs for two hours before

a DB2 abend, it takes at least four hours to restart DB2. Decide how long you can afford for startup, and avoid units of work that run for more than half that long.

You can use accounting traces to detect long-running units of work. For tasks that modify tables, divide the elapsed time by the number of commit operations to get the average time between commit operations. Add commit operations to applications for which this time is unacceptable.

Recommendation: To detect long-running units of recovery, enable the UR CHECK FREQ option of installation panel DSNTIPL. If long-running units of recovery are unavoidable, consider enabling the LIMIT BACKOUT option on installation panel DSNTIPL.

- Size of active logs

If you archive to tape, you can avoid unnecessary startup delays by making each active log big enough to hold the log records for a typical unit of work. This lessens the probability that DB2 will need to wait for tape mounts during startup.

Where to find recovery information

Information that is needed for recovery is contained in several locations.

SYSIBM.SYSCOPY

SYSIBM.SYSCOPY is a catalog table that contains information about full and incremental image copies. If concurrent updates were allowed when making the copy, the log RBA corresponds to the image copy start time; otherwise, it corresponds to the end time. The RECOVER utility uses the log RBA to look for log information after restoring the image copy. The SYSCOPY catalog table also contains information that is recorded by the COPYTOCOPY utility.

SYSCOPY also contains entries with the same kinds of log RBAs that are recorded by the utilities QUIESCE, REORG, LOAD, REBUILD INDEX, RECOVER TOCOPY, and RECOVER TOLOGPOINT.

When the REORG utility is used, the time at which DB2 writes the log RBA to SYSIBM.SYSCOPY depends on the value of the SHRLEVEL parameter:

- For SHRLEVEL NONE, the log RBA is written at the end of the reload phase.

If a failure occurs before the end of the reload phase, the RBA is not written to SYSCOPY.

If a failure occurs after the reload phase is complete (and thus, after the log RBA is written to SYSCOPY), the RBA is not backed out of SYSCOPY.

- For SHRLEVEL REFERENCE and SHRLEVEL CHANGE, the log RBA is written at the end of the switch phase.

If a failure occurs before the end of the switch phase, the RBA is not written to SYSCOPY.

If a failure occurs after the switch phase is complete (and thus, after the log RBA is written to SYSCOPY), the RBA is not backed out of SYSCOPY.

The log RBA is put in SYSCOPY regardless of whether the LOG option is YES or NO, or whether the UNLOAD PAUSE option is specified.

When DSNDB01.DBD01, DSNDB01.SYSUTILX, and DSNDB06.SYSCOPY are quiesced or copied, a SYSCOPY record is created for each table space and any associated index that has the COPY YES attribute. For recovery reasons, the SYSCOPY records for these three objects are placed in the log.

SYSIBM.SYSLGRNX

SYSIBM.SYSLGRNX is a directory table that contains records of the log RBA ranges that are used during each period of time that any recoverable page set is open for update. Those records speed recovery by limiting the scan of the log for changes that must be applied.

If you discard obsolete image copies, you should consider removing their records from SYSIBM.SYSCOPY and the obsolete log ranges from SYSIBM.SYSLGRNX.

BSDS (bootstrap data set)

The BSDS contains information about system-level backups, and the DB2 archive log data sets. The RECOVER utility uses the recovery base log point RBA or LRSN value associated with the system-level backup to look for the log information after restoring the object from the system-level backup.

In a data-sharing environment, each DB2 member keeps track of the system-level backups taken on that particular member in its BSDS.

DFSMSHsm

The RECOVER utility queries DFSMSHsm for information about whether a system-level backup resides on disk or tape.

Related reference

"DB2 catalog tables" (DB2 SQL Reference)

How to report recovery information

You can use the REPORT utility in planning for recovery.

The REPORT utility provides information necessary for recovering a page set. REPORT displays:

- Recovery information from the SYSIBM.SYSCOPY catalog table
- Log ranges of the table space from the SYSIBM.SYSLGRNX directory
- Archive log data sets from the bootstrap data set
- The names of all members of a table space set

You can also use REPORT to obtain recovery information about the catalog and directory.

Use the output from the DFSMSHsm LIST COPYPOOL command with the ALLVALS option, in conjunction with the DB2 system-level backup information in the PRINT LOG MAP (DSNJU004) utility output, to determine whether the system-level backups of your database copy pool still reside on DASD or if they have been dumped to tape. For a data sharing system, the print log map utility should be run with the MEMBER * option to obtain system-level backup information from all members. For information on how to issue the LIST COPY-POOL command, see the *DFSMSHsm Storage Administration Reference*.

Related reference

"Review of REPORT output" (DB2 Utility Guide and Reference)

How to discard SYSCOPY and SYSLGRNX records

Use the MODIFY utility to delete obsolete records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX.

To keep a table space and its indexes synchronized, the MODIFY utility deletes the SYSCOPY and SYSLGRNX records for the table space and its indexes that are defined with the COPY YES option.

1. Complete the first three steps of the procedure that is presented in “Locating archive log data sets” on page 495. In the third step, note the date of the earliest image copy that you intend to keep.

Important: The earliest image copies and log data sets that you need for recovery to the present date are not necessarily the earliest ones that you want to keep. If you foresee resetting the DB2 subsystem to its status at any earlier date, you also need the image copies and log data sets that allow you to recover to that date.

If the most recent image copy of an object is damaged, the RECOVER utility seeks a backup copy. If no backup copy is available, or if the backup is lost or damaged, RECOVER uses a previous image copy. It continues searching until it finds an undamaged image copy or no more image copies exist. The process has important implications for keeping archive log data sets. At the very least, you need all log records since the most recent image copy; to protect against loss of data from damage to that copy, you need log records as far back as the earliest image copy that you keep.

2. Run the MODIFY RECOVERY utility to clean up obsolete entries in SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX.

Pick one of the following MODIFY strategies, based on keyword options, that is consistent with your overall backup and recovery plan:

DELETE DATE

With the DELETE DATE keyword, you can specify date of the earliest entry you want to keep.

DELETE AGE

With the DELETE AGE keyword, you can specify the age of the earliest entry you want to keep.

RETAIN[®] LAST

With the RETAIN LAST keyword, you can specify the number of image copy entries you want to keep.

RETAIN GDGLIMIT

If GDG data sets are used for your image copies, you can specify RETAIN GDGLIMIT keyword to keep the number of image copies matching your GDG definition.

RETAIN LOGLIMIT

With the RETAIN LOGLIMIT keyword, you can clean up all of the obsolete entries that are older than the oldest archive log in your BOOT STRAP DATA SET (BSDS).

For example, you could enter one of the following commands:

The DELETE DATE option removes records that were written earlier than the given date. You can also specify the DELETE AGE option to remove records that are older than a specified number of days or the DELETE RETAIN option to specify a minimum number of image copies to keep.

```
MODIFY RECOVERY TABLESPACE dbname.tsname
DELETE DATE date
MODIFY RECOVERY TABLESPACE dbname.tsname
RETAIN LAST( n )
```

The RETAIN LAST(*n*) option keeps the *n* recent records and removes the older one.

You can delete SYSCOPY records for a single partition by naming it with the DSNUM keyword. That option does not delete SYSLGRNX records and does not delete SYSCOPY records that are later than the earliest point to which you can recover the entire table space. Thus, you can still recover by partition after that point.

The MODIFY utility discards SYSLGRNX records that meet the deletion criteria when the AGE or DATE options are specified, even if no SYSCOPY records were deleted.

You cannot run the MODIFY utility on a table space that is in RECOVER-pending status.

Even if you take system-level backups, use the MODIFY utility to delete obsolete records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX. You do not need to delete the system-level backup information in the bootstrap data set (BSDS) because only the last 50 system-level backups are kept.

For all options of the MODIFY utility, see Part 2 of the *DB2 Utility Guide and Reference*.

Preparations for disaster recovery

In the case of a total loss of a DB2 computing center, you can recover on another DB2 system at a recovery site. To do this, you must regularly back up the data sets and the log for recovery. As with all data recovery operations, the objectives of disaster recovery are to lose as little data, workload processing (updates), and time as possible.

You can provide shorter restart times after system failures by using the installation options LIMIT BACKOUT and BACKOUT DURATION. These options postpone the backout processing of long-running URs during DB2 restart.

For data sharing environments, you can use the LIGHT(YES) parameter to quickly bring up a DB2 member to recover retained locks. This option is not recommended for refreshing a single subsystem and is intended only for a cross-system restart for a system that has inadequate capacity to sustain the DB2 IRLM pair. Restart light can be used for normal restart and recovery.

For data sharing, you need to consider whether you want the DB2 group to use light mode at the recovery site. A light start might be desirable if you have configured only minimal resources at the remote site. If this is the case, you might run a subset of the members permanently at the remote site. The other members are restarted and then directly shutdown.

To perform a light start at the remote site:

1. Start the members that run permanently with the LIGHT(NO) option. This is the default.
2. Start other members with LIGHT(YES). The members started with LIGHT(YES) use a smaller storage footprint. After their restart processing completes, they automatically shut down. If ARM is in use, ARM does not automatically restart the members with LIGHT(YES) again.

3. Members started with LIGHT(NO) remain active and are available to run new work.

Several levels of preparation for disaster recovery exist:

- Prepare the recovery site to recover to a fixed point in time.
For example, you could copy everything weekly with a DFSMSdss volume dump (logical), manually send it to the recovery site, and then restore the data there.
- For recovery through the last archive, copy and send the following objects to the recovery site as you produce them:
 - Image copies of all catalog, directory, and user page sets
 - Archive logs
 - Integrated catalog facility catalog EXPORT and list
 - BSDS lists

With this approach you can determine how often you want to make copies of essential recovery elements and send them to the recovery site.

After you establish your copy procedure and have it operating, you must prepare to recover your data at the recovery site. See “Performing remote-site disaster recovery” on page 679 for step-by-step instructions on the disaster recovery process.

- Use the log capture exit routine to capture log data in real time and send it to the recovery site. See “Reading log records with the log capture exit routine” and “Log capture routines.”

Related reference

“Active log data set parameters: DSNTIPL” (DB2 Installation Guide)

Related information

“Restart light” (DB2 Data Sharing: Planning and Administration)

System-wide points of consistency

In any disaster recovery scenario, system-wide points of consistency are necessary for maintaining data integrity and preventing a loss of data. A direct relationship exists between the frequency at which you make and send copies to the recovery site and the amount of data that you can potentially lose.

The following figure is an overview of the process of preparing to start DB2 at a recovery site.

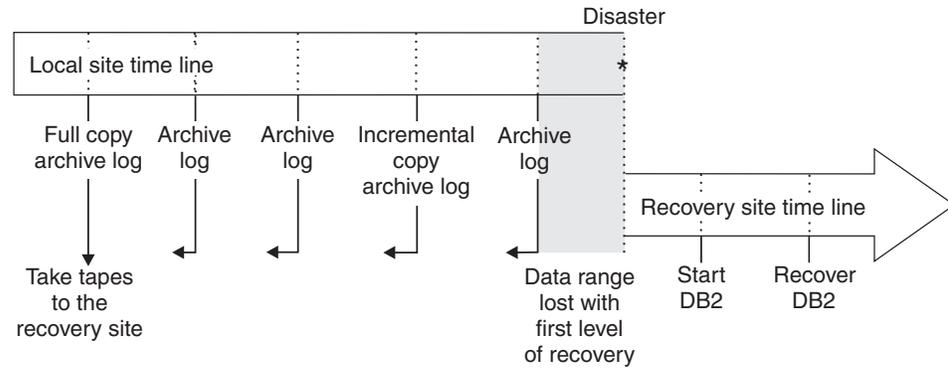


Figure 57. Preparing for disaster recovery. The information that you need to recover is contained in the copies of data (including the DB2 catalog and directory) and the archive log data sets.

Recommendations for more effective recovery from inconsistency

Use of the DB2 RECOVER utility is often the quickest and easiest method of resolving data inconsistency problems. However, these problems can involve data that the RECOVER utility needs to use, such as the recovery log or image copy data sets. If the data that the RECOVER utility needs is damaged or unavailable, you might need to resolve the problem manually.

Actions to take to aid in successful recovery of inconsistent data

Take steps well in advance to prepare for successful recovery of inconsistent data.

- During the installation of, or migration to, Version 9.1, make a full image copy of the DB2 directory and catalog using installation job DSNTIJC.

If you did not do this during installation or migration, use the COPY utility to make a full image copy of the DB2 catalog and directory. If you do not do this and you subsequently have a problem with inconsistent data in the DB2 catalog or directory, you cannot use the RECOVER utility to resolve the problem.

This is recommended even if you take system-level backups.

- Periodically make an image copy of the catalog, directory, and user databases.

This minimizes the time that the RECOVER utility requires to perform recovery. In addition, this increases the probability that the necessary archive log data sets will still be available. You should keep two copies of each level of image copy data set. This reduces the risk involved if one image copy data set is lost or damaged.

This is recommended even if you take system-level backups.

- Use dual logging for your active log, archive log, and bootstrap data sets.

This increases the probability that you can recover from unexpected problems. Dual logging is especially useful in resolving data inconsistency problems.

- Before using RECOVER, rename your data sets.

Restriction: Do not rename your data sets if you take system-level backups.

If the image copy or log data sets are damaged, you can compound your problem by using the RECOVER utility. Therefore, before using RECOVER, rename your data sets by using one of the following methods:

- Rename the data sets that contain the page sets that you want to recover.
- Copy your data sets using DSN1COPY.

- For user-defined data sets, use access method services to define a new data set with the original name.

The RECOVER utility applies log records to the new data set with the old name. Then, if a problem occurs during RECOVER utility processing, you have a copy (under a different name) of the data set that you want to recover.

- Keep back-level copies of your data.

If you make an image copy or system-level backup of a page set that contains inconsistent data, the RECOVER utility cannot resolve the data inconsistency problem. However, you can use RECOVER TOCOPY or TOLOGPOINT to resolve the inconsistency if you have an older image copy or system-level backup of the page set that was taken before the problem occurred. You can also resolve the inconsistency problem by using the RESTOREBEFORE recovery option to avoid using the most recent image copy.

- Maintain consistency between related objects.

A referential structure is a set of tables including indexes and their relationships. It must include at least one table, and for every table in the set, include all of the relationships in which the table participates, as well as all the tables to which it is related. To help maintain referential consistency, keep the number of table spaces in a table space set to a minimum, and avoid tables of different referential structures in the same table space. The TABLESPACESET option of the REPORT utility reports all members of a table space set that are defined by referential constraints.

A referential structure must be kept consistent with respect to point-in-time recovery. Use the QUIESCE utility to establish a point of consistency for a table space set, to which the table space set can later be recovered without introducing referential constraint violations.

A base table space must be kept consistent with its associated LOB or XML table spaces with respect to point-in-time recovery. Use the TABLESPACESET option of the REPORT utility to identify related objects which may include objects related by RI or auxiliary relationships to one or more LOB or XML table spaces and their indexes. Run CHECK INDEX to validate the consistency of indexes with their associated table data. Run CHECK DATA to validate the consistency of base table space data with LOB, XML, and referentially related table spaces. If LOB columns exist, run CHECK LOB on any related LOB table spaces to validate the integrity of each LOB table space within itself.

Related concepts

“How the initial DB2 logging environment is established” on page 481

Related reference

“Installation step 18: Back up the DB2 directory and catalog: DSNTIJIC” (DB2 Installation Guide)

“COPY” (DB2 Utility Guide and Reference)

Actions to avoid in recovery of inconsistent data

To minimize problems when recovering inconsistent data, avoid these actions.

- Do not discard archive logs you might need.

The RECOVER utility might need an archive log to recover from an inconsistent data problem. If you have discarded it, you cannot use the RECOVER utility and must resolve the problem manually.

- Do not make an image copy of a page set that contains inconsistent data.

If you use the COPY utility to make an image copy of a page set that contains inconsistent data, the RECOVER utility cannot recover a problem that involves that page set unless you have an older image copy of that page set that was

taken before the problem occurred. You can run DSN1COPY with the CHECK option to determine whether intra-page data inconsistency problems exist on page sets before making image copies of them. You can also specify the CHECKPAGE parameter on the COPY utility which will cause the image copy to fail if an inconsistent page is detected. If you are taking a copy of a catalog or directory page set, you can run DSN1CHKR, which verifies the integrity of the links, and the CHECK DATA utility, which checks the DB2 catalog (DSNDB06).

- Do not use the TERM UTILITY command on utility jobs that you want to restart.

If an error occurs while a utility is running, the data on which the utility was operating might continue to be written beyond the commit point. If the utility is restarted later, processing resumes at the commit point or at the beginning of the current phase, depending on the restart parameter that was specified. If the utility stops while it has exclusive access to data, other applications cannot access that data. In this case, you might want to issue the TERM UTILITY command to terminate the utility and make the data available to other applications. However, use the TERM UTILITY command only if you cannot restart or do not need to restart the utility job.

When you issue the TERM UTILITY command, two different situations can occur:

- If the utility is active, it terminates at its next commit point.
- If the utility is stopped, it terminates immediately.

If you use the TERM UTILITY command to terminate a utility, the objects on which the utility was operating are left in an indeterminate state. Often, the same utility job cannot be rerun. The specific considerations vary for each utility, depending on the phase in process when you issue the command.

Related tasks

“Discarding archive log records” on page 494

Related reference

“COPY” (DB2 Utility Guide and Reference)

“CHECK DATA” (DB2 Utility Guide and Reference)

“-TERM UTILITY (DB2)” (DB2 Command Reference)

How to recover multiple objects in parallel

To recover multiple objects in parallel, you can either use the RECOVER utility with the PARALLEL keyword or schedule concurrent RECOVER jobs.

You can use the PARALLEL keyword on the RECOVER utility to support the recovery of a list of objects in parallel. For those objects in the list that can be processed independently, multiple subtasks are created to restore the image copies for the objects. The parallel function can be used for either disk or tape.

If an image copy on tape was taken at the tablespace level and not on partition or dataset level, the PARALLEL keyword cannot enable RECOVER utility on different parts (RECOVER TABLESPACE name DSNUM 1 TABLESPACE name DSNUM 2 etc.) to restore the parts in parallel. RECOVER must read the appropriate part of the dataset for every DSNUM specification. This means that for a tablespace level copy on tape, the tape is always read from the beginning. It is recommended for image copies to tape, that a partitioned tablespace should be copied at the partition level and if practical, to different tape stacks. You can use LISTDEF with PARTLEVEL to simplify your work.

When you use one utility statement to recover indexes and table spaces, the logs for all indexes and table spaces are processed in one pass. This approach results in a significant performance advantage, especially when the required archive log data is on tape, or the fast log apply function is enabled, or if both of these conditions occur.

This parallel log processing for fast log apply is not dependent whether you specify RECOVER TABLESPACE name DSNUM 1 TABLESPACE name DSNUM 2 etc. or only RECOVER TABLESPACE name, because the log apply is always done at the partition level. You should also note that if you have copies at the partition level, you cannot specify RECOVER TABLESPACE dbname.tsname, you must specify RECOVER TABLESPACE dbname.tsname DSNUM 1 TABLESPACE dbname.tsname DSNUM 2 etc.. You can simplify this specification by using LISTDEF with PARTLEVEL if all parts must be recovered.

You can schedule concurrent RECOVER jobs that process different partitions. The degree of parallelism in this case is limited by contention for both the image copies and the required log data.

Log data is read by concurrent jobs as follows:

- Active logs and archive logs are read entirely in parallel.
- A data set that is controlled by DFSMSHsm is first recalled. It then resides on disk and is read in parallel.

Related reference

"Syntax and options of the RECOVER control statement" (DB2 Utility Guide and Reference)

"DSNU512I" (DB2 Messages)

Automatic fast log apply during RECOVER

The RECOVER utility automatically uses the fast log apply process during the LOGAPPLY phase if fast log apply has been enabled on the DB2 subsystem.

For detailed information about defining storage for the sort used in fast log apply processing, see the Log Apply Storage field on panel DSNTIPL in *DB2 Installation Guide*.

Recovery of page sets and data sets

You can recover objects in several ways, if the objects have the LOGGED attribute.

- If you made backup copies of table spaces by using the COPY utility, the COPYTOCOPY utility, or the inline copy feature of the LOAD or REORG utility, use the RECOVER utility to recover the table spaces to the current or a previous state.
- If you made backup copies of indexes by using the DB2 COPY utility, use the RECOVER utility to recover the indexes to the current or a previous state. Backing up indexes is optional.
- If you made system-level backups using the BACKUP SYSTEM utility, use the RECOVER utility to recover the objects to the current or a previous state.
- If you have z/OS Version 1.8 and have made backup copies using a method outside of DB2 control, such as with DSN1COPY or the DFSMSdss concurrent copy function, use the same method to restore the objects to a prior point in time. Then, if you want to restore the objects to currency, run the RECOVER utility with the LOGONLY option.

The RECOVER utility performs these actions:

- Restores the most current backup, which can be either a full image copy, concurrent copy, or system-level backup.
- Applies changes recorded in later incremental image copies of table spaces, if applicable, and applies later changes from the archive or active log.

RECOVER can act on:

- A table space, or list of table spaces
- An index, or list of indexes
- A specific partition or data set within a table space
- A specific partition within an index space
- A mixed list of table spaces, indexes, partitions, and data sets
- A single page
- A page range within a table space that DB2 finds in error
- The catalog and directory

Typically, RECOVER restores an object to its current state by applying all image copies or a system-level backup and log records. It can also restore the object to a prior state, which is one of the following points in time:

- A specified point on the log (use the TOLOGPOINT or TORBA keyword)
- A particular image copy (use the TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY keywords)

With z/OS Version 1.8, the RECOVER utility can use system-level backups for object level recovery. The RECOVER utility chooses the most recent backup of either an image copy, concurrent copy, or system-level backup to restore. The most recent backup determination is based on the point of recovery (current or prior point in time) for the table spaces or indexes (with the COPY YES attribute) being recovered.

The RECOVER utility can use image copies for the local site or the recovery site, regardless of where you invoke the utility. The RECOVER utility locates all full and incremental image copies.

The RECOVER utility first attempts to use the primary image copy data set. If an error is encountered (allocation, open, or I/O), RECOVER attempts to use the backup image copy. If DB2 encounters an error in the backup image copy or no backup image copy exists, RECOVER falls back to an earlier full copy and attempts to apply incremental copies and log records. If an earlier full copy is not available, RECOVER attempts to apply log records only.

Not every recovery operation requires RECOVER; see also:

“Recovering error ranges for a work file table space” on page 568

“Recovery of the work file database” on page 552

“Recovery of data to a prior point of consistency” on page 556

Important: Be very careful when using disk dump and restore for recovering a data set. Disk dump and restore can make one data set inconsistent with DB2 subsystem tables in some other data set. Use disk dump and restore only to restore the entire subsystem to a previous point of consistency, and prepare that point as described in the alternative in step 2 on page 565 under “Preparing to recover to a prior point of consistency” on page 563.

Related reference

“Implications of moving data sets after a system-level backup” on page 559

“RECOVER” (DB2 Utility Guide and Reference)

Recovery of the work file database

The work file database (called DSNDB07, except in a data sharing environment) is used for temporary space for certain SQL operations, such as join and ORDER BY.

If DSNDB01.DBD01 is stopped or otherwise inaccessible when DB2 is started, the descriptor for the work file database is not loaded into main storage, and the work file database is not allocated. To recover from this condition after DSNDB01.DBD01 has been made available, stop and then start the work file database again.

You cannot use RECOVER with the work file database.

Page set and data set copies

You can use the COPY utility to copy data from a page set to a z/OS sequential data set on disk or tape. It makes a full or incremental image copy, as you specify, and it can be used to make copies that can be used for local or off-site recovery operations.

Use the COPYTOCOPY utility to make additional image copies from a primary image copy that you made with the COPY utility.

A full image copy is required for indexes.

You can use the CONCURRENT option of the COPY utility to make a copy, with DFSMSdss concurrent copy, that is recorded in the DB2 catalog.

Use the MERGECOPY utility to merge several image copies. MERGECOPY does not apply to indexes.

The CHANGELIMIT option of the COPY utility causes DB2 to make an image copy automatically when a table space has changed past a default limit or a limit that you specify. DB2 determines whether to make a full or incremental image copy based on the values specified for the CHANGELIMIT option.

- If the percent of changed pages is greater than the low CHANGELIMIT value and less than the high CHANGELIMIT value, then DB2 makes an incremental image copy.
- If the percentage of changed pages is greater than or equal to the high CHANGELIMIT value, then DB2 makes a full image copy.
- If the ANY keyword is used with the CHANGELIMIT option is used, then DB2 makes a full image copy.

The CHANGELIMIT option does not apply to indexes.

If you want DB2 to recommend what image copies should be made but not to make the image copies, use the CHANGELIMIT and REPORTONLY options of the COPY utility. If you specify the parameter DSNUM(ALL) with CHANGELIMIT and REPORTONLY, DB2 reports information for each partition of a partitioned table space or each piece of a nonpartitioned table space. For partitioned objects, if you only want the partitions in COPY-pending status or informational COPY-pending status to be copied, then you must specify a list of partitions. You can do this by invoking COPY on a LISTDEF list built with the PARTLEVEL option. An output image copy dataset is created for each partition that is in COPY-pending or informational COPY-pending status.

If you want to copy objects that are in copy pending (COPY) or informational copy pending (ICOPY), you can use the SCOPE PENDING option of the COPY utility. If you specify the parameter DSNUM(ALL) with SCOPE PENDING for partitioned objects, and if one or more of the partitions are in COPY or ICOPY, the copy will be taken of the entire table or index space.

You can add conditional code to your jobs so that an incremental, full image copy, or some other step is performed depending on how much the table space has changed. When you use the COPY utility with the CHANGELIMIT option to display image copy statistics, the COPY utility uses the following return codes to indicate the degree that a table space or list of table spaces has changed:

Code Meaning

- 1 Successful; no CHANGELIMIT value is met. No image copy is recommended or taken.
- 2 Successful; the percentage of changed pages is greater than the low CHANGELIMIT value and less than the high CHANGELIMIT value. An incremental image copy is recommended or taken.
- 3 Successful; the percentage of changed pages is greater than or equal to the high CHANGELIMIT value. A full image copy is recommended or taken.

When you use generation data groups (GDGs) and need to make an incremental image copy, you can take the following steps to prevent an empty image copy output data set from being created if no pages have been changed. You can perform either of the following actions:

- Make a copy of your image copy step, but add the REPORTONLY and CHANGELIMIT options to the new COPY utility statement. The REPORTONLY keyword specifies that you want only image copy information to be displayed. Change the SYSCOPY DD card to DD DUMMY so that no output data set is allocated. Run this step to visually determine the change status of your table space.
- Add step 1 before your existing image copy step, and add a JCL conditional statement to examine the return code and execute the image copy step if the table space changes meet either of the CHANGELIMIT values.

You can use the COPY utility with the CHANGELIMIT option to determine whether any space map pages are broken. You can also use the COPY utility to identify any other problems that might prevent an image copy from being taken, such as the object being in RECOVER-pending status. You need to correct these problems before you run the image copy job.

You can also make a full image copy when you run the LOAD or REORG utility. This technique is better than running the COPY utility after the LOAD or REORG utility because it decreases the time that your table spaces are unavailable. However, only the COPY utility makes image copies of indexes.

Related concepts

“Plans for recovery of indexes” on page 533

Related information

“COPY” (DB2 Utility Guide and Reference)

“MERGECOPY” (DB2 Utility Guide and Reference)

How to make concurrent copies using DFSMS:

The concurrent copy function of Data Facility Storage Management Subsystem (DFSMS) can copy a data set concurrently with access by other processes, without significant impact on application performance.

The two ways to use the concurrent copy function of Data Facility Storage Management Subsystem (DFSMS) are:

- Run the COPY utility with the CONCURRENT option. DB2 records the resulting image copies in SYSIBM.SYSCOPY. To recover with these DFSMS copies, you can run the RECOVER utility to restore those image copies and apply the necessary log records to them to complete recovery.
- Make copies using DFSMS outside of DB2 control. To recover with these copies, you must manually restore the data sets, and then run RECOVER with the LOGONLY option to apply the necessary log records.

Backing up with RVA storage control or Enterprise Storage Server:

The IBM RAMAC[®] Virtual Array (RVA) storage control with the peer-to-peer remote copy (PPRC) function or Enterprise Storage Server[®] provides a faster method of recovering DB2 subsystems at a remote site in the event of a disaster at the local site.

You can use RVAs, PPRC, and the RVA fast copy function, SnapShot, to create entire DB2 subsystem backups to a point in time on a hot stand-by remote site without interruption of any application process. Another option is to use the Enterprise Storage Server FlashCopy function to create point-in-time backups of entire DB2 subsystems.

Using RVA SnapShot or Enterprise Storage Server FlashCopy for a DB2 backup requires a method of suspending all update activity for a DB2 subsystem. This is done to allow a remote copy of the entire subsystem to be made without quiescing the update activity at the primary site. Use the SUSPEND option on the SET LOG command to suspend all logging activity at the primary site, which also prevents any database updates.

After the remote copy is created, use the RESUME option on the SET LOG command to return to normal logging activities. See the *DB2 Command Reference* for more details on using the SET LOG command.

For more information about RVA, see *IBM RAMAC Virtual Array*. For more information about using PPRC, see *RAMAC Virtual Array: Implementing Peer-to-Peer Remote Copy*. For more information about Enterprise Storage Server and the FlashCopy function, see *Enterprise Storage Server Introduction and Planning*.

System-level backups for object-level recoveries

If a system-level backup is chosen as a recovery base for an object, DFSMSShsm is invoked by the RECOVER utility to restore the data sets for the object from the system-level backup of the database copy pool.

Restriction: You can take advantage of certain backup and restore features only if you have z/OS V1.8. These features include:

- The DUMP, DUMPONLY, and FORCE keywords of the BACKUP SYSTEM utility.
- The FROMDUMP and TAPEUNITS keywords of the RESTORE SYSTEM utility.
- The FROMDUMP and TAPEUNITS keywords of the RECOVER utility.

- The ability to restore objects from a system-level backup with the RECOVER utility.

You need to set subsystem parameter `SYSTEM_LEVEL_BACKUPS` to YES so that the RECOVER utility will consider system-level backups.

- If the system-level backup resides on DASD, it is used for the restore of the object.
- If the system-level backup no longer resides on DASD, and has been dumped to tape, then the dumped copy is used for the restore of the object if FROMDUMP was specified.

Message DSNU1520I is issued to indicate that a system-level backup was used as the recovery base.

If DFSMSHsm cannot restore the data sets, message DSNU1522I with RC8 is issued. If `OPTIONS EVENT(ITEMERROR,SKIP)` was specified, then the object is skipped and the recovery proceeds on the rest of the objects, otherwise the RECOVER Utility will terminate.

If you specify YES for the `RESTORE/RECOVER FROM DUMP` install option on installation panel DSNTIP6 or if you specify the FROMDUMP option on the RECOVER utility statement, then only dumps on tape of the database copy pool are used for the restore of the data sets. In addition, if you specify a dump class name on installation panel DSNTIP6 or if you specify the DUMPCLASS option on the RECOVER utility statement, then the data sets will be restored from the system-level backup that was dumped to that particular DFSMSHsm dump class. If you do not specify a dump class name on installation panel DSNTIP6 or on the RECOVER utility statement, then the RESTORE SYSTEM utility issues the DFSMSHsm `LIST COPYPOOL` command and uses the first dump class listed in the output. For more information on DFSMSHsm dump classes, see the *z/OS DFSMSHsm Storage Administration Guide*.

Use the output from `LISTCOPY POOL` and `PRINT LOG MAP` to see the system-level backup information.

Use the output from the `REPORT RECOVERY` utility to determine whether the objects to be recovered have image copies, concurrent copies, or a utility LOG YES event that can be used as a recovery base.

You can take system-level backups using the `BACKUP SYSTEM` utility. However, if any of the following utilities were run since the system-level backup that was chosen as the recovery base, then the use of the system-level backup is prohibited for object level recoveries to a prior point in time:

- REORG TABLESPACE
- REORG INDEX
- REBUILD INDEX
- LOAD REPLACE
- RECOVER from image copy or concurrent copy

In the following illustration, RECOVER TOLOGPOINT receives message DSNU1582I with return code 8, indicating that the recovery has failed.

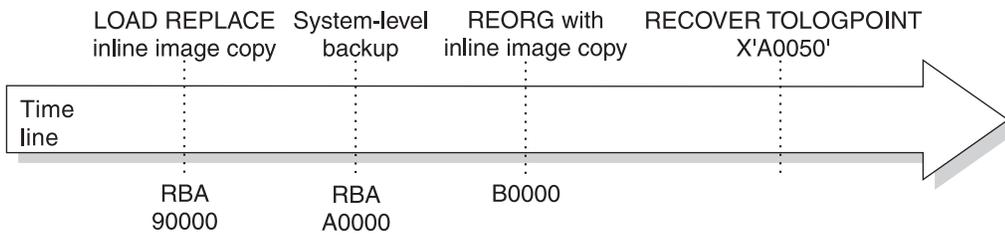


Figure 58. Failed recovery

In this example, use RECOVER with RESTOREBEFORE X'A0000' to use inline image recovery copy taken at X'90000' as a recovery base.

Related concepts

“How to report recovery information” on page 543

Recovery of data to a prior point of consistency

You can restore data to a state it was in at a prior point in time.

To restore data to a prior point in time, use the methods described in the following topics:

- “Options for restoring data to a previous point-in-time” on page 582
- “Restoring data by using DSN1COPY” on page 575
- “Backing up and restoring data with non-DB2 dump and restore” on page 575

The following terms apply to the subject of recovery to a prior point of consistency:

Term Meaning

DBID Database identifier

OBID Data object identifier

PSID Table space identifier

Point-in-time recovery with system-level backups

System-level backups are fast replication backups that are created using the BACKUP SYSTEM utility.

The BACKUP SYSTEM utility invokes z/OS Version 1 Release 5 or later DFSMSHsm services to take volume copies of the data in a sharing DB2 system. All DB2 data sets that are to be copied (and then recovered) must be managed by SMS.

The BACKUP SYSTEM utility requires z/OS Version 1 Release 5 or later data structures called copy pools. Because these data structures are implemented in z/OS, DB2 cannot generate copy pools automatically. Before you invoke the BACKUP SYSTEM utility, copy pools must be allocated in z/OS. For information about how to allocate a copy pool in z/OS, see *z/OS DFSMSdftp Storage Administration Reference*.

The BACKUP SYSTEM utility invokes the DFSMSHsm fast replication function to take volume level backups using FlashCopy.

You can use the BACKUP SYSTEM utility to ease the task of managing data recovery. Choose either DATA ONLY or FULL, depending on your recovery needs. Choose FULL if you want to backup both your DB2 data and your DB2 logs.

Because the BACKUP SYSTEM utility does not quiesce transactions, the system-level backup is a fuzzy copy, which might not contain committed data and might contain uncommitted data. The RESTORE SYSTEM utility uses these backups to restore databases to a given point in time. The DB2 data is made consistent by DB2 restart processing and the RESTORE SYSTEM utility. DB2 restart processing determines which transactions were active at the given recovery point, and writes the compensation log records for any uncommitted work that needs to be backed out. The RESTORE SYSTEM utility restores the database copy pool, and then applies the log records to bring the DB2 data to consistency. During the LOGAPPLY phase of the RESTORE SYSTEM utility, log records are applied to redo the committed work that is missing from the system-level backup, and log records are applied to undo the uncommitted work that might have been contained in the system-level backup.

Data-only system backups

The BACKUP SYSTEM DATA ONLY utility control statement creates system-level backups that contain only databases.

The RESTORE SYSTEM utility uses these backups to restore databases to a given point in time. In this type of recovery, you lose only a few seconds of data, or none, based on the given recovery point. However, recovery time varies and might be extended due to the processing of the DB2 logs during DB2 restart and during the LOGAPPLY phase of the RESTORE SYSTEM utility. The number of logs to process depends on the amount of activity on your DB2 system between the time of the system-level backup and the given recovery point.

Full system backups

The BACKUP SYSTEM FULL utility control statement creates system-level backups that contain both logs and databases. With these backups, you can recover your DB2 system to the point in time of a backup by using normal DB2 restart recovery, or to a given point in time by using the RESTORE SYSTEM utility.

To recover your DB2 system to the point in time of a backup by using normal DB2 restart recovery, stop DB2, and then restore both the database and log copy pools outside of DB2 by using DFSMSHsm FRRECOV COPYPOOL (cpname) GENERATION (gen). After you successfully restart DB2, your DB2 system has been recovered to a point of consistency based on the time of the backup.

The RESTORE SYSTEM utility uses full system backup copies as input, but the utility does not restore the volumes in the log copy pool. If your situation requires that the volumes in the log copy pool be restored, you must restore the log copy pool before restarting DB2. For example, you should restore the log copy pool when you are using a full system-level backup at your remote site for disaster recovery.

When you recover your DB2 system to the point in time of a full system backup, you could lose a few hours of data, because you are restoring your DB2 data and logs to the time of the backup. However, recovery time is

brief, because DB2 restart processing and the RESTORE SYSTEM utility need to process a minimal number of logs.

If you choose not to restore the log copy pool prior to running the RESTORE SYSTEM utility, the recovery is equivalent to the recovery of a system with data-only backups. In this type of recovery, you lose only a few seconds of data, or none, based on the given recovery point. However, recovery time varies and might be extended due to the processing of the DB2 logs during DB2 restart and during the LOGAPPLY phase of the RESTORE SYSTEM utility. The number of logs to process depends on the amount of activity on your DB2 system between the time of the system-level backup and the given recovery point.

You can use the BACKUP SYSTEM utility to manage system-level backups on tape. Choose either DUMP or DUMPPONLY to dump to tape.

Restriction: The DUMP and DUMPPONLY options require z/OS Version 1.8.

Use the DUMP and DUMPPONLY options for:

- Managing the available DASD space.
- Retaining the system-level backups for the long term.
- Providing a means of recovery after a media failure.
- Remote site recovery procedure.

Related concepts

“Restrictions associated with the BACKUP SYSTEM utility and DFSMSHsm” on page 30

Related tasks

“Recovering your DB2 system to a given point in time by using the RESTORE SYSTEM utility” on page 581

“Recovering a DB2 subsystem to a prior point in time” on page 661

Related reference

“Implications of moving data sets after a system-level backup” on page 559

“BACKUP SYSTEM” (DB2 Utility Guide and Reference)

“RESTORE SYSTEM” (DB2 Utility Guide and Reference)

Point-in-time recovery using the RECOVER utility

You can take system-level backups by using the BACKUP SYSTEM utility. Within certain limitations, the RECOVER utility can use these system-level backups for object-level recoveries.

Restriction: Recovery of system-level backups requires z/OS Version 1.8 or later.

If any of the following utilities were run since the system-level backup that was chosen as the recovery base, then the use of the system-level backup by the RECOVER utility is prohibited:

- REORG TABLESPACE
- REORG INDEX
- REBUILD INDEX
- LOAD REPLACE
- RECOVER from image copy or concurrent copy

Related concepts

“Data restore of an entire system” on page 595

Related reference

“Implications of moving data sets after a system-level backup”

Implications of moving data sets after a system-level backup

The movement of data sets after the creation of a system-level backup can affect the ability to do recovery to a prior point in time.

You can still recover the objects if you have an object-level backup such as an image copy or concurrent copy. Take object-level backups to augment system-level backups, or take new system-level backups whenever a data set is moved.

You can force the RECOVER utility to ignore system-level backups and use the latest applicable object-level backup by setting the SYSTEM-LEVEL-BACKUPS parameter on installation panel DSN TIP6 to NO. This subsystem parameter can be updated online.

Activities that can affect the ability to recover an object to a prior point in time from a system-level backup include:

- Migrating to new disk storage or redistributing data sets for performance reasons
The movement of data sets for disk management should be restricted or limited when system-level backups are taken. When movement of data sets does occur, a new system-level backup or object-level backup should immediately be taken.
- Using the DFSMSHsm migrate and recall feature
Do not use the DFSMSHsm migrate and recall feature if you take system-level backups.
- Using REORG TABLESPACE or LOAD REPLACE
To minimize the impact of REORG TABLESPACE and LOAD REPLACE on recoverability, you should always take an inline copy of the underlying table space. You can then use this copy to recover at the object level.
- Using REORG INDEX or REBUILD INDEX
If a REORG INDEX or REBUILD INDEX is preventing the use of the system-level backup during recovery of an index, you can use the REBUILD INDEX utility to rebuild rather than recover the index.
- Using RECOVER from an image copy or concurrent copy

These activities prevent the recovery at the object level to a point in time that would use the system-level backup because the current volume allocation information in the ICF catalog for the data set (or data sets) differs from the volume allocation at the time of the system-level backup.

Consider the following restrictions when planning your backup strategy:

- Data sets can move within a single volume, for example because the volume has been defragmented. In this case, the ability to restore the data is not affected. Recovery of data is not affected if the data set is extended to another volume after it has been copied.
In the case where the data set originally spanned multiple volumes, the data set would need to be reallocated with extents on each of the same volumes before it could be recovered successfully. The amount of space for the data set extents at the time of the backup can differ from the amount of space that is available at recovery time.
- The RECOVER utility cannot use the copy pool backup as the source for a recovery if the data set has been moved to different volumes.

- The movement of one data set in a system-level backup does not prevent the object-level recovery of other objects from the backup.
- The movement of data sets does not prevent the use of the RESTORE SYSTEM utility.

Recovery of table spaces

The way that you recover DB2 table spaces depends on several factors, including the type of table space that needs recovery.

When you recover table spaces to a prior point of consistency, you need to consider:

- How partitioned table spaces, segmented table spaces, LOB table spaces, XML table spaces, and table space sets can restrict recovery.
- If you take system-level backups, how certain utility events can prohibit recovery to a prior point in time.

Related reference

“Implications of moving data sets after a system-level backup” on page 559

Recovery of partitioned table spaces:

You cannot recover a table space to a point in time that is prior to rotating partitions. After you rotate a partition, you cannot recover the contents of that partition.

If you recover to a point in time that is prior to the addition of a partition, DB2 cannot roll back the definition of the partition. In such a recovery, DB2 clears all data from the partition, and the partition remains part of the database.

If you recover a table space partition to a point in time that is before the table space partitions were rebalanced, you must include all partitions that are affected by that rebalance in your recovery list.

Recovery of segmented table spaces:

When data is restored to a prior point in time on a segmented table space, information in the database descriptor (DBD) for the table space might not match the restored table space.

If you use the DB2 RECOVER utility, the DBD is updated dynamically to match the restored table space on the next non-index access of the table. The table space must be in write access mode.

If you use a method outside of DB2 control, such as DSN1COPY to restore a table space to a prior point in time, run the REPAIR utility with the LEVELID option to force DB2 to accept the down-level data. Then run the REORG utility on the table space to correct the DBD.

Recovery of LOB table spaces:

When you recover tables with LOB columns, recover the entire set of objects, including the base table space, the LOB table spaces, and index spaces for the auxiliary indexes.

If you use the RECOVER utility to recover a LOB table space to a prior point of consistency, RECOVER might place the table space in a pending state.

Related concepts

“Options for restoring data to a previous point-in-time” on page 582

Recovery of XML table spaces:

If it is necessary to recover database objects to a prior point in time (RECOVER TOCOPY, TORBA, TOLOGPOINT), all related objects must be recovered to a consistent point in time. This consistency requirement is extended to XML objects.

The RECOVER utility performs consistency checking of the recovery points of these related objects during point-in-time recoveries. Use REPORT TABLESPACESET to obtain a list of these related objects. Use QUIESCE TABLESPACESET if it is necessary to quiesce all objects in the related set.

Recovery of table space sets:

If you restore a page set to a prior state, restore all related tables and indexes to the same point to avoid inconsistencies.

The table spaces that contain referentially related tables, related LOB tables, and related XML tables are called a *table space set*. For example, in the DB2 sample application, a column in the EMPLOYEE table identifies the department to which each employee belongs. The departments are described by records in the DEPARTMENT table, which is in a different table space. If only that table space is restored to a prior state, a row in the unrestored EMPLOYEE table might then identify a department that does not exist in the restored DEPARTMENT table. Run CHECK INDEX to validate the consistency of indexes with their associated table data. Run CHECK DATA to validate the consistency of base table space data with LOB, XML, and referentially related table spaces. If LOB columns exist, run CHECK LOB on any related LOB table spaces to validate the integrity of each LOB table space within itself.

You can use the REPORT TABLESPACESET utility to determine all the page sets that belong to a single table space set and then restore those page sets that are related. However, if page sets are logically related outside of DB2 in application programs, you are responsible for identifying all the page sets on your own.

To determine a valid quiesce point for the table space set, determine a RECOVER TOLOGPOINT value.

Related reference

“RECOVER” (DB2 Utility Guide and Reference)

Recovery of partition-by-growth table spaces:

The RECOVER utility supports both table space and partition level recovery on a partition-by-growth table space.

If an image copy was made on the partition level, the table space can only be recovered at the partition level. A violation of this rule causes message DSNU512I to be issued.

Because the number of partitions is defined on demand, the total number of partitions in an image copy may not be consistent with the number of partitions that reside in the current table space. In such a case, when recovering the table

space back to a point in time with an image copy that has less partitions, the excess partitions in the table space will be empty because the recover process has reset the partitions.

Recovery of indexes

When you recover indexes to a prior point of consistency, some rules apply.

In general, the following rules apply:

- If image copies exist for an indexes, use the RECOVER utility.
- If you take system-level backups, use the RECOVER utility. (This use requires z/OS Version 1.8.)
- If indexes do not have image copies or system-level backups, use REBUILD INDEX to re-create the indexes after the data has been recovered.

More specifically, you must consider how indexes on altered tables and indexes on tables in partitioned table spaces can restrict recovery.

Before attempting recovery, analyze the recovery information.

Related concepts

"How to report recovery information" on page 543

Related reference

"Implications of moving data sets after a system-level backup" on page 559

Recovery of indexes on altered tables:

The use of any of several ALTER statements interferes with the use of the RECOVER utility to restore that index to a point in time that existed before the use of the ALTER statement.

GUPI You cannot use the RECOVER utility to recover an index to a point in time that existed before you issued any of the following ALTER statements on that index. These statements place the index in REBUILD-pending (RBDP) status:

- ALTER INDEX PADDED
- ALTER INDEX NOT PADDED
- ALTER TABLE SET DATA TYPE on an indexed column for numeric data type changes
- ALTER TABLE ADD COLUMN and ALTER INDEX ADD COLUMN that are not issued in the same commit scope
- ALTER INDEX REGENERATE **GUPI**

When you recover a table space to prior point in time and the table space uses indexes that were set to RBDP at any time after the point of recovery, you must use the REBUILD INDEX utility to rebuild these indexes.

Related reference

"REBUILD INDEX" (DB2 Utility Guide and Reference)

"RECOVER" (DB2 Utility Guide and Reference)

"-DISPLAY DATABASE (DB2)" (DB2 Command Reference)

Recovery of indexes on tables in partitioned table spaces:

The partitioning of secondary indexes allows you to copy and recover indexes at the entire index or individual partition level. However, if you use COPY at the partition level, you need to use RECOVER at the partition level also. If you use COPY at the partition level and then try to RECOVER the index, an error occurs. If COPY is performed at the index level, you can use RECOVER at either the index level or the partition level.

You cannot recover an index space to a point in time that is prior to rotating partitions. After you rotate a partition, you cannot recover the contents of that partition to a point in time that is before the rotation.

If you recover to a point in time that is prior to the addition of a partition, DB2 cannot roll back the addition of that partition. In such a recovery, DB2 clears all data from the partition, and it remains part of the database.

Preparing to recover to a prior point of consistency

With some preparation, recovery to the point of consistency is as quick and simple as possible. DB2 begins recovery with the image copy or system-level backup that you made and reads the log only up to the point of consistency. At that point, no indoubt units of recovery hinder restarting.

For a partition-by-growth table space, if the number of physical partitions on the copy is not the same as the number of physical partitions on the current table space, you should delete the extra partition data before or after recovery. This will ensure that the data on the current table space matches the data on the copy.

Related reference

“Implications of moving data sets after a system-level backup” on page 559

Identifying objects to recover:

Identify the related set of objects that must be recovered to the same point in time.

You must recover base table spaces that contain tables with referential constraints, LOB columns, or XML columns to the same point in time.

Use the REPORT TABLESPACESET utility to identify the objects that must be recovered to the same point in time.

Resetting exception status:

Release the data from any exception status.

You can use the DISPLAY DATABASE RESTRICT command to determine whether the data is in an exception status.

Copying the data:

Copy the data, taking appropriate precautions about concurrent activity.

You can copy the data and also establish a point of consistency for a list of objects, in one operation, by using the COPY utility with the option SHRLEVEL REFERENCE. That operation allows only read access to the data while it is copied. The data is consistent at the moment when copying starts and remains consistent until copying ends. The advantage of this method is that the data can be restarted

at a point of consistency by restoring the copy only, with no need to read log records. The disadvantage is that updates cannot be made throughout the entire time that the data is being copied.

You can use the CONCURRENT option of the COPY utility to make a backup, with DFSMSdss concurrent copy, that is recorded in the DB2 catalog. For more information about using this option, see *DB2 Utility Guide and Reference*.

Ideally, you should copy data without allowing updates. However, restricting updates is not always feasible. To allow updates while the data is being copied, you can take either of the following actions:

- Use the COPY utility with the option SHRLEVEL CHANGE.
- Use an off-line program to copy the data, such as DSN1COPY, DFSMSHsm, or disk dump.

If you allow updates while copying, step 3 is recommended. With concurrent updates, the copy can include uncommitted changes. Those might be backed out after copying ends. Thus, the copy does not necessarily contain consistent data, and recovery cannot rely on the copy only. Recovery requires reading the log up to a point of consistency, so you want to establish such a point as soon as possible. Although RECOVER can recover your data to any point in time and ensure data consistency, recovering to a quiesce point can be more efficient. Therefore, taking periodic quiesce points is recommended, if possible.

You can copy all of the data in your system, including the DB2 catalog and directory data by using the BACKUP SYSTEM utility. Since the BACKUP SYSTEM utility allows updates to the data, the system-level backup can include uncommitted data.

Establishing a point of consistency:

After copying the data, immediately establish a point when the data is consistent and no unit of work is changing it.

Use the QUIESCE utility also to establish a single point of consistency (a *quiesce point*) for one or more page sets. Typically, you name all the table spaces in a table space set that you want recovered to the same point in time to avoid referential integrity violations. Alternatively you can use the QUIESCE utility with the TABLESPACESET keyword for referential integrity-related tables. The following statement quiesces two table spaces in database DSN8D91A:

```
QUIESCE TABLESPACE DSN8D91A.DSN8S91E  
        TABLESPACE DSN8D91A.DSN8S91D
```

QUIESCE writes changed pages from the page set to disk. The catalog table SYSIBM.SYSCOPY records the current RBA and the timestamp of the quiesce point. At that point, neither page set contains any uncommitted data. A row with ICTYPE Q is inserted into SYSCOPY for each table space that is quiesced. Page sets DSNDB06.SYSCOPY, DSNDB01.DBD01, and DSNDB01.SYSUTILX are an exception: their information is written to the log. Indexes are quiesced automatically when you specify WRITE(YES) on the QUIESCE statement. A SYSIBM.SYSCOPY row with ICTYPE Q is inserted for indexes that have the COPY YES attribute.

QUIESCE allows concurrency with many other utilities; however, it does not allow concurrent updates until it has quiesced all specified page sets and depending on the amount of activity, that can take considerable time. Try to run QUIESCE when system activity is low.

Also, consider using the MODE(QUIESCE) option of the ARCHIVE LOG command when planning for off-site recovery. It creates a system-wide point of consistency, which can minimize the number of data inconsistencies when the archive log is used with the most current image copy during recovery.

Related tasks

“Archiving the log” on page 489

Preparing to recover an entire DB2 subsystem to a prior point in time using image copies or object-level backups

Under certain circumstances, you might want to reset the entire DB2 subsystem to a point of consistency.

To prepare a point of consistency:

1. Display and resolve any indoubt units of recovery.
2. Use the COPY utility to make image copies of all data, including user data, DB2 catalog and directory table spaces, and optionally indexes. Copy SYSLGRNX and SYSCOPY last.

Install job DSNTIJIC creates image copies of the DB2 catalog and directory table spaces. If you decide to copy your directory and catalog indexes, modify job DSNTIJIC to include those indexes.

Alternate method: Alternatively, you can use an off-line method to copy the data. In that case, stop DB2 first; that is, do the next step before doing this step. If you do not stop DB2 before copying, you might have trouble restarting after restoring the system. If you do a volume restore, verify that the restored data is cataloged in the integrated catalog facility catalog. Use the access method services LISTCAT command to get a listing of the integrated catalog.

3. Stop DB2 with the command STOP DB2 MODE (QUIESCE).

Important: Be sure to use MODE (QUIESCE); otherwise, I/O errors can occur when you fall back before a DB2 restart.

DB2 does not actually stop until all currently executing programs have completed processing.

4. When DB2 has stopped, use access method services EXPORT to copy all BSDS and active log data sets. If you have dual BSDSs or dual active log data sets, export both copies of the BSDS and the logs.
5. Save all the data that has been copied or dumped, and protect it and the archive log data sets from damage.

Related reference

“Installation step 18: Back up the DB2 directory and catalog: DSNTIJIC” (DB2 Installation Guide)

Creating essential disaster recovery elements

You must take steps to create essential disaster recovery elements. For example, you must determine how often to make copies and send them to the recovery site.

1. Make image copies:
 - a. Make copies of your data sets and DB2 catalogs and directories.

Use the COPY utility to make copies for the local subsystem and additional copies for disaster recovery. You can also use the COPYTOCOPY utility to make additional image copies from the primary image copy made by the COPY utility. Install your local subsystem with the LOCALSITE option of

the SITE TYPE field on installation panel DSNTIPO. Use the RECOVERYDDN option when you run COPY to make additional copies for disaster recovery. You can use those copies on any DB2 subsystem that you have installed using the RECOVERYSITE option.

Tip: You can also use these copies on a subsystem that is installed with the LOCALSITE option if you run RECOVER with the RECOVERYSITE option. Alternatively, you can use copies that are prepared for the local site on a recovery site if you run RECOVER with the option LOCALSITE.

Important: Do not produce copies by invoking COPY twice.

- b. Optional: Catalog the image copies if you want to track them.
- c. Create a QMF report or use SPUFI to issue a SELECT statement to list the contents of SYSCOPY.
- d. Send the image copies, and report to the recovery site.
- e. Record this activity at the recovery site when the image copies and the report are received.

All table spaces should have valid image copies. Indexes can have valid image copies or they can be rebuilt from the table spaces.

2. Make copies of the archive logs for the recovery site:

- a. Use the ARCHIVE LOG command to archive all current DB2 active log data sets. For more ARCHIVE LOG command information see “Archiving the log” on page 489.

Recommendation: When using dual logging, keep both copies of the archive log at the local site in case the first copy becomes unreadable. If the first copy is unreadable, DB2 requests the second copy. If the second copy is not available, the read fails.

However, if you take precautions when using dual logging, such as making another copy of the first archive log, you can send the second copy to the recovery site. If recovery is necessary at the recovery site, specify YES for the READ COPY2 ARCHIVE field on installation panel DSNTIPO. Using this option causes DB2 to request the second archive log first.

- b. Optional: Catalog the archive logs if you want to track them.
You will probably need some way to track the volume serial numbers and data set names. One way of doing this is to catalog the archive logs to create a record of the necessary information. You can also create your own tracking method and do it manually.
- c. Use the print log map utility to create a BSDS report.
- d. Send the archive copy, the BSDS report, and any additional information about the archive log to the recovery site.
- e. Record this activity at the recovery site when the archive copy and the report are received.

3. Choose consistent system time:

Important: After you establish a consistent system time, do not alter the system clock. Any manual change in the system time (forward or backward) can affect how DB2 writes and processes image copies and log records.

- a. Choose a consistent system time for all DB2 subsystems.

DB2 utilities, including the RECOVER utility, require system clocks that are consistent across all members of a DB2 data-sharing group. To prevent

- inconsistencies during data recovery, ensure that all system times are consistent with the system time at the failing site.
 - b. Ensure that the system clock remains consistent.
4. Back up integrated catalog facility catalogs:
 - a. Back up all DB2-related integrated catalog facility catalogs with the VSAM EXPORT command on a daily basis.
 - b. Synchronize the backups with the cataloging of image copies and archives.
 - c. Use the VSAM LISTCAT command to create a list of the DB2 entries.
 - d. Send the EXPORT backup and list to the recovery site.
 - e. Record this activity at the recovery site when the EXPORT backup and list are received.
 5. Back up DB2 libraries:
 - a. Back up DB2 libraries to tape when they are changed. Include the SMP/E, load, distribution, and target libraries, as well as the most recent user applications and DBRMs.
 - b. Back up the DSNTIJUZ job that builds the ZPARM and DECP modules.
 - c. Back up the data set allocations for the BSDS, logs, directory, and catalogs.
 - d. Document your backups.
 - e. Send backups and corresponding documentation to the recovery site.
 - f. Record activity at the recovery site when the library backup and documentation are received.

For disaster recovery to be successful, all copies and reports must be updated and sent to the recovery site regularly. Data is up to date through the last archive that is sent. For disaster recovery start up procedures, see “Performing remote-site disaster recovery” on page 679.

Related information

“Multiple image copies” (DB2 Utility Guide and Reference)

Resolving problems with a user-defined work file data set

You can resolve problems on a volume of a user-defined data set for the work file database

1. Issue the following DB2 command:


```
-STOP DATABASE (DSNDB07)
```
2. Use the DELETE and DEFINE functions of access method services to redefine a user work file on a different volume, and reconnect it to DB2.
3. Issue the following DB2 command:


```
-START DATABASE (DSNDB07)
```

Resolving problems with DB2-managed work file data sets

You can resolve problems on a volume in a DB2 storage group for the work file database, such as a system I/O problem.

GUIP

1. Enter the following SQL statement to remove the problem volume from the DB2 storage group:


```
ALTER STOGROUP stogroup-name
  REMOVE VOLUMES (xxxxxx);
```
2. Issue the following DB2 command:

- STOP DATABASE (DSNDB07)
- 3. Enter the following SQL statement to drop the table space with the problem:
DROP TABLESPACE DSNDB07.*tsname*;
- 4. Re-create the table space. You can use the same storage group, because the problem volume has been removed, or you can use an alternate.
CREATE TABLESPACE *tsname*
IN DSNDB07
USING STOGROUP *stogroup-name*;
- 5. Issue the following command:
-START DATABASE (DSNDB07)



Recovering error ranges for a work file table space

Page error ranges operate for work file table spaces in the same way as for other DB2 table spaces, except for the process of recovering them. Error ranges in a work file table space cannot be reset by RECOVER ERROR RANGE. Instead, perform the following procedure

To recover error ranges for a work file table space:

1. Stop the work file table space.
2. Correct the disk error, using the ICKDSF service utility or access method services to delete and redefine the data set.
3. Start the work file table space. When the work file table space is started, DB2 automatically resets the error range.

Recovery of error ranges for a work file table space

DB2 always resets any error ranges when the work file table space is initialized, regardless of whether the disk error has really been corrected.

Work file table spaces are initialized when:

- The work file table space is stopped and then started.
- The work file database is stopped and then started, and the work file table space was not previously stopped.
- DB2 is started and the work file table space was not previously stopped.

If the error range is reset while the disk error still exists, and if DB2 has an I/O error when using the work file table space again, DB2 sets the error range again.

Recovering after a conditional restart of DB2

After a DB2 conditional restart in which a log record range is specified, such as with a cold start, a portion of the DB2 recovery log is no longer available. If the unavailable portion includes information that is needed for internal DB2 processing, an attempt to use the RECOVER utility to restore directory table spaces DSNDBD01 or SYSUTILX, or catalog table space SYSCOPY, fails with ABEND04E RC00E40119.

Instead of using the RECOVER utility, use this procedure to recover those table spaces and their indexes:

1. Run DSN1COPY to restore the table spaces from an image copy.
2. Run the RECOVER utility with the LOGONLY option to apply updates from the log records to the recovered table spaces.

3. Rebuild the indexes.
4. Make a full image copy of the table spaces, and optionally the indexes, to make the table spaces and indexes recoverable.

Recovery of the catalog and directory

Catalog and directory objects must be recovered in a particular order. Because the recovery of some catalog and directory objects depends on information that is derived from other catalog and directory objects, you must recover some of these objects in separate RECOVER utility control statements.

You can, however, use the same RECOVER control statement to recover a catalog or directory table space along with its corresponding IBM-defined indexes. After these logically dependent objects are restored to an undamaged state, you can recover the remaining catalog and directory objects in a single RECOVER utility control statement. These restrictions apply regardless of the type of recovery that you perform on the catalog.

You can use the REPORT utility to report on recovery information about the catalog and directory.

To avoid restart processing of any page sets before attempts are made to recover any of the members of the list of catalog and directory objects, you must set subsystem parameters DEFER and ALL. You can do this by setting the values DEFER in field 1 and ALL in field 2 of installation panel DSNTIPS.

Important: Recovering the DB2 catalog and directory to a prior point in time is strongly discouraged.

Related concepts

“How to report recovery information” on page 543

Related tasks

“Deferring restart processing” on page 510

Related reference

“RECOVER” (DB2 Utility Guide and Reference)

Regenerating missing identity column values

You can regenerate missing identity column values.

To regenerate missing identity column values:

1. **GUI** Choose a starting value for the identity column with the following ALTER TABLE statement:

```
ALTER TABLE table-name
ALTER COLUMN identity-column-name
RESTART WITH starting-identity-value
```

GUI

2. Run the REORG utility to regenerate lost sequence values.

If you do not choose a starting value in step 1, the REORG utility generates a sequence of identity column values that starts with the next value that DB2 would have assigned before the recovery.

Recovery of tables that contain identity columns

When you recover a table that contains an identity column, consider the point in time to which you recover. Your recovery procedure can depend on whether that identity column existed, or was not yet defined at the point in time to which you recover.

The following considerations apply for each of these two cases.

Identity column already defined

If you recover to a point in time at which the identity column existed, you might create a gap in the sequence of identity column values. When you insert a row after this recovery, DB2 produces an identity value for the row as if all previously added rows still exist.

For example, assume that at time T1 an identity column that is incremented by 1 contains the identity column values 1 through 100. At T2, the same identity column contains the values 1 through 1000. Now, assume that the table space is recovered back to time T1. When you insert a row after the recovery, DB2 generates an identity value of 1001. This value leaves a gap from 101 to 1000 in the values of the identity column.

GUPI To prevent a gap in identity column values, use the following ALTER TABLE statement to modify the attributes of the identity column before you insert rows after the recovery:

```
ALTER TABLE table-name
  ALTER COLUMN identity-column-name
  RESTART WITH next-identity-value
```

GUPI

Tip: To determine the last value in an identity column, issue the MAX column function for ascending sequences of identity column values, or the MIN column function for descending sequences of identity column values. This method works only if the identity column does not use CYCLE.

Identity column not yet defined

If you recover to a point in time at which the identity column was not yet defined, that identity column remains part of the table. The resulting identity column no longer contains values.

A table space that contains an identity column is set to REORG-pending (REORP) status if you recover the table space to a point in time that is before the identity column was defined. To access the recovered table, you need to remove this status.

Related concepts

“Data consistency for point-in-time recoveries” on page 584

Recovering a table space and all of its indexes

To recover a table space and all of its indexes (or a table space set and all related indexes), use a single RECOVER utility statement that specifies TOLOGPOINT.

For the log point, you can identify a quiesce point or a common SHRLEVEL REFERENCE copy point. This action avoids placing indexes in the

CHECK-pending or RECOVER-pending status. If the log point is not a common quiesce point or SHRLEVEL REFERENCE copy point for all objects, use the following procedure, which ensures that the table spaces and indexes are synchronized and eliminates the need to run the CHECK INDEX utility.

With recovery to a point in time with consistency, which is the default recovery type, you do not need to identify a quiesce point or a common SHRLEVEL REFERENCE copy point. This recovery might be faster because inconsistencies do not have to be resolved.

To recover to a log point:

1. RECOVER table spaces to the log point.
2. Use concurrent REBUILD INDEX jobs to rebuild the indexes for each table space.

Recovery implications for objects that are not logged

You can use the RECOVER utility on objects that have the NOT LOGGED attribute. The NOT LOGGED attribute does not mean that the contents of an object are non-recoverable, however, the modifications to the object that is not logged are non-recoverable.

Objects that are not logged include the table space, the index, and the index space. Recovery can be to any recoverable point. A *recoverable point* is established when:

- A table space is altered from logged to not logged.
- When an image copy is taken against an object that is not logged.
- An ALTER TABLE statement is issued with the ADD PARTITION clause, against a table in a table space that has the NOT LOGGED attribute.
- DB2 adds a new partition in response to insertion of data into a partition-by-growth table space.

The TORBA or TOLOGPOINT keywords can also be used for a point-in-time recovery on an object that is not logged, but the RBA or LRSN must correspond to a recoverable point or message DSNU1504I is issued.

If a base table space is altered so that it is not logged, and its associated LOB table spaces already have the NOT LOGGED attribute, then the point where the table space is altered is not a recoverable point.

If DB2 restart recovery determines that a table space that is not logged might have been in the process of being updated at the time of the failure, then the table space or partition is placed in the LPL and is marked RECOVER-pending.

Related reference

"CREATE TABLESPACE" (DB2 SQL Reference)

"ALTER TABLESPACE" (DB2 SQL Reference)

Clearing the informational COPY-pending status (ICOPY):

If you update a table space while it has the NOT LOGGED logging attribute, the table space is marked Informational Copy Pending (ICOPY).

You can use the DISPLAY DATABASE ADVISORY command to display the ICOPY status for table spaces. For example:

```

DSNT361I - * DISPLAY DATABASE SUMMARY
          * ADVISORY
DSNT360I - *****
DSNT362I - DATABASE = DBIQUA01 STATUS = RW
          DBD LENGTH = 8066
DSNT397I -
NAME      TYPE PART STATUS      HYERRLO PHYERRHI CATALOG PIECE
-----
TPIQUQ01 TS      001 RW,AUXW
TPIQUQ01 TS      002 RW,AUXW
TPIQUQ01 TS      003 RW,AUXW
TPIQUQ01 TS      004 RW,ICOPY

```

To clear the ICOPY status, you must take a full image copy of the table space.

The LOG option of LOAD or REORG:

The logging attribute that you specify when you run the LOAD and REORG utilities has different results depending on the logging attribute of the table space that you are working with.

The following tables show how the logging attribute of the utility and the logging attribute of the table space interact:

Table 105. Attribute interaction for LOB table spaces

LOAD or REORG keyword	Table space logging attribute	What is logged	Table space status after completion
LOG YES	LOGGED	Control records and LOB data redo information. LOB data undo information is never logged for LOB table spaces.	No pending status
LOG YES	NOT LOGGED	Control information.	No pending status
LOG NO	LOGGED	Nothing.	COPY-pending
LOG NO	NOT LOGGED	Nothing.	No pending status

Table 106. Attribute interaction for non-LOB table spaces

LOAD or REORG keyword	Table space logging attribute	What is logged	Table space status after completion
LOG YES	LOGGED	Control records and data.	No pending status
LOG YES	NOT LOGGED	LOG YES is changed to LOG NO.	See Table 107 on page 573
LOG NO	LOGGED	Nothing.	COPY-pending
LOG NO	NOT LOGGED	Nothing.	See Table 107 on page 573

The following table shows the possible table space statuses for non-LOB tables spaces that are not logged:

Table 107. Status of non-LOB table spaces that are not logged, after LOAD or REORG with LOG NO keyword

Inline copy	Records discarded	Table space status
Yes	No	No pending status
Yes	Yes	ICOPY-pending
No	not applicable	ICOPY-pending

Clearing the RECOVER-pending status:

If it becomes necessary for DB2 to undo work that has not been logged (as when a rollback occurs), the table space has lost its data integrity and is marked RECOVER-pending. To prevent access to corrupt data, the pages are placed in the LPL.

Tip: Application programmers should commit frequently and try to avoid duplicate key or referential integrity violations when modifying a table in a NOT LOGGED table space.

If DB2 restart recovery determines that a not logged table space may have been in the process of being updated at the time of the failure, then the table space or partition is placed in the LPL and is marked RECOVER-pending. You have several options for removing a table space from the LPL and resetting the RECOVER-pending status:

- Dropping and re-creating the table space and repopulating the table.
- “Using a REFRESH TABLE statement”
- “Using the RECOVER utility”
- “Using the LOAD REPLACE utility” on page 574
- “Using a DELETE statement without a WHERE clause” on page 574
- “Using a TRUNCATE TABLE statement” on page 574

When a job fails and a rollback begins, the undo records are not available for table spaces that are not logged during the back-out. Therefore, the rows that are in the table space after recovery might not be the correct rows. You can issue the appropriate SQL statements to re-create the intended rows.

Using a REFRESH TABLE statement:

GUPI Use the REFRESH TABLE statement to repopulate a materialized query table, but only if the materialized query table is alone in its table space. If the table is not alone in its table space, a utility must be used to reset the table space and remove it from RECOVER-pending status. **GUPI**

Using the RECOVER utility:

Use the RECOVER utility to recover to a recoverable point.

You can run the RECOVER utility against a table space with the NOT LOGGED logging attribute. To do so, the current logging attribute of the table space must match the logging attribute of the recovery base (that is, the logging attribute of the table space when the image copy was taken). If no changes have been made to

the table space since the last point of recovery, the utility completes successfully. If changes have been made, the utility completes with message DSNU1504I.

You can use RECOVER with the TOCOPY, TOLASTFULLCOPY, or TOLASTCOPY keyword to identify which image copy to use. You can also use TORBA or TOLOGPOINT, but the RBA or LRSN must correspond to a recoverable point.

You cannot use RECOVER with the LOGONLY keyword.

Using the LOAD REPLACE utility:

Use the LOAD REPLACE utility or the LOAD REPLACE PART utility in the following situations:

- With an input data set to empty the table space and repopulate the table.
- Without an input data set to empty the table space to prepare for one or more INSERT statements to repopulate the table.

Using a DELETE statement without a WHERE clause:

GUPI Use the DELETE statement without a WHERE clause to empty the table, when the table space is segmented or universal, the table is alone in its table space and the table does not have:

- A VALIDPROC
- Referential constraints
- Delete Triggers
- A SECURITY LABEL column (or it does have such a column, but multilevel security with row level granularity is not in effect) **GUPI**

Using a TRUNCATE TABLE statement:

GUPI Use the TRUNCATE TABLE statement to empty the table, when the table space is segmented and the table is alone in its table space and the table does not have:

- A VALIDPROC
- Referential constraints
- A SECURITY LABEL column (or it does have such a column, but multilevel security with row level granularity is not in effect) **GUPI**

Removing various pending states from LOB and XML table spaces

You can remove various pending states from an LOB or XML table space by using a collection of utilities in a specific order.

1. Use the REORG TABLESPACE utility to remove the REORP status.
2. If the table space status is auxiliary CHECK-pending status:
 - a. Use CHECK LOB for all associated LOB table spaces.
 - b. Use CHECK INDEX for all LOB indexes, as well as the document ID, node ID, and XML indexes.
3. Use the CHECK DATA utility to remove the CHECK-pending status.

Restoring data by using DSN1COPY

You can use DSN1COPY to restore data that was previously backed up by DSN1COPY or by COPY. If you use DSN1COPY to restore data or move data, the data definitions for the target object must be exactly the same as when the copy was created. You cannot use DSN1COPY to restore data that was backed up with the DFSMSdss concurrent copy facility.

Be careful when creating backups with DSN1COPY. You must ensure that the data is consistent, or you might produce faulty backup copies. One advantage of using COPY to create backups is that it does not allow you to copy data that is in CHECK-pending or RECOVER-pending status. You can use COPY to prepare an up-to-date image copy of a table space, either by making a full image copy or by making an incremental image copy and merging that incremental copy with the most recent full image copy.

Keep access method services LISTCAT listings of table space data sets that correspond to each level of retained backup data.

Related reference

"DSN1COPY" (DB2 Utility Guide and Reference)

Backing up and restoring data with non-DB2 dump and restore

You can use certain non-DB2 facilities to dump and restore data sets and volumes. However, certain limitations exist.

Even though DB2 data sets are defined as VSAM data sets, DB2 data cannot be read or written by VSAM record processing because it has a different internal format. The data can be accessed by VSAM control interval (CI) processing. If you manage your own data sets, you can define them as VSAM linear data sets (LDSs), and access them through services that support data sets of that type.

Access method services for CI and LDS processing are available in z/OS. IMPORT and EXPORT use CI processing; PRINT and REPRO do not, but they do support LDSs.

DFSMS Data Set Services (DFSMSdss) is available on z/OS and provides dump and restore services that can be used on DB2 data sets. Those services use VSAM CI processing.

Recovering accidentally dropped objects

If a table or table space is inadvertently dropped, you can recover the object.

Recommendation: To prepare for recovery of an object, run regular catalog reports that include a list of all OBIDs in the subsystem. In addition create catalog reports that list dependencies on the table or (such as referential constraints, indexes, and so on). After a table is dropped, this information disappears from the catalog.

If an OBID has been reused by DB2, you must run DSN1COPY to translate the OBIDs of the objects in the data set. However, this event is unlikely; DB2 reuses OBIDs only when no image copies exist that contain data from that table.

Important: When you recover a dropped object, you essentially recover a table space to a point in time. If you want to use log records to perform forward recovery on the table space, you need the IBM DB2 Log Analysis Tool for z/OS.

How to avoid accidentally dropping objects

To avoid the problem of accidentally dropping tables, you can create a table with the clause `WITH RESTRICT ON DROP`.

GUPI When a table has been created with the clause `WITH RESTRICT ON DROP`, then nobody can drop the table, or the table space or database that contains the table, until the restriction on the table is removed. The `ALTER TABLE` statement includes a clause to remove the restriction, as well as one to impose it.

GUPI

Recovering an accidentally dropped table

Tables in a partitioned table space cannot be dropped without dropping the table space.

To perform this procedure, you need a full image copy or a `DSN1COPY` file that contains the data from the dropped table.

For segmented or universal table spaces, the image copy or `DSN1COPY` file must contain the table when it was active (that is, created). Because of the way space is reused for segmented table spaces, this procedure cannot be used if the table was not active when the image copy or `DSN1COPY` was made. For nonsegmented table spaces, the image copy or `DSN1COPY` file can contain the table when it was active or not active.

To recover a dropped table:

1. If you know the `DBID`, the `PSID`, the original `OBID` of the dropped table, and the `OBIDs` of all other tables in the table space, go to step 2.

If you do not know all of the preceding items, use the following steps to find them. For later use with `DSN1COPY`, record the `DBID`, the `PSID`, and the `OBIDs` of all the tables in the table space, not just the dropped table.

- a. For the data set that contains the dropped table, run `DSN1PRNT` with the `FORMAT` and `NODATA` options. Record the `HPGOBID` field in the header page and the `PGSOBD` field from the data records in the data pages.

For the auxiliary table of a LOB table space, record the `HPGROID` field in the header page instead of the `PGSOBD` field in the data pages.

- Field `HPGOBID` is 4 bytes long and contains the `DBID` in the first 2 bytes and the `PSID` in the last 2 bytes.
 - Field `HPGROID` (for LOB table spaces) contains the `OBID` of the table. A LOB table space can contain only one table.
 - Field `PGSOBD` (for non-LOB table spaces) is 2 bytes long and contains the `OBID` of the table. If your table space contains more than one table, check for all `OBIDs`. By searching for all different `PGSOBD` fields. You need to specify all `OBIDs` from the data set as input for the `DSN1COPY` utility.
- b. Convert the hex values in the identifier fields to decimal so that they can be used as input for the `DSN1COPY` utility.
2. **GUPI** Use the SQL `CREATE` statement to re-create the table and any indexes on the table. **GUPI**
 3. To allow `DSN1COPY` to access the DB2 data set, stop the table space using the following command:
`-STOP DATABASE(database-name) SPACENAM(tablespace-name)`

Stopping the table space is necessary to ensure that all changes are written out and that no data updates occur during this procedure.

4. **PSPI** Find the OBID for the table that you created in step 2 on page 576 by querying the SYSIBM.SYSTABLES catalog table.

The following statement returns the object ID (OBID) for the table:

```
SELECT NAME, OBID FROM SYSIBM.SYSTABLES
WHERE NAME='table_name'
AND CREATOR='creator_name';
```

This value is returned in decimal format, which is the format that you need for DSN1COPY. **PSPI**

5. Run DSN1COPY with the OBIDXLAT and RESET options to perform the OBID translation and to copy data from the dropped table into the original data set. You must specify a previous full image copy data set, inline copy data set, or DSN1COPY file as the input data set SYSUT1 in the control statement. Specify each of the input records in the following order in the SYSXLAT file to perform OBID translations:
 - a. The DBID that you recorded in step 1 on page 576 as both the translation source and the translation target
 - b. The PSID that you recorded in step 1 on page 576 as both the translation source and the translation target
 - c. The original OBID that you recorded in step 1 on page 576 for the dropped table as the translation source and the OBID that you recorded in step 4 as the translation target
 - d. OBIDs of all other tables in the table space that you recorded in step 2 on page 576 as both the translation sources and translation targetsBe sure that you have named the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.
6. Use DSN1COPY with the OBIDXLAT and RESET options to apply any incremental image copies. You must apply these incremental copies in sequence, and specify the same SYSXLAT records that step 5 specifies.

Important: After you complete this step, you have essentially recovered the table space to the point in time of the last image copy. If you want to use log records to perform forward recovery on the table space, you must use the IBM DB2 Log Analysis Tool for z/OS at this point in the recovery procedure.

7. Start the table space for normal use by using the following command:
`-START DATABASE(database-name) SPACENAM(tablespace-name)`
8. Rebuild all indexes on the table space.
9. Execute SELECT statements on the previously dropped table to verify that you can access the table. Include all LOB columns in these queries.
10. Make a full image copy of the table space.
11. Re-create the objects that are dependent on the recovered table.

When a table is dropped, objects that are dependent on that table (synonyms, views, indexes, referential constraints, and so on) are dropped. (Aliases are not dropped.) Privileges that are granted for that table are also dropped. Catalog reports or a copy of the catalog taken prior to the DROP TABLE can make this task easier.

Related concepts

“Recovery of data to a prior point of consistency” on page 556

“Page set and data set copies” on page 552

“Implications of dropping a table” on page 120

Related tasks

“Recovering an accidentally dropped table space”

Related reference

“DSN1COPY” (DB2 Utility Guide and Reference)

Recovering an accidentally dropped table space

If you accidentally drop a table space, including LOB table spaces, you can recover that table space.

You might accidentally drop a table space, for example, when all tables in an implicitly created table space are dropped, or if someone unintentionally executes a DROP TABLESPACE statement for a particular table space.

When a table space is dropped, DB2 loses all information about the image copies of that table space. Although the image copy data set is not lost, locating it might require examination of image copy job listings or manually recorded information about the image copies.

The recovery procedures for user-managed data sets and for DB2-managed data sets are slightly different.

Recovering accidentally dropped DB2-managed data sets:

If a consistent full image copy or DSN1COPY file is available, you can use DSN1COPY to recover a dropped table space that is part of the catalog.

To recover a dropped table space, complete the following procedure:

1. Find the original DBID for the database, the PSID for the table space, and the OBIDs of all tables that are contained in the dropped table space. For information about how to do this, see step 1 on page 576 of “Recovering an accidentally dropped table” on page 576.
2. Re-create the table space and all tables. This re-creation can be difficult when any of the following conditions is true:
 - A table definition is not available.
 - A table is no longer required.

If you cannot re-create a table, you must use a dummy table to take its place. A *dummy table* is a table with an arbitrary structure of columns that you delete after you recover the dropped table space.

Attention: When you use a dummy table, you lose all data from the dropped table that you do not re-create.

3. Re-create auxiliary tables and indexes if a LOB table space has been dropped.
4. To allow DSN1COPY to access the DB2 data set, stop the table space with the following command:

```
-STOP DATABASE(database-name) SPACENAM(tablespace-name)
```

5.  Find the new PSID and OBIDs by querying the SYSIBM.SYSTABLESPACE and SYSIBM.SYSTABLES catalog tables.

The following statement returns the object ID for a table space; this is the PSID.

```
SELECT DBID, PSID FROM SYSIBM.SYSTABLESPACE  
WHERE NAME='tablespace_name' and DBNAME='database_name'  
AND CREATOR='creator_name';
```

The following statement returns the object ID for a table:

```
SELECT NAME, OBID FROM SYSIBM.SYSTABLES
WHERE NAME='table_name'
AND CREATOR='creator_name';
```

These values are returned in decimal format, which is the format that you need for DSN1COPY. (Find the OBID of the dummy table that you created in step 2 on page 578 if you could not re-create a table.) 

6. Run DSN1COPY with the OBIDLAT and RESET options to translate the OBID and to copy data from a previous full image copy data set, inline copy data set, or DSN1COPY file. Use one of these copies as the input data set SYSUT1 in the control statement. Specify each of the input records in the following order in the SYSXLAT file to perform OBID translations:
 - a. The DBID that you recorded in step 1 on page 578 as both the translation source and the translation target
 - b. The PSID that you recorded in step 1 on page 578 as the translation source and the PSID that you recorded in step 5 on page 578 as the translation target
 - c. The OBIDs that you recorded in step 1 on page 578 as the translation sources and the OBIDs that you recorded in step 5 on page 578 as the translation targets

Be sure that you name the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.

7. Use DSN1COPY with the OBIDLAT and RESET options to apply any incremental image copies to the recovered table space. You must apply these incremental copies in sequence, and specify the same SYSXLAT records that step 7 on page 580 specifies.

Important: After you complete this step, you have essentially recovered the table space to the point in time of the last image copy. If you want to use log records to perform forward recovery on the table space, you must use the IBM DB2 UDB Log Analysis Tool for z/OS at this point in the recovery procedure. For more information about point-in-time recovery, see “Recovery of data to a prior point of consistency” on page 556.

8. Start the table space for normal use by using the following command:
`-START DATABASE(database-name) SPACENAM(tablespace-name)`
9. Drop all dummy tables. The row structure does not match the table definition. This mismatch makes the data in these tables unusable.
10. Reorganize the table space to remove all rows from dropped tables.
11. Rebuild all indexes on the table space.
12. Execute SELECT statements on each table in the recovered table space to verify the recovery. Include all LOB columns in these queries.
13. Make a full image copy of the table space.
See “Page set and data set copies” on page 552 for more information about the COPY utility.
14. Re-create the objects that are dependent on the table.
See step 11 on page 577 of “Recovering an accidentally dropped table” on page 576 for more information.

Related reference

“DSN1COPY” (DB2 Utility Guide and Reference)

Recovering accidentally dropped user-managed data sets:

You can recover dropped table spaces that are not part of the catalog by copying the data sets that contain data from the dropped table space to redefined data sets.

To accomplish this copy, you use the OBID-translate function of DSN1COPY.

1. Find the DBID for the database, the PSID for the dropped table space, and the OBIDs for the tables that are contained in the dropped table space. For information about how to do this, see step 1 on page 576 of “Recovering an accidentally dropped table” on page 576.
2. Rename the data set that contains the dropped table space by using the IDCAMS ALTER command. Rename both the CLUSTER and DATA portion of the data set with a name that begins with the integrated catalog facility catalog name or alias.
3. Redefine the original DB2 VSAM data sets.

Use the access method services LISTCAT command to obtain a list of data set attributes. The data set attributes on the redefined data sets must be the same as they were on the original data sets.

4. Use SQL CREATE statements to re-create the table space, tables, and any indexes on the tables.
5. To allow DSN1COPY to access the DB2 data sets, stop the table space by using the following command:

```
-STOP DATABASE(database-name) SPACENAM(tablespace-name)
```

This step is necessary to prevent updates to the table space during this procedure in the event that the table space has been left open.

6.  Find the target identifiers of the objects that you created in step 4 (which consist of a PSID for the table space and the OBIDs for the tables within that table space) by querying the SYSIBM.SYSTABLESPACE and SYSIBM.SYSTABLES catalog tables.

The following statement returns the object ID for a table space; this is the PSID.

```
SELECT DBID, PSID FROM SYSIBM.SYSTABLESPACE  
WHERE NAME='tablespace_name' and DBNAME='database_name'  
AND CREATOR='creator_name';
```

The following statement returns the object ID for a table:

```
SELECT NAME, OBID FROM SYSIBM.SYSTABLES  
WHERE NAME='table_name'  
AND CREATOR='creator_name';
```

These values are returned in decimal format, which is the format that you need for DSN1COPY. 

7. Run DSN1COPY with the OBIDLAT and RESET options to perform the OBID translation and to copy the data from the renamed VSAM data set that contains the dropped table space to the newly defined VSAM data set. Specify the VSAM data set that contains data from the dropped table space as the input data set SYSUT1 in the control statement. Specify each of the input records in the following order in the SYSXLAT file to perform OBID translations:
 - a. The DBID that you recorded in step 1 as both the translation source and the translation target
 - b. The PSID that you recorded in step 1 as the translation source and the PSID that you recorded in step 6 as the translation target
 - c. The original OBIDs that you recorded in step 1 as the translation sources and the OBIDs that you recorded in step 6 as the translation targets

Be sure that you have named the VSAM data sets correctly by checking messages DSN1998I and DSN1997I after DSN1COPY completes.

8. Use DSN1COPY with the OBIDLAT and RESET options to apply any incremental image copies to the recovered table space. You must apply these incremental copies in sequence, and specify the same SYSXLAT records that step 7 on page 580 specifies.

Important: After you complete this step, you have essentially recovered the table space to the point in time of the last image copy. If you want to use log records to perform forward recovery on the table space, you must use the IBM DB2 UDB Log Analysis Tool for z/OS.

For more information about point-in-time recovery, see “Recovery of data to a prior point of consistency” on page 556.

9. Start the table space for normal use by using the following command:

```
-START DATABASE(database-name) SPACENAM(tablespace-name)
```

10. Rebuild all indexes on the table space.
11. Execute SELECT statements on each table in the recovered table space to verify the recovery. Include all LOB columns in these queries.
12. Make a full image copy of the table space.
See “Page set and data set copies” on page 552 for more information about the COPY utility.
13. Re-create the objects that are dependent on the table.
See step 11 on page 577 of “Recovering an accidentally dropped table” on page 576 for more information.

Related reference

“DSN1COPY” (DB2 Utility Guide and Reference)

Recovering your DB2 system to a given point in time by using the RESTORE SYSTEM utility

Use the RESTORE SYSTEM utility to recover your DB2 system to a given point in time. Recovering to a given point in time minimizes the amount of data that you lose when you recover.

The RESTORE SYSTEM utility uses system-level backups that contain only DB2 objects to restore your DB2 system to a given point in time.

Prerequisite: Before you can use the RESTORE SYSTEM utility, you must use the BACKUP SYSTEM utility to create system-level backups. Choose either DATA ONLY or FULL, depending on your recovery needs. Choose FULL if you want to backup both your DB2 data and your DB2 logs.

To recover data to a given point in time:

1. Issue the STOP DB2 command to stop your DB2 system. If your system is a data sharing group, stop all members of the group.
2. If the backup is a full system backup, you might need to restore the log copy pool outside of DB2 by using DFSMSHsm FRRECOV COPYPOOL (cpname) GENERATION (gen). For data-only system backups, skip this step.
3. Create a conditional restart record, where the SYSPITR option specifies the given point-in-time that you want to recover to. Run DSNJU003 (the change log inventory utility) with the CRESTART SYSPITR and SYSPITRT options, and specify the log truncation point that corresponds to the point-in-time to which

you want to recover the system. For data sharing systems, run DSNJU003 on all active members of the data-sharing group, and specify the same LRSN truncation point for each member. If the point-in-time that you specify for recovery is prior to the oldest system backup, you must manually restore the volume backup from tape.

4. For data sharing systems, delete all CF structures that the data sharing group owns.
5. Restore any logs on tape to disk.
6. Issue the START DB2 command to restart your DB2 system. For data sharing systems, start all active members.
7. Run the RESTORE SYSTEM utility. If you manually restored the backup, use the LOGONLY option of RESTORE SYSTEM to apply the current logs.
8. Stop and restart DB2 again to remove ACCESS(MAINT) status.

After the RESTORE SYSTEM utility completes successfully, your DB2 system has been recovered to the given point in time with consistency.

Related concepts

“Backup and recovery involving clone tables” on page 595

Related tasks

“Recovering a DB2 subsystem to a prior point in time” on page 661

Options for restoring data to a previous point-in-time

TOCOPY, TOLOGPOINT, TOLASTCOPY, TORBA, and TOLASTFULLCOPY are options of the RECOVER utility. All of these options terminate recovery at a specified point. Because they recover data to a prior time, and not to the present, they are referred to as point-in-time recoveries.

The TOCOPY, TOLASTCOPY, and TOLASTFULLCOPY options identify an image copy as the point-in-time to which to recover. With these options, the RECOVER utility restores objects to the value of a specified image copy and does not apply subsequent changes from the log. If the image copy that is specified in one of these options cannot be applied, the RECOVER utility uses an earlier full image copy and applies the changes in the log up to the point-in-time at which the specified image copy was taken.

If the image copy data set is cataloged when the image copy is made, the entry for that copy in SYSIBM.SYSCOPY does not record the volume serial numbers of the data set. You can identify that copy by its name by using TOCOPY *data set name*. If the image copy data set was not cataloged when it was created, you can identify the copy by its volume serial identifier by using TOVOLUME *volser*.

With TOLOGPOINT, the RECOVER utility restores the object from the most recent of either a full image copy or system-level backup taken prior to the recovery point. If a full image copy is restored, the most recent set of incremental copies that occur before the specified log point are restored. The logged changes are applied up to, and including, the record that contains the log point. If no full image copy or system-level backup exists before the chosen log point, recovery is attempted entirely from the log. The log is applied from the log point at which the page set was created or the last LOAD or REORG TABLESPACE utility was run to a log point that you specify. You can apply the log only if you have not used the MODIFY RECOVERY utility to delete the SYSIBM.SYSLGRNX records for the log range your recovery requires.

Uncommitted transactions running at the recover point-in-time are automatically detected and the changes on the recovered objects are rolled back leaving them in a transactionally consistent state.

You can use the TOLOGPOINT option in both data sharing and non-data-sharing environments. In a non-data-sharing environment, TOLOGPOINT and TORBA are interchangeable keywords that identify an RBA on the log at which recovery is to stop. TORBA can be used in a data sharing environment only if the TORBA value is before the point at which data sharing was enabled.

Recommendation: Use the TOLOGPOINT keyword instead of the TORBA keyword. Although DB2 still supports the TORBA option, the TOLOGPOINT option supports both data sharing and non-data-sharing environments and is used for both of these environments throughout this information.

Plans for point-in-time recovery:

In some circumstances, you cannot recover to the current point in time. If you plan for this possibility, you can establish a consistent point in time from which to recover if these circumstances occur.

TOCOPY is a viable alternative in many situations in which recovery to the current point in time is not possible or is not desirable. When making copies of a single object, use SHRLEVEL REFERENCE to establish consistent points for TOCOPY recovery. Copies that are made with SHRLEVEL CHANGE do not copy data at a single instant, because changes can occur as the copy is made. A subsequent RECOVER TOCOPY operation can produce inconsistent data.

When copying a list of objects, use SHRLEVEL REFERENCE. If a subsequent recovery to a point in time is necessary, you can use a single RECOVER utility statement to list all of the objects, along with TOLOGPOINT, to identify the common RBA or LRSN value.

An inline copy that is made during LOAD REPLACE can produce unpredictable results if that copy is used later in a RECOVER TOCOPY operation. DB2 makes the copy during the RELOAD phase of the LOAD operation. Therefore, the copy does not contain corrections for unique index violations and referential constraint violations because those corrections occur during the INDEXVAL and ENFORCE phases.

You can use the QUIESCE utility to establish an RBA or LRSN to recover to. The RBA or LRSN can be used in point-in-time recovery.

If you are working with partitioned table spaces, image copies or system-level backups that were taken prior to resetting the REORG-pending status of any partition of a partitioned table space cannot be used for recovery to a current point in time. Avoid performing a point-in-time recovery for a partitioned table space to a point in time that is after the REORG-pending status was set, but before a rebalancing REORG was performed.

If you use the REORG TABLESPACE utility with the SHRLEVEL REFERENCE or SHRLEVEL CHANGE option on only some partitions of a table space, you must recover that table space at the partition level. When you take an image copy of such a table space, the COPY utility issues the informational message DSNU429I.

You can take system-level backups using the BACKUP SYSTEM utility.

Recommendation: Restrict the use of the TOCOPY, TOLOGPOINT, TOLASTCOPY, and TOLASTFULLCOPY options of the RECOVER utility to personnel with a thorough knowledge of the DB2 recovery environment.

Related concepts

“Point-in-time recovery using the RECOVER utility” on page 558

Related reference

“RECOVER” (DB2 Utility Guide and Reference)

Data consistency for point-in-time recoveries:

The RECOVER utility can automatically detect uncommitted transactions running at the recover point in time and roll back the changes on the recovered objects. After recovery, the objects will be left in their transactionally consistent state.

RECOVER TOLOGPOINT and RECOVER TORBA have the recover with consistency as their default behavior. However, RECOVER TOCOPY, TOLASTCOPY, and TOLASTFULLCOPY using SHRLEVEL CHANGE image copy do not ensure data consistency.

RECOVER TOLOGPOINT and RECOVER TOCOPY can be used on a single:

- Partition of a partitioned table space
- Partition of a partitioning index space
- Data set of a simple table space

Tip: If you take SHRLEVEL CHANGE image copies and need to recover to a prior point in time, then you can use the RBA or LRSN (the START_RBA syscopy column value) associated with image copy as the TOLOGPOINT value.

All page sets must be restored to the same level; otherwise the data is inconsistent.

Point-in-time recovery can cause table spaces to be placed in CHECK-pending status if they have table check constraints or referential constraints that are defined on them. When recovering tables that are involved in a referential constraint, you should recover all the table spaces that are involved in a constraint. This is the *table space set*.

To avoid setting CHECK-pending status, you must perform both of the following tasks:

- Recover the table space set to a quiesce point.
If you do not recover each table space of the table space set to the same quiesce point, and if any of the table spaces are part of a referential integrity structure:
 - All dependent table spaces that are recovered are placed in CHECK-pending status with the scope of the whole table space.
 - All table spaces that are dependent on the table spaces that are recovered are placed in CHECK-pending status with the scope of the specific dependent tables.

- Establish a quiesce point or take an image copy after you add check constraints or referential constraints to a table.

If you recover each table space of a table space set to the same quiesce point, but referential constraints were defined after the quiesce point, the CHECK-pending status is set for the table space that contains the table with the referential constraint.

The RECOVER utility sets various states on table spaces. The following point-in-time recoveries set various states on table spaces:

- When the RECOVER utility finds an invalid column during the LOGAPPLY phase on a LOB table space, it sets the table space to auxiliary-warning (AUXW) status.
- When you recover a LOB or XML table space to a point in time that is not a quiesce point or to an image copy that is produced with SHRLEVEL CHANGE, the LOB or XML table space is placed in CHECK-pending (CHKP) status.
- When you recover the LOB or XML table space, but not the base table space, to any previous point in time, the base table space is placed in auxiliary CHECK-pending (ACHKP) status, and the index space that contains an index on the auxiliary table is placed in REBUILD-pending (RBDP) status.
- When you recover only the base table space to a point in time, the base table space is placed in CHECK-pending (CHKP) status.
- When you recover only the index space that contains an index on the auxiliary table to a point in time, the index space is placed in CHECK-pending (CHKP) status.
- When you recover partitioned table spaces with the RECOVER utility to a point in time that is prior to a partition rebalance, all partitions that were rebalanced are placed in REORG-pending (REORP) status.
- When you recover a table space to point in time prior to when an identity column was defined with the RECOVER utility, the table space is placed in REORG-pending (REORP) status.
- If you do not recover all objects that are members of a referential set to a prior point in time with the RECOVER utility, or if you recover a referential set to a point in time that is not a point of consistency with the RECOVER utility, all dependent table spaces are placed in CHECK-pending (CHKP) status.
- When you recover a table space to a point in time prior to the REORG or LOAD REPLACE that first materializes the default value for the row change timestamp column, the table space that contains the table with the row change timestamp column is placed in REORG-pending (REORP) status.

Important: The RECOVER utility does not back out CREATE or ALTER statements. After a recovery to a previous point in time, all previous alterations to identity column attributes remain unchanged. Because these alterations are not backed out, a recovery to a point in time might put identity column tables out of sync with the SYSIBM.SYSSEQUENCES catalog table. You might need to modify identity column attributes after a recovery to resynchronize identity columns with the catalog table.

A table space that is in reordered row format cannot be recovered to a prior point in time in which the table space was in basic row format, or vice versa. If you need to recover table spaces with different row formats, you need to recover the entire table space.

Related troubleshooting information

“Recovering from referential constraint violation” on page 672

Related information

“Recovering a table space that contains LOB or XML data” (DB2 Utility Guide and Reference)

The RECOVER TOLOGPOINT option in a data sharing system:

You can use the RECOVER utility with the TOLOGPOINT option to recover a data sharing DB2 subsystem.

The following figure shows a data sharing system with three DB2 members (A, B, and C). Table space TS1 and TS2 are being recovered to time TR using the RECOVER TOLOGPOINT option, they are listed in the same RECOVER job. UR1 was inflight at TR time running on member A, its start time was T1, at time T2 it updated TS1, and at time T3 it updated TS2, T2 is earlier than T3. UR2 was aborting at TR time running on member C, its start time was T4 and at time T5 it updated TS2. There was no active UR on member B at time TR.

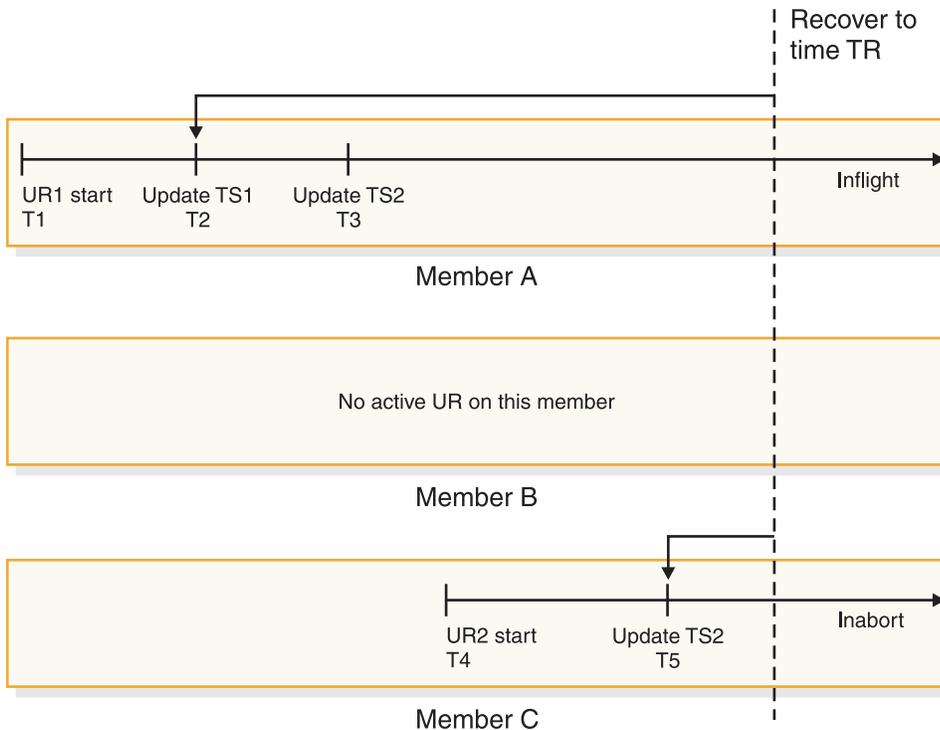


Figure 59. Using the RECOVER TOLOGPOINT option in a data sharing system

RECOVER utility job output messages

The RECOVER utility takes the following actions and provides the following messages during recovery in a data sharing system.

LOGCSR phase

After the RECOVER LOGAPPLY phase, the RECOVER utility enters the log analysis phase, known as the LOGCSR phase. The following messages are issued during this phase:

DSNU1550I

Indicates the start of log analysis on member A.

DSNU1551I

Indicates the end of log analysis on member A.

DSNU1550I

Indicates the start of log analysis on member C.

DSNU1551I

Indicates the end of log analysis on member C.

|
|
| **DSNU1552I**

Indicates the end of LOGCSR phase of RECOVER utility.

|
| **DSNU1553I**

Issued after the end of the LOGCSR phase. The following information is shown in the message:

- UR1 on member A modified TS1 at T2 time.
- UR1 on member A modified TS2 at T3 time.
- UR2 on member C modified TS2 at T5 time.

|
| **LOGUNDO phase**

The RECOVER utility enters the LOGUNDO phase. The following messages are issued during this phase:

| **DSNU1554I**

Indicates the start of backout for member A.

- Backout is performed on TS2 and TS1 and log records are read from time TR to T2.
- There maybe one or more DSNU1555I messages showing the backout progress for member A.

| **DSNU1556I**

Indicates the end of backout on member A.

| **DSNU1554I**

Indicates the start of backout on member C.

- Backout is performed on TS2 and log records are read from time TR to T5.
- There may be one or more DSNU1555I messages showing the backout progress for member C.

| **DSNU1556I**

Indicates the end of backout for member C.

| **DSNU1557I**

Indicates the end of the LOGUNDO phase of the RECOVER utility.

A special type of compensation log record will be written during the LOGUNDO phase of log apply, known as the *pseudo* compensation log record in the following context. For each undo log record applied during the LOGUNDO phase, there will be one pseudo compensation log record written for it. This is a REDO-only log record and not units of recovery (UR) related. It will only be used for future log apply in the case that you first recover objects to a point in time with consistency and then recover the same objects again to currency using the same image copy used by previous recover job.

This type of log record will be skipped by DB2 restart and the RESTORE SYSTEM utility LOGAPPLY. This type of log record will always be written to the active log datasets of the DB2 member that is running the RECOVER job, although it may compensate the log record which was originally from another DB2 member. The member ID part of this log record will always store the ID of the DB2 member that is running the RECOVER job.

During the LOGUNDO phase, if either of the following conditions exist:

- Applying the log record to a data page causes the data on the page logically inconsistent.
- The UNDO log record has already been applied and the page is now physically inconsistent.

The page is flagged as broken and the pseudo compensation log record of the log record that is being backed out is written to the active log dataset. All of the subsequent log apply processes on this data page are skipped, but the pseudo compensation log records of the skipped log records are written to the active log dataset. The log apply process continues for other pages in the same table space and for other objects in the recover list. For each error that is encountered, a DSNI012I message is issued. At the end, the RECOVER utility completes with return code 8.

If an error occurs on the index during the LOGUNDO phase, the entire index is marked as REBUILD-pending (RBDP) and no further log is applied on this index. You have to rebuild this index after the RECOVER utility completes with return code 8.

UTILTERM phase

The RECOVER utility enters the UTILTERM phase, which is an existing phase of the RECOVER utility.

The RECOVER TOLOGPOINT option in a non-data sharing system:

You can use the RECOVER utility with the TOLOGPOINT option to recover a non-data sharing DB2 subsystem.

Figure 60 shows a non-data sharing system. Table spaces TS1 and TS2 are being recovered to time TR using the RECOVER TORBA option. TS1 and TS2 are listed in the same RECOVER job. UR1 was inflight at TR time, the start time was T1, at time T2 it updated TS1, and at time T3 it updated TS2. T2 is earlier than T3. UR2 was aborting at TR time, the start time was T4, and at time T5 it updated TS2.

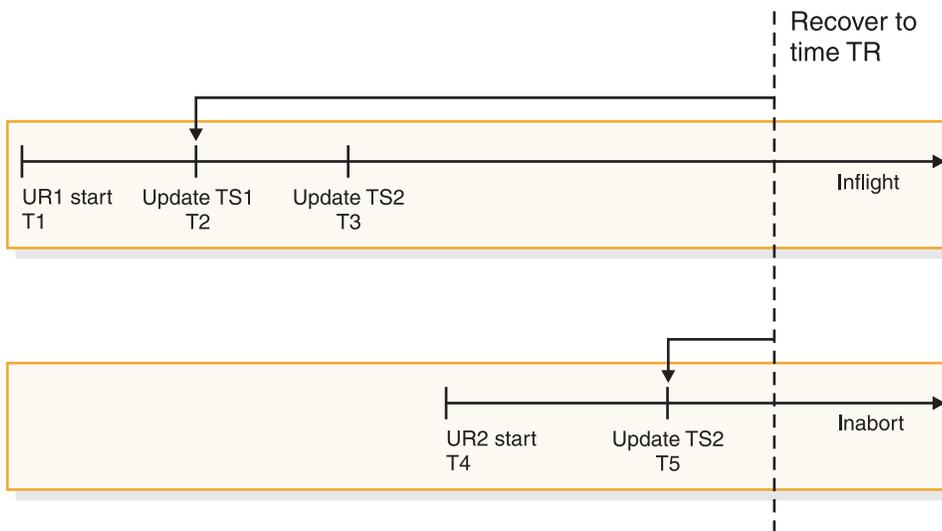


Figure 60. Using the RECOVER TOLOGPOINT option in a non-data sharing system

RECOVER utility job output messages

The RECOVER utility takes the following actions and provides the following messages during recovery in a non-data sharing system.

LOGCSR phase

After the LOGAPPLY phase, the RECOVER utility enters the log analysis phase, known as the LOGCSR phase. The following messages are issued during this phase:

DSNU1550I

Indicates the start of log analysis.

DSNU1551I

Indicates the end of log analysis.

DSNU1552I

Indicates the end of the LOGCSR phase of the RECOVER utility.

DSNU1553I

Issued after the end of the LOGCSR phase. The following information is shown in the message:

- UR1 modified TS1 at T2 time.
- UR1 modified TS2 at T3 time.
- UR2 modified TS2 at T5 time.

LOGUNDO phase

The RECOVER utility enters the LOGUNDO phase. The following messages are issued during this phase:

DSNU1554I

Indicates the start of backout.

- Backout is performed on TS2 and TS1 and log records are read from time TR to T2.
- There may be one or more DSNU1555I messages showing the backout progress.

DSNU1556I

Indicates the end of backout.

DSNU1557I

Indicates the end of LOGUNDO phase of the RECOVER utility.

A special type of compensation log record will be written during the LOGUNDO phase of log apply, known as the *pseudo* compensation log record in the following context. For each undo log record applied during the LOGUNDO phase, there will be one pseudo compensation log record written for it. This is a REDO-only log record and not units of recovery (UR) related. It will only be used for future log apply in the case that you first recover objects to a point in time with consistency and then recover the same objects again to currency using the same image copy used by previous recover job.

This type of log record will be skipped by DB2 restart and the RESTORE SYSTEM utility LOGAPPLY. This type of log record will always be written to the active log datasets of the DB2 member that is running the RECOVER job, although it may compensate the log record which was originally from another DB2 member. The member ID part of this log record will always store the ID of the DB2 member that is running the RECOVER job.

During the LOGUNDO phase, if either of the following conditions exist:

- Applying the log record to a data page causes the data on the page logically inconsistent.
- The UNDO log record has already been applied and the page is now physically inconsistent.

The page is flagged as broken and the pseudo compensation log record of the log record that is being backed out is written to the active log dataset. All of the subsequent log apply processes on this data page are skipped, but the pseudo compensation log records of the skipped log records are written to the active log dataset. The log apply process continues for other pages in the same table space and for other objects in the recover list. For each error that is encountered, a DSNI012I message is issued. At the end, the RECOVER utility completes with return code 8.

If an error occurs on the index during the LOGUNDO phase, the entire index is marked as REBUILD-pending (RBDP) and no further log is applied on this index. You have to rebuild this index after the RECOVER utility completes with return code 8.

UTILTERM phase

The RECOVER utility enters the UTILTERM phase, which is an existing phase of the RECOVER utility.

Recommendations for recovery of compressed data:

Use care when recovering a single data set of a non-partitioned page set to a prior point in time. If the data set that is recovered was compressed with a different dictionary from the rest of the page set, you can no longer read the data.

The RECOVER utility does not reset the values that DB2 generates for identity columns.

Related reference

"LOAD" (DB2 Utility Guide and Reference)

"RECOVER" (DB2 Utility Guide and Reference)

Recovering by using DB2 restart recovery

You can use normal DB2 restart recovery to recover your DB2 system to the point-in-time of a backup that was created by using the BACKUP SYSTEM utility.

To recover your DB2 system to the point-in-time of a backup by using normal DB2 restart recovery:

1. Back up your system by issuing the BACKUP SYSTEM FULL command.
DFSMSHsm maintains up to 85 versions of system backups on disk at any given time.
2. Recover the system:
 - a. Stop the DB2 subsystem. For data sharing systems, stop all members of the group.
 - b. Use the DFSMSHsm command FRRECOV * COPYPOOL(*cpname*) GENERATION(*gen*) to restore the database and log copy pools that the BACKUP SYSTEM utility creates. In this command, *cpname* specifies the name of the copy pool, and *gen* specifies which version of the copy pool is to be restored.
 - c. For data sharing systems, delete all CF structures that are owned by this group.
 - d. Restore any logs on tape to disk.
 - e. Start DB2. For data sharing systems, start all active members.

- f. For data sharing systems, execute the GRECP and LPL recovery, which recovers the changed data that was stored in the coupling facility at the time of the backup.

Related concepts

“Point-in-time recovery with system-level backups” on page 556

Recovering by using FlashCopy backups

You can use FlashCopy backups to recover your DB2 system to the point-in-time of a backup.

For more information about the FlashCopy function, see *z/OS DFSMS Advanced Copy Services*.

To recover by using FlashCopy backups:

1. Back up your system:
 - a. Issue the DB2 command SET LOG SUSPEND to suspend logging and update activity, and to quiesce 32KB page writes and data set extensions. For data sharing systems, issue the command to each member of the group.
 - b. Use the FlashCopy function to copy all DB2 volumes. Include any ICF catalogs that are used by DB2, as well as active logs and BSDSs.
 - c. Issue the DB2 command SET LOG RESUME to resume normal DB2 update activity. To save disk space, you can use DFSMSdss to dump the disk copies that you just created to a lower-cost medium, such as tape.
2. Recover your system:
 - a. Stop the DB2 subsystem. For data sharing systems, stop all members of the group.
 - b. Use DFSMSdss RESTORE to restore the FlashCopy data sets to disk. See *z/OS DFSMSdss Storage Administration Reference* for more information.
 - c. For data sharing systems, delete all CF structures that are owned by this group.
 - d. Start DB2. For data sharing systems, start all active members.
 - e. For data sharing systems, execute the GRECP and LPL recovery, which recovers the changed data that was stored in the coupling facility at the time of the backup.

Making catalog definitions consistent with your data after recovery to a prior point in time

Avoiding point-in-time recovery of the catalog is easier than attempting to correct the inconsistencies that the recovery causes.

If you choose to recover catalog and directory tables to a prior point in time, you need to first shut down the DB2 system cleanly and then restart in ACCESS(MAINT) mode before the recovery.

If you recover catalog tables to a prior point in time, you must perform the following actions to make catalog definitions consistent with your data.

1. Run the DSN1PRNT utility with the PARM=(FORMAT, NODATA) option on all data sets that might contain user table spaces. The NODATA option suppresses all row data, which reduces the output volume that you receive. Data sets that contain user tables are of the following form, where *y* can be either I or J:
`catname.DSNDBC.dbname.tsname.y0001.A00n`

2.  Execute the following SELECT statements to find a list of table space and table definitions in the DB2 catalog:

```
SELECT NAME, DBID, PSID FROM SYSIBM.SYSTABLESPACE;  
SELECT NAME, TSNAME, DBID, OBID FROM SYSIBM.SYSTABLES;
```



3. For each table space name in the catalog, look for a data set with a corresponding name. If a data set exists, take the following additional actions:
- Find the field HPGOBID in the header page section of the DSN1PRNT output. This field contains the DBID and PSID for the table space. Check if the corresponding table space name in the DB2 catalog has the same DBID and PSID.
 - If the DBID and PSID do not match, execute DROP TABLESPACE and CREATE TABLESPACE statements to replace the incorrect table space entry in the DB2 catalog with a new entry. Be sure to make the new table space definition exactly like the old one. If the table space is segmented, SEGSIZE must be identical for the old and new definitions.

You can drop a LOB table space only if it is empty (that is, it does not contain auxiliary tables). If a LOB table space is not empty, you must first drop the auxiliary table before you drop the LOB table space. To drop auxiliary tables, you can perform one of the following actions:

 - Drop the base table.
 - Delete all rows that reference LOBs from the base table, and then drop the auxiliary table.
 - Find the PGSOBD fields in the data page sections of the DSN1PRNT output. These fields contain the OBIDs for the tables in the table space. For each OBID that you find in the DSN1PRNT output, search the DB2 catalog for a table definition with the same OBID.
 - If any of the OBIDs in the table space do not have matching table definitions, examine the DSN1PRNT output to determine the structure of the tables that are associated with these OBIDs. If you find a table whose structure matches a definition in the catalog, but the OBIDs differ, proceed to the next step. The OBIDXLAT option of DSN1COPY corrects the mismatch. If you find a table for which no table definition exists in the catalog, re-create the table definition by using the CREATE TABLE statement. To re-create a table definition for a table that has had columns added, first use the **original** CREATE TABLE statement, and then use ALTER TABLE to add columns, which makes the table definition match the current structure of the table.
 - Use the DSN1COPY utility with the OBIDXLAT option to copy the existing data to the new tables in the table space, and translate the DBID, PSID, and OBIDs.

If a table space name in the DB2 catalog does not have a data set with a corresponding name, one of the following events has probably occurred:

- The table space was dropped after the point in time to which you recovered. In this case, you cannot recover the table space. Execute DROP TABLESPACE to delete the entry from the DB2 catalog.
- The table space was defined with the DEFINE(NO) option. In this case, the data set is allocated when you insert data into the table space.

4. For each data set in the DSN1PRNT output, look for a corresponding DB2 catalog entry. If no entry exists, follow the instructions in “Recovering an accidentally dropped table space” on page 578 to re-create the entry in the DB2 catalog.
5. If you recover the catalog tables SYSSEQ and SYSSEQ2, identity columns and sequence objects are inconsistent. To avoid duplicate identity column values, recover all table spaces that contain tables that use identity columns to the point in time to which you recovered SYSSEQ and SYSSEQ2. To eliminate gaps between identity column values, use the ALTER TABLE statement. For sequence objects, use the ALTER SEQUENCE statement to eliminate these gaps.
6. Ensure that the IPREFIX values of user table spaces and index spaces that were reorganized match the IPREFIX value in the VSAM data set names that are associated with each table space or partition. If the IPREFIX that is recorded in the DB2 catalog and directory is different from the VSAM cluster names, you cannot access your data. To ensure that these IPREFIX values match, complete the following procedure:
 - a. Query the SYSIBM.SYSTABLEPART and SYSIBM.SYSINDEXPART catalog tables to determine the IPREFIX value that is recorded in the catalog for objects that were reorganized.
 - b. Compare this IPREFIX value to the IPREFIX value in the VSAM data set name that is associated with the table space or index space.
 - c. When IPREFIX values do not match for an object, rename the VSAM data set to specify the correct IPREFIX.

Important: For objects involved in cloning, rename the base and clone objects at the same time.

Example: Assume that the catalog specifies an IPREFIX of J for an object but the VSAM data set that corresponds to this object is .

catname.DSNDBC.dbname.spname.I0001.A001

You must rename this data set to:

catname.DSNDBC.dbname.spname.J0001.A001

7. Delete the VSAM data sets that are associated with table spaces that were created with the DEFINE NO option and that reverted to an unallocated state. After you delete the VSAM data sets, you can insert or load rows into these unallocated table spaces to allocate new VSAM data sets.

Related concepts

“Recovery of tables that contain identity columns” on page 570

Related reference

“CREATE TABLESPACE” (DB2 SQL Reference)

“CREATE INDEX” (DB2 SQL Reference)

“DSN1COPY” (DB2 Utility Guide and Reference)

“DSN1PRNT” (DB2 Utility Guide and Reference)

Recovery of catalog and directory tables

Recovering catalog and directory tables to a prior point in time is strongly discouraged for several reasons:

- You must recover all table spaces that are associated with the catalog tables that you recover to the same point in time. For example, if you recover any table space in the DB2 catalog (DSNDB06) and directory (DSNDB01), all table spaces (except SYSUTILX) must be recovered.

The catalog and directory contain definitions of all databases. When databases DSNDB01 and DSNDB06 are restored to a prior point, information about later definitions, authorizations, binds, and recoveries is lost. If you restore the catalog and directory, you might need to restore user databases; if you restore user databases, you might need to restore the catalog and directory.

- You might create catalog definitions that are inconsistent with your data. These catalog and data inconsistencies are usually the result of one of the following actions:
 - A catalog table space was restored.
 - SYSSEQ and SYSSEQ2 were recovered to a prior point in time.
 - The definition of a table, table space, index, or index space was changed after the data was last backed up.
- You can cause problems for user table spaces or index spaces that have been reorganized with SHRLEVEL REFERENCE or SHRLEVEL CHANGE.
- You can cause a populated VSAM data set that was defined with DEFINE NO option to revert back to the undefined state. To avoid errors, you must delete the existing VSAM data sets before the table space or index can be accessed.

Performing remote site recovery from a disaster at a local site

After a disaster at your local site, you can recover at a remote site by using the RESTORE SYSTEM utility.

You can use RESTORE SYSTEM to recover DB2 from the backups that the BACKUP SYSTEM utility produces, or you can use RESTORE SYSTEM LOGONLY to recover from backups that you produce in some other way. For DB2 remote-site recovery procedures that do not use the RESTORE SYSTEM utility, see “Performing remote-site disaster recovery” on page 679.

Recovering with BACKUP SYSTEM

You can use the BACKUP SYSTEM and RESTORE SYSTEM utilities to recover your DB2 system.

The following procedures outline this recovery method.

1. Prepare for recovery:
 - a. Use BACKUP SYSTEM FULL to take the system backup.
 - b. Transport the system backups to the remote site.
2. Recover:
 - a. Run the DSNJU003 utility using either of the following control statements:
 - In this control statement, substitute *log-truncation-point* with the RBA or LRSN of the point to which you want to recover
CRESTART CREATE, SYSPITR=*log-truncation-point*
 - where .
 - In this control statement, substitute *log-truncation-timestamp* with the timestamp of the point to which you want to recover.
CRESTART CREATE, SYSPITRT=*log-truncation-timestamp*
 - b. Start DB2.
 - c. Run the RESTORE SYSTEM utility by issuing the RESTORE SYSTEM control statement.

This utility control statement performs a recovery to the current time (or to the time of the last log transmission from the local site).

Recovering without BACKUP SYSTEM

You can recover your DB2 system if you do not use the BACKUP SYSTEM utility to produce backups

The following procedures outline this recovery method.

1. Prepare for recovery.
 - a. Issue the DB2 command SET LOG SUSPEND to suspend logging and update activity, and to quiesce 32KB page writes and data set extensions. For data sharing systems, issue the command to each member of the data sharing group.
 - b. Use the FlashCopy function to copy all DB2 volumes. Include any ICF catalogs that are used by DB2, as well as active logs and BSDSs.
 - c. Issue the DB2 command SET LOG RESUME to resume normal DB2 activity.
 - d. Use DFSMSdss to dump the disk copies that you just created to tape, and then transport this tape to the remote site. You can also use other methods to transmit the copies that you make to the remote site.
2. Recover:
 - a. Use DFSMSdss to restore the FlashCopy data sets to disk.
 - b. Run the DSNJU003 utility by using the CRESTART CREATE, SYSPITR=*log-truncation-point* control statement. The *log-truncation-point* is the RBA or LRSN of the point to which you want to recover.
 - c. Restore any logs on tape to disk.
 - d. Start DB2.
 - e. Run the RESTORE SYSTEM utility using the RESTORE SYSTEM LOGONLY control statement to recover to the current time (or to the time of the last log transmission from the local site).

Backup and recovery involving clone tables

When performing recovery on a clone table that has been exchanged, you can use an image copy that was made prior to an exchange, but no point in time recovery is possible prior to the most recent exchange.

Clone tables spaces and index spaces are stored in separate physical data sets. You must copy them and recover them separately. Output from the REPORT utility includes information about clone tables, if they exist.

The QUIESCE command and the COPY and RECOVER utilities each use the CLONE keyword to function on clone objects:

- Running QUIESCE with the CLONE keyword establishes a quiesce point for a clone object.
- Running the COPY utility with the CLONE keyword takes a copy of a clone object.
- Running the RECOVER utility with the CLONE keyword recovers a clone object.

Related concepts

"Types of DB2 tables" (Introduction to DB2 for z/OS)

Data restore of an entire system

The RESTORE SYSTEM utility invokes z/OS Version 1 Release 5 DFSMSHsm services to recover a DB2 system to a prior point in time by restoring the databases

| in the volume copies that have been provided by the BACKUP SYSTEM utility.
| After restoring the data, this utility can then recover to a given point in time.

| The SYSPITR option of DSNJU003 CRESTART allows you to create a conditional
| restart control record (CRCR) to truncate logs for system point-in-time recovery in
| preparation for running RESTORE SYSTEM.

| The SYSPITRT option of DSNJU003 CRESTART allows you to add a timestamp, in
| the ENDTIME format, to truncate logs for system point-in-time recovery restart.

| You can specify a value of FFFFFFFFFF to cause system point-in-time recovery
| to occur without log truncation.

| **Related reference**

| "DSNJU003 (change log inventory)" (DB2 Utility Guide and Reference)

| **Related information**

| "RESTORE SYSTEM" (DB2 Utility Guide and Reference)

Chapter 20. Recovering from different DB2 for z/OS problems

When you use DB2, occasional problems might occur. You can troubleshoot and recover from many problems on your own by using recovery procedures.

Recovering from IRLM failure

You can recover from an IRLM failure, regardless of whether the failure results in a wait, loop, or abend.

Symptoms

The IRLM waits, loops, or abends. The following message might be issued:

```
DXR122E irllmm ABEND UNDER IRLM TCB/SRB IN MODULE xxxxxxxx  
ABEND CODE zzzz
```

Environment

If the IRLM abends, DB2 terminates. If the IRLM waits or loops, the IRLM terminates, and DB2 terminates automatically.

Resolving the problem

Operator response:

1. Start the IRLM if you did not set it for automatic start when you installed DB2.
2. Start DB2.
3. Connect IMS to DB2, by issuing the following command, where *ssid* is the subsystem ID:

```
/START SUBSYS ssid
```

4. Connect CICS to DB2 by issuing the following command:

```
DSNC STRT
```

Related tasks

“Starting the IRLM” on page 430

“Starting DB2” on page 365

“Connecting from CICS” on page 442

Recovering from z/OS or power failure

You can recover from a situation in which z/OS or your processor power fails.

Symptoms

No processing is occurring.

Resolving the problem

Operator response:

- If the power failure or z/OS failure has occurred:
 1. IPL z/OS, and initialize the job entry subsystem (JES).
 2. If you normally run VTAM with DB2, start VTAM at this point.

3. Start the IRLM if it was not set for automatic start during DB2 installation.
4. Start DB2.
5. Use the RECOVER POSTPONED command if postponed-abort units of recovery were reported after restarting DB2, and if the AUTO option of the LIMIT BACKOUT field on installation panel DSNTIPL was not specified.
6. Restart IMS or CICS.
 - IMS automatically connects and resynchronizes when it is restarted.
 - CICS automatically connects to DB2 if the CICS PLT contains an entry for the attachment facility module DSNCCOM0. Alternatively, use the command DSNC STRT to connect the CICS attachment facility to DB2.
- If you know that a power failure is imminent, issue a STOP DB2 MODE(FORCE) command to allow DB2 to stop cleanly before the power is interrupted. If DB2 is unable to stop completely before the power failure, the situation is no worse than if DB2 were still operational.

Related concepts

“Connections to the IMS control region” on page 448

Related tasks

“Starting the IRLM” on page 430

“Starting DB2” on page 365

“Connecting from CICS” on page 442

Recovering from disk failure

When a disk hardware failure occurs and an entire unit is lost, you can recover.

Symptoms

No I/O activity occurs for the affected disk address. Databases and tables that reside on the affected unit are unavailable.

Resolving the problem

Operator response:

1. Assure that no incomplete I/O requests exist for the failing device. One way to do this is to force the volume offline by issuing the following z/OS command, where *xxx* is the unit address:

```
VARY xxx,OFFLINE,FORCE
```

To check disk status, issue the following command:

```
D U,DASD,ONLINE
```

The following console message is displayed after you force a volume offline:

```
UNIT  TYPE  STATUS  VOLSER  VOLSTATE
4B1   3390  0-BOX  XTRA02  PRIV/RSDNT
```

The disk unit is now available for service.

If you previously set the I/O timing interval for the device class, the I/O timing facility terminates all requests that are incomplete at the end of the specified time interval, and you can proceed to the next step without varying the volume offline. You can set the I/O timing interval either through the IECIOSxx z/OS parameter library member or by issuing the following z/OS command:

```
SETIOS MIH,DEV=devnum,IOTIMING=mm:ss.
```

- Issue (or request that an authorized operator issue) the following DB2 command to stop all databases and table spaces that reside on the affected volume:

```
-STOP DATABASE(database-name) SPACENAM(space-name)
```

If the disk unit must be disconnected for repair, stop all databases and table spaces on all volumes in the disk unit.

- Select a spare disk pack, and use ICKDSF to initialize from scratch a disk unit with a different unit address (*yyy*) and the same volume serial number (VOLSER).

```
// Job
//ICKDSF EXEC PGM=ICKDSF
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
        REVAL UNITADDRESS(yyy) VERIFY(volser)
```

If you initialize a 3380 or 3390 volume, use REVAL with the VERIFY parameter to ensure that you initialize the intended volume, or to revalidate the home address of the volume and record 0. Alternatively, use ISMF to initialize the disk unit.

- Issue the following z/OS console command, where *yyy* is the new unit address:

```
VARY yyy,ONLINE
```

- To check disk status, issue the following command:

```
D U,DASD,ONLINE
```

The following console message is displayed:

```
UNIT  TYPE  STATUS  VOLSER  VOLSTATE
7D4   3390  0         XTRA02  PRIV/RSDNT
```

- Issue the following DB2 command to start all the appropriate databases and table spaces that were previously stopped:

```
-START DATABASE(database-name) SPACENAM(space-name)
```

- Delete all table spaces (VSAM linear data sets) from the ICF catalog by issuing the following access method services command for each one of them, where *y* is either I or J:

```
DELETE catnam.DSNDBC.dbname.tsname.y0001.A00x CLUSTER NOSCRATCH
```

- For user-managed table spaces, define the VSAM cluster and data components for the new volume by issuing the access method services DEFINE CLUSTER command with the same data set name as in the previous step, in the following format: *catnam*.DSNDBC.*dbname*.*tsname*.y0001.A00x. The *y* is I or J, and the *x* is C (for VSAM clusters) or D (for VSAM data components).

- For a user-defined table space, define the new data set before an attempt to recover it. You can recover table spaces that are defined in storage groups without prior definition.

- Recover the table spaces by using the DB2 RECOVER utility.

Related reference

"RECOVER" (DB2 Utility Guide and Reference)

Related information

 z/OS Internet Library

 z/OS Internet Library

 z/OS Internet Library

 z/OS Internet Library

Recovering from application errors

You can recover from a problem in which an application program placed a logically incorrect value in a table.

Symptoms

Unexpected data is returned from an SQL SELECT statement, even though the SQLCODE that is associated with the statement is 0.

Causes

An SQLCODE of 0 indicates that DB2 and SQL did not cause the problem, so the cause of the incorrect data in the table is the application.

Resolving the problem

System programmer response: You might be able to use the DB2 RECOVER utility with the TOLOGPOINT option to restore the database to a point before the error occurred. However, in many circumstances you must manually back out the changes that were introduced by the application. Among those circumstances are:

- Other applications changed the database after the error occurred. If you recover the table spaces that were modified by the bad application, all subsequent changes that were made by the other applications are lost.
- DB2 checkpoints were taken after the error occurred. In this case, you can use RECOVER TOLOGPOINT to restore the data up to the last checkpoint before the error occurred. However, all subsequent changes to the database are lost.

If you have a situation for which using RECOVER TOLOGPOINT is appropriate, you can use one of the following procedures as a basis for backing out the incorrect changes that were made by the application. The procedure that you use depends on whether you have established a quiesce point.

Backing out incorrect application changes (with a quiesce point)

If you have an established quiesce point, you can back out incorrect changes that your application made.

To back out the incorrect changes:

1. Run the REPORT utility twice, once using the RECOVERY option and once using the TABLESPACESET option. On each run, specify the table space that contains the inaccurate data. If you want to recover to the last quiesce point, specify the option CURRENT when running REPORT RECOVERY.
2. Examine the REPORT output to determine the RBA of the quiesce point.
3. Run RECOVER TOLOGPOINT with the RBA that you found, specifying the names of all related table spaces.

Recovering all related table spaces to the same quiesce point prevents violations of referential constraints.

Backing out incorrect application changes (without a quiesce point)

Even if you do not have an established quiesce point, you can back out incorrect changes that your application made. Be aware, however, that if you use this procedure, you lose any updates to the database that occurred after the last checkpoint and before the application error occurred.

To back out the incorrect changes:

1. Run the DSN1LOGP stand-alone utility on the log scope that is available at DB2 restart, using the SUMMARY(ONLY) option.
2. Determine the RBA of the most recent checkpoint before the first bad update occurred, from one of the following sources:
 - Message DSNR003I on the operator's console, which looks similar to this message:

```
DSNR003I RESTART ..... PRIOR CHECKPOINT RBA=000007425468
```

The required RBA in this example is X'7425468'.

This technique works only if no checkpoints have been taken since the application introduced the bad updates.
 - Output from the print log map utility. You must know the time that the first bad update occurred. Find the last BEGIN CHECKPOINT RBA before that time.
3. Run DSN1LOGP again, using SUMMARY(ONLY), and specify the checkpoint RBA as the value of RBASTART. The output lists the work in the recovery log, including information about the most recent complete checkpoint, a summary of all processing, and an identification of the databases that are affected by each active user.
4. Find the unit of recovery in which the error was made. One of the messages in the output (identified as DSN1151I or DSN1162I) describes the unit of recovery in which the error was made. To find the unit of recovery, use your knowledge of the time that the program was run (START DATE= and TIME=), the connection ID (CONNID=), authorization ID (AUTHID=), and plan name (PLAN=). In that message, find the starting RBA as the value of START=.
5. Run the DB2 RECOVER utility with the TOLOGPOINT option, and specify the starting RBA that you found in the previous step.
6. Recover any related table spaces or indexes to the same point in time.

Related concepts

"DSN1LOGP summary report" on page 636

Related reference

"DSN1LOGP" (DB2 Utility Guide and Reference)

Recovering from IMS-related failures

When you work in a DB2-IMS environment and problems occur, you can recover from those problems.

Symptoms

Problems that occur in a DB2-IMS environment can result in a variety of symptoms:

- An IMS wait, loop, or abend is accompanied by a DB2 message that goes to the IMS console. This symptom indicates an IMS control region failure.

- When IMS connects to DB2, DB2 detects one or more units of recovery that are indoubt.
- When IMS connects to DB2, DB2 detects that it has committed one or more units of recovery that IMS indicates should be rolled back.
- Messages are issued to the IMS master terminal, to the logical terminal, or both to indicate that some sort of IMS or DB2 abend has occurred.

Environment

DB2 can be used in an XRF (Extended Recovery Facility) recovery environment with IMS.

To resolve IMS-related problems, follow the appropriate procedure.

Related concepts

“Plans for extended recovery facility toleration” on page 532

Recovering from IMS control region failure

You can recover from a problem in which the IMS control region fails.

Symptoms

- IMS waits, loops, or abends.
- DB2 attempts to send the following message to the IMS master terminal during an abend:

```
DSNM002 IMS/TM xxxx DISCONNECTED FROM SUBSYSTEM yyyy RC=RC
```

This message cannot be sent if the failure prevents messages from being displayed.

- DB2 does not send any messages for this problem to the z/OS console.

Environment

- DB2 detects that IMS has failed.
- DB2 either backs out or commits work that is in process.
- DB2 saves indoubt units of recovery, which need to be resolved at reconnection time.

Resolving the problem

Operator response: Use normal IMS restart procedures, which include starting IMS by issuing the z/OS START IMS command. The following results occur:

1. All DL/I and DB2 updates that have not been committed are backed out.
2. IMS is automatically reconnected to DB2.
3. IMS passes the recovery information for each entry to DB2 through the IMS attachment facility. (IMS indicates whether to commit or roll back.)
4. DB2 resolves the entries according to IMS instructions.

Recovering from IMS indoubt units of recovery

When IMS connects to DB2, and DB2 has one or more indoubt units of recovery that have not been resolved, these units of recovery need to be resolved.

Symptoms

If DB2 has indoubt units of recovery that IMS did not resolve, the following message is issued at the IMS master terminal, where *xxxx* is the subsystem identifier:

```
DSNM004I RESOLVE INDOUBT ENTRY(S) ARE OUTSTANDING FOR SUBSYSTEM xxxx
```

Causes

When this message is issued, IMS was either cold started, or it was started with an incomplete log tape. Message DSNM004I might also be issued if DB2 or IMS abnormally terminated in response to a software error or other subsystem failure.

Environment

- The connection remains active.
- IMS applications can still access DB2 databases.
- Some DB2 resources remain locked out.

If the indoubt thread is not resolved, the IMS message queues might start to back up. If the IMS queues fill to capacity, IMS terminates. Be aware of this potential difficulty, and monitor IMS until the indoubt units of work are fully resolved.

Resolving the problem

System programmer response:

1. Force the IMS log closed by using the /DBR FEOV command.
2. Archive the IMS log.
3. Issue the command DFSERA10 to print the records from the previous IMS log tape for the last transaction that was processed in each dependent region. Record the PSB and the commit status from the X'37' log that contains the recovery ID.
4. Run the DL/I batch job to back out each PSB that is involved that has not reached a commit point. The process might be time-consuming because transactions are still being processed. This process might also lock a number of records, which could affect the rest of the processing and the rest of the message queues.
5. Enter the DB2 command DISPLAY THREAD (*imsid*) TYPE (INDOUBT).
6. Compare the NIDs (IMSID + OASN in hexadecimal) that are displayed in the DISPLAY THREAD output with the OASNs (4 bytes decimal) as shown in the DFSERA10 output. Decide whether to commit or roll back.
7. Use DFSERA10 to print the X'5501FE' records from the current IMS log tape. Every unit of recovery that undergoes indoubt resolution processing is recorded; each record with an 'IDBT' code is still indoubt. Note the correlation ID and the recovery ID, for use during the next step.
8. **GUPI** Enter the following DB2 command, choosing to commit or roll back, and specify the correlation ID:

```
-RECOVER INDOUBT (imsid) ACTION(COMMIT|ABORT) NID (nid)
```

If the command is rejected because of associated network IDs, use the same command again, substituting the recovery ID for the network ID.

GUPI

Related concepts

“Duplicate IMS correlation IDs” on page 448

Recovering IMS indoubt units of work that need to be rolled back

When units of recovery between IMS and DB2 are indoubt at restart time, DB2 and IMS sometimes handle the indoubt units of recovery differently. When this happens, you might need to roll back the changes.

Symptoms

The following messages are issued after a DB2 restart:

```
DSNM005I  IMS/TM RESOVLE INDOUBT PROTOCOL PROBLEM WITH SUBSYSTEM xxxx
```

```
DFS3602I  xxxx SUBSYSTEM RESOLVE-INDOUBT FAILURE,RC=yyyy
```

Causes

The reason that these messages are issued is that indoubt units of work exist for a DB2-IMS application, and the way that DB2 and IMS handle these units of work differs.

At restart time, DB2 attempts to resolve any units of work that are indoubt. DB2 might commit some units and roll back others. DB2 records the actions that it takes for the indoubt units of work. At the next connect time, DB2 verifies that the actions that it took are consistent with the IMS decisions. If the DB2 RECOVER INDOUBT command is issued prior to an IMS attempt to reconnect, DB2 might decide to commit the indoubt units of recovery, whereas IMS might decide to roll back the units of recovery. This inconsistency results in the DSNM005I message being issued. Because DB2 tells IMS to retain the inconsistent entries, the DFS3602I message is issued when the attempt to resolve the indoubt units of recovery ends.

Environment

- The connection between DB2 and IMS remains active.
- DB2 and IMS continue processing.
- No DB2 locks are held.
- No units of work are in an incomplete state.

Resolving the problem

System programmer response: Do not use the DB2 RECOVER INDOUBT command. The problem is that DB2 was not indoubt but should have been. Database updates have probably been committed on one side (IMS or DB2) and rolled back on the other side.

1. Enter the IMS command /DISPLAY OASN SUBSYS DB2 to display the IMS list of units of recovery that need to be resolved. This command generates the list of OASNs in a decimal format, not in a hexadecimal format.
2. Issue the IMS command /CHANGE SUBSYS DB2 RESET to reset all the entries in the list. (No entries are passed to DB2.)
3. Use DFSERA10 to print the log records that were recorded at the time of failure and during restart. Look at the X'37', X'56', and X'5501FE' records at reconnect time. Notify IBM Software Support about the problem.
4. Determine what the inconsistent unit of recovery was doing by examining the log information, and manually make the IMS and DB2 databases consistent.

Related concepts

“Duplicate IMS correlation IDs” on page 448

Recovering from IMS application failure

You can recover from a situation in which an IMS application abnormally terminates in a DB2 environment.

Symptoms

The following messages are issued at the IMS master terminal and at the LTERM that entered the transaction that is involved:

```
DFS555 - TRAN tttttttt ABEND (SYSIDssss);  
          MSG IN PROCESS: xxxx (up to 78 bytes of data) timestamp  
DFS555A - SUBSYSTEM xxxx OASN yyyyyyyyyyyyyyyy STATUS COMMIT|ABORT
```

Causes

The problem might be caused by a usage error in the application or by a DB2 problem.

Environment

- The failing unit of recovery is backed out by both DL/I and DB2.
- The connection between IMS and DB2 remains active.

Resolving the problem

Operator response:

- If you think that the problem was caused by a usage error, investigate and resolve the error.
- If you think that the problem is a DB2 problem, rather than a usage error, try to diagnose the problem using standard diagnostic procedures. You might need to contact IBM Software Support if you cannot resolve the problem yourself.

Related concepts

"Techniques for debugging programs in IMS" (DB2 Application Programming and SQL Guide)

Related information

DB2 Diagnosis Guide and Reference

Recovering from a DB2 failure in an IMS environment

When DB2 fails in a DB2-IMS environment, you can recover from this situation.

Symptoms

DB2 fails or is not running, and one of the following status situations exists:

- If you specified error option Q, the program terminates with a U3051 user abend completion code.
- If you specified error option A, the program terminates with a U3047 user abend completion code.

In either of these situations, the IMS master terminal receives IMS message DFS554, and the terminal that is involved in the problem receives IMS message DFS555.

Resolving the problem

Operator response:

1. Restart DB2.
2. Follow the standard IMS procedures for handling application abends.

Recovering from CICS-related failure

When you work in a DB2-CICS environment and problems occur, you can recover from those problems.

Symptoms

Problems that occur in a DB2-CICS environment can result in a variety of symptoms:

- Messages that indicate an abend in CICS or the CICS attachment facility
- A CICS wait or a loop
- Indoubt units of recovery between CICS and DB2

Environment

DB2 can be used in an XRF (Extended Recovery Facility) recovery environment with CICS.

Resolving the problem

To resolve CICS-related problems, follow the appropriate procedure.

Related concepts

“Plans for extended recovery facility toleration” on page 532

Recovering from CICS application failures

You can recover from a CICS application abend in a DB2 environment.

Symptoms

The following message is issued at the user’s terminal:

```
DFH2206 TRANSACTION tranid ABEND abcode BACKOUT SUCCESSFUL
```

In this message, *tranid* represents the transaction that abnormally terminated, and *abcode* represents the specific abend code.

Environment

- The failing unit of recovery is backed out in both CICS and DB2.
- The connection between CICS and DB2 remains active.

Resolving the problem

Operator response: Investigate the abend by reading about the abend code.

- For an AEY9 abend, start the CICS attachment facility.
- For an ASP7 abend, determine why the CICS SYNCPOINT was unsuccessful.
- For other abends, follow appropriate diagnostic procedures.

Recovering DB2 when CICS is not operational

You can recover DB2 from a situation in which CICS is not operational.

Symptoms

Any of the following symptoms might occur:

- CICS waits or loops.
- CICS abends, as indicated by messages or dump output.

Environment

DB2 performs each of the following actions:

- Detects the CICS failure.
- Backs out inflight work.
- Saves indoubt units of recovery that need to be resolved when CICS is reconnected.

Diagnosing the problem

If you think that CICS is in a wait or loop situation, find the origin of the wait or loop. The origin might be in CICS, in CICS applications, or in the CICS attachment facility.

If you receive messages that indicate a CICS abend, examine the messages and dump output for more information.

If threads are connected to DB2 when CICS terminates, DB2 issues message DSN3201I. The message indicates that DB2 end-of-task (EOT) routines have cleaned up and disconnected any connected threads.

Resolving the problem

Operator response:

1. Correct the problem that caused CICS to terminate abnormally.
2. Do an emergency restart of CICS. The emergency restart performs each of the following actions:
 - Backs out inflight transactions that changed CICS resources
 - Remembers the transactions with access to DB2 that might be indoubt
3. Start the CICS attachment facility by entering the appropriate command for your release of CICS. The CICS attachment facility performs the following actions:
 - Initializes and reconnects to DB2
 - Requests information from DB2 about the indoubt units of recovery and passes the information to CICS
 - Allows CICS to resolve the indoubt units of recovery

Related tasks

“Connecting from CICS” on page 442

Related information

DB2 Diagnosis Guide and Reference

 CICS Transaction Server for z/OS Information Center

Recovering DB2 when the CICS attachment facility cannot connect to DB2

You can recover DB2 when the CICS attachment facility cannot connect to DB2.

Symptoms

Any of the possible symptoms can occur:

- CICS remains operational, but the CICS attachment facility abends.
- The CICS attachment facility issues a message that indicates the reason for the connection failure, or it requests a X'04E' dump.
- The reason code in the X'04E' dump indicates the reason for failure.
- CICS issues message DFH2206 that indicates that the CICS attachment facility has terminated abnormally with the DSNB abend code.
- CICS application programs that try to access DB2 while the CICS attachment facility is inactive are abnormally terminated. The code AEY9 is issued.

Environment

CICS backs out the abnormally terminated transaction and treats it like an application abend.

Resolving the problem

Operator response: Start the CICS attachment facility by entering the appropriate command for your release of CICS. After you start the CICS attachment facility, the following events occur:

1. The CICS attachment facility initializes and reconnects to DB2.
2. The CICS attachment facility requests information about the indoubt units of recovery and passes the information to CICS.
3. CICS resolves the indoubt units of recovery.

Recovering CICS indoubt units of recovery

When the CICS attachment facility abends, CICS and DB2 build lists of indoubt units of work, either dynamically or during restart, depending on the failing subsystem. If any units of recovery are indoubt at connect time, you can recover from this situation.

Symptoms

One of the following messages is sent to the user-named CICS destination that is specified for the MSGQUEUE*n(name)* attribute in the RDO (resource definition online): DSN2001I, DSN2034I, DSN2035I, or DSN2036I.

Causes

For CICS, a DB2 unit of recovery might be indoubt if the forget entry (X'FD59') of the task-related installation exit routine is absent from the CICS system journal. The indoubt condition applies only to the DB2 unit of recovery in this case because CICS already committed or backed out any changes to its resources.

A DB2 unit of recovery is indoubt for DB2 if an End Phase 1 is present and the Begin Phase 2 is absent.

Environment

The following table summarizes the situations that can exist when CICS units of recovery are indoubt.

Table 108. Situations that involve CICS abnormal indoubt units of recovery

Message ID	Meaning
DSN2001I	The named unit of recovery cannot be resolved by CICS because CICS was cold started. The CICS attachment facility continues the startup process.
DSN2034I	The named unit of recovery is not indoubt for DB2, but it is indoubt according to CICS log information. The reason is probably a CICS restart with the wrong tape. The problem might also be caused by a DB2 restart to a prior point in time.
DSN2035I	The named unit of recovery is indoubt for DB2, but it is not in the CICS indoubt list. This is probably due to an incorrect CICS restart. The CICS attachment facility continues the startup process and provides a transaction dump. The problem might also be caused by a DB2 restart to a prior point in time.
DSN2036I	CICS indicates rollback for the named unit of recovery, but DB2 has already committed the unit of recovery. The CICS attachment facility continues the startup process.

CICS retains details of indoubt units of recovery that were not resolved during connection startup. An entry is purged when it no longer shows up on the list that is presented by DB2 or, when the entry is present in the list, when DB2 resolves it.

Resolving the problem

System programmer response: If CICS cannot resolve one or more indoubt units of recovery, resolve them manually by using DB2 commands. Using the steps in this procedure is rarely necessary because it is required only where operational errors or software problems have prevented automatic resolution.

1. Obtain a list of the indoubt units of recovery from DB2 by issuing the following command:

```
-DISPLAY THREAD (connection-name) TYPE (INDOUBT)
```

Messages like these are then issued:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS - DSNV406I - INDOUBT THREADS - COORDINATOR
          STATUS      RESET URID          AUTHID
coordinator_name status yes/no urid authid
DISPLAY INDOUBT REPORT COMPLETE DSN9022I - DSNVDT '-DISPLAY THREAD' NORMAL
COMPLETION
```

The *corr_id* (correlation ID) for CICS Transaction Server for z/OS 1.2 and subsequent releases of CICS consists of:

Bytes 1 - 4

Thread type: COMD, POOL, or ENTR

Bytes 5 - 8

Transaction ID

Bytes 9 - 12

Unique thread number

Two threads can sometimes have the same correlation ID when the connection has been broken several times and the indoubt units of recovery have not been

resolved. In this case, use the network ID (NID) instead of the correlation ID to uniquely identify indoubt units of recovery.

The network ID consists of the CICS connection name and a unique number that is provided by CICS at the time that the syncpoint log entries are written. This unique number is an 8-byte store clock value that is stored in records that are written to both the CICS system log and to the DB2 log at syncpoint processing time. This value is referred to in CICS as the *recovery token*.

2. Scan the CICS log for entries that are related to a particular unit of recovery. Look for a PREPARE record (JCRSTRID X'F959'), for the task-related installation where the recovery token field (JCSRMTKN) equals the value that is obtained from the network-ID. The network ID is supplied by DB2 in the DISPLAY THREAD command output.

You can find the CICS task number by locating the prepare log record in the CICS log for indoubt units of recovery. Using the CICS task number, you can locate all other entries on the log for this CICS task.

You can use the CICS journal print utility DFHJUP to scan the log.

3. Use the change log inventory utility (DSNJU003) to scan the DB2 log for entries that are related to a particular unit of recovery. Locate the End Phase 1 record with the required network ID. Then use the URID from this record to obtain the rest of the log records for this unit of recovery.

When scanning the DB2 log, note that the DB2 startup message DSNJ099I provides the start log RBA for this session.

4. If needed, do indoubt resolution in DB2.  To invoke DB2 to take the recovery action for an indoubt unit of recovery, issue the DB2 RECOVER INDOUBT command, where the *correlation_id* is unique:

```
DSNC -RECOVER INDOUBT (connection-name)
        ACTION (COMMIT|ABORT)
        ID (correlation_id)
```

If the transaction is a pool thread, use the value of the correlation ID (*corr_id*) that is returned by DISPLAY THREAD for *thread#.tranid* in the RECOVER INDOUBT command. In this case, the first letter of the correlation ID is P. The transaction ID is in characters five through eight of the correlation ID.

If the transaction is assigned to a group (group is a result of using an entry thread), use *thread#.groupname* instead of *thread#.tranid*. In this case, the first letter of the correlation ID is a G, and the group name is in characters five through eight of the correlation ID. The *groupname* is the first transaction that is listed in a group.

Where the correlation ID is not unique, use the following command:

```
DSNC -RECOVER INDOUBT (connection-name)
        ACTION (COMMIT|ABORT)
        NID (network-id)
```

When two threads have the same correlation ID, use the NID keyword instead of the ID keyword. The NID value uniquely identifies the work unit.

To recover all threads that are associated with *connection-name*, omit the ID option.

The command results that are in either of the following messages indicate whether the thread is committed or rolled back:

```
DSNV414I - THREAD thread#.tranid COMMIT SCHEDULED
DSNV414I - THREAD thread#.tranid ABORT SCHEDULED
```

When you resolve indoubt units of work, note that CICS and the CICS attachment facility are not aware of the commands to DB2 to commit or abort indoubt units of recovery because only DB2 resources are affected. However,

CICS keeps details about the indoubt threads that could not be resolved by DB2. This information is purged either when the presented list is empty or when the list does not include a unit of recovery that CICS remembers.

Investigate any inconsistencies that you found in the preceding steps. 

Related reference

"DSNJU003 (change log inventory)" (DB2 Utility Guide and Reference)

Related information

 CICS Transaction Server for z/OS Information Center

Recovering from CICS attachment facility failure

You can recover DB2 when the CICS attachment facility abends or when a CICS attachment thread subtask abends.

Symptoms

The symptoms depend on whether the CICS attachment facility or one of its thread subtasks terminated:

- If the main CICS attachment facility subtask abends, an abend dump is requested. The contents of the dump indicate the cause of the abend. When the dump is issued, shutdown of the CICS attachment facility begins.
- If a thread subtask terminates abnormally, a X'04E' dump is issued, and the CICS application abends with a DSNB dump code. The X'04E' dump generally indicates the cause of the abend. The CICS attachment facility remains active.

Resolving the problem

Operator response: Correct the problem that caused the abend by analyzing the CICS formatted transaction dump or subtask SNAP dump. If the CICS attachment facility shuts down, use CICS commands to stop the execution of any CICS-DB2 applications.

Related information

"DB2 messages" (DB2 Messages)

Recovering from subsystem termination

You can recover DB2 after DB2 or an operator-issued cancel causes the subsystem to terminate.

Symptoms

When a DB2 subsystem terminates, the specific failure is identified in one or messages. The following messages might be issued at the z/OS console:

```
DSNV086E - DB2 ABNORMAL TERMINATION REASON=XXXXXXXXX
DSN3104I - DSN3EC00 -TERMINATION COMPLETE
DSN3100I - DSN3EC00 - SUBSYSTEM ssnm READY FOR -START COMMAND
```

The following message might be issued to the IMS master terminal:

```
DSNM002I  IMS/TM xxxx DISCONNECTED FROM SUBSYSTEM
          yyyy RC=rc
```

The following message might be issued to the CICS transient data error destination, which is defined in the RDO:

Environment

- IMS and CICS continue.
- In-process IMS and CICS applications receive SQLCODE -923 (SQLSTATE '57015') when accessing DB2.
 In most cases, if an IMS or CICS application program is running when a -923 SQLCODE is returned, an abend occurs. This is because the application program generally terminates when it receives a -923 SQLCODE. To terminate, some synchronization processing occurs (such as a commit). If DB2 is not operational when synchronization processing is attempted by an application program, the application program abends. In-process applications can abend with an abend code X'04F'.
- IMS applications that begin to run after subsystem termination begins are handled according to the error options.
 - For option R, SQL return code -923 is sent to the application, and IMS pseudo abends.
 - For option Q, the message is enqueued again, and the transaction abends.
 - For option A, the message is discarded, and the transaction abends.
- CICS applications that begin to run after subsystem termination begins are handled as follows:
 - If the CICS attachment facility has not terminated, the application receives a -923 SQLCODE.
 - If the CICS attachment facility has terminated, the application abends (code AEY9).

Resolving the problem

Operator response:

1. Restart DB2 by issuing the command START DB2.
2. For IMS environments, re-establish the IMS connection by issuing the IMS command /START SUBSYS DB2.
3. For CICS environments, re-establish the CICS connection by issuing the CICS attachment facility command DSNC STRT.

Recovering from temporary resource failure

DB2 sometimes experiences a temporary problem when it accesses log data sets. In this case, you need to recover from the situation so that processing can continue as normal.

Symptoms

DB2 issues messages for the access failure for each log data set. These messages provide information that is needed to resolve the access error. For example:

```
DSNJ104I ( DSNJR206 RECEIVED ERROR STATUS 00000004
          FROM DSNPCLOC FOR DSNAME=DSNC710.ARCHLOG1.A0000049
```

```
*DSNJ153E ( DSNJR006 CRITICAL LOG READ ERROR
            CONNECTION-ID = TEST0001
            CORRELATION-ID = CTHDCORID001
            LUWID = V71A.SYEC1DB2.B3943707629D=10
            REASON-CODE = 00D10345
```

Causes

DB2 might experience a problem when it attempts to allocate or open archive log data sets during the rollback of a long-running unit of recovery. These temporary failures can be caused by:

- A temporary problem with DFHSM recall
- A temporary problem with the tape subsystem
- Uncataloged archive logs
- Archive tape mount requests being canceled

Resolving the problem

User response: You can attempt to recover from temporary failures by issuing a positive reply (Y) to the following message:

```
*26 DSNJ154I ( DSNJR126 REPLY Y TO RETRY LOG READ REQUEST, N TO ABEND
```

If the problem persists, quiesce other work in the system before replying N, which terminates DB2.

Recovering from active log failures

A variety of active log failures might occur, but you can recover from them.

Symptoms

Most active log failures are accompanied by or preceded by error messages to inform you of out-of-space conditions, write or read I/O errors, or loss of dual active logging.

If you receive message DSNJ103I at startup time, the active log is experiencing dynamic allocation problems. If you receive message DSNJ104I, the active log is experiencing open-close problems. In either case, you should follow procedures in “Recovering from BSDS or log failures during restart” on page 626.

Recovering from being out of space in active logs

Because the available space in the active log is finite, the active log might fill to capacity for one of several reasons, such as delays in offloading and excessive logging. You can recover from out-of-space conditions in the active log.

Symptoms

The following warning message is issued when the last available active log data set is 5% full:

```
DSNJ110E - LAST COPY n ACTIVE LOG DATA SET IS nnn PERCENT FULL
```

The DB2 subsystem reissues the message after each additional 5% of the data set space is filled. Each time the message is issued, the offload process is started. IFCID trace record 0330 is also issued if statistics class 3 is active.

If the active log fills to capacity, after having switched to single logging, DB2 issues the following message, and an offload is started.

```
DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The DB2 subsystem then halts processing until an offload is completed.

Causes

The active log is out of space.

Environment

An out-of-space condition on the active log has very serious consequences. Corrective action is required before DB2 can continue processing. When the active log becomes full, the DB2 subsystem cannot do any work that requires writing to the log until an offload is completed. Until that offload is completed, DB2 waits for an available active log data set before resuming normal DB2 processing. Normal shutdown, with either a QUIESCE or FORCE command, is not possible because the shutdown sequence requires log space to record system events that are related to shutdown (for example, checkpoint records).

Resolving the problem

Operator response:

1. Ensure that the offload is not waiting for a tape drive. If it is, mount a tape. DB2 then processes the offload task.
2. If you are uncertain about what is causing the problem, enter the following command:

```
-ARCHIVE LOG CANCEL OFFLOAD
```

This command causes DB2 to restart the offload task. Issuing this command might solve the problem.

3. If issuing this command does not solve the problem, determine and resolve the cause of the problem, and then reissue the command. If the problem cannot be resolved quickly, have the system programmer define additional active logs until you can resolve the problem.

System programmer response: Define additional active log data sets so that DB2 can continue its normal operation while the problem that is causing the offload failures is corrected.

1. Use the z/OS command CANCEL to stop DB2.
2. Use the access method services DEFINE command to define new active log data sets.
3. Run utility DSNJLOGF to initialize the new active log data sets.
4. Define the new active log data sets in the BSDS by using the change log inventory utility (DSNJU003).
5. Restart DB2. Offload is started automatically during startup, and restart processing occurs.

Recommendation: To minimize the number of offloads that are taken per day in your installation, consider increasing the size of the active log data sets.

Related reference

"DSNJU003 (change log inventory)" (DB2 Utility Guide and Reference)

Recovering from a write I/O error on an active log data set

You can recover from a situation in which a write error occurs on an active log data set.

Symptoms

The following message is issued:

```
DSNJ105I - csect-name LOG WRITE ERROR DSNAME=..., LOGRBA=...,  
          ERROR STATUS= cccffss
```

Causes

Although this problem can be caused by several problems, one possible cause is a CATUPDT failure.

Environment

When a write error occurs on an active log data set, the following characteristics apply:

- DB2 marks the failing DB2 log data set TRUNCATED in the BSDS.
- DB2 goes on to the next available data set.
- If dual active logging is used, DB2 truncates the other copy at the same point.
- The data in the truncated data set is offloaded later, as usual.
- The data set is not stopped; it is reused on the next cycle. However, if a DSNJ104 message indicates a CATUPDT failure, the data set is marked STOPPED.

Resolving the problem

System programmer response: If the DSNJ104 message indicates a CATUPDT failure, use access method services and the change log inventory utility (DSNJU003) to add a replacement data set. In this case, you need to stop DB2. The timing of when you should take this action depends on how widespread the problem is.

- If the additional problem is localized and does not affect your ability to recover from any other problems, you can wait until the earliest convenient time.
- If the problem is widespread (perhaps affecting an entire set of active log data sets), stop DB2 after the next offload.

Related reference

"DSNJU003 (change log inventory)" (DB2 Utility Guide and Reference)

Recovering from a loss of dual active logging

If you use dual active logs, which is generally recommended, and one of the active log fails, DB2 reverts to use of a single active log. You can recover from this situation and return to dual active-log mode.

Symptoms

The following message is issued:

```
DSNJ004I - ACTIVE LOG COPY n INACTIVE, LOG IN SINGLE MODE,  
          ENDRBA=...
```

Causes

This problem occurs when DB2 completes one active log data set and then finds that the subsequent copy (COPY *n*) data sets have not been offloaded and are marked STOPPED.

Environment

DB2 continues in single mode until offloading completes and then returns to dual mode. If the data set is marked STOPPED, however, intervention is required.

Resolving the problem

System programmer response:

1. Verify that offload is proceeding and is not waiting for a tape mount. You might need to run the DB2 print log map utility (DSNJU004) to determine the status of all data sets.
2. If any data sets are marked STOPPED, use IDCAMS to delete the data sets, and then re-add them by using the DB2 change log inventory utility (DSNJU003).

Related reference

"DSNJU003 (change log inventory)" (DB2 Utility Guide and Reference)

Recovering from I/O errors while reading the active log

You can recover from situations in which an I/O error occurs when DB2 is reading the active log.

Symptoms

The following message is issued:

```
DSNJ106I - LOG READ ERROR DSNAME=..., LOGRBA=...,  
          ERROR STATUS=ccccffss
```

Environment

- If the error occurs during offload, offload tries to identify the RBA range from a second copy of the active log.
 - If no second copy of the active log exists, the data set is stopped.
 - If the second copy of the active log also has an error, only the original data set that triggered the offload is stopped. Then the archive log data set is terminated, leaving a discontinuity in the archived log RBA range.
 - The following message is issued:

```
DSNJ124I - OFFLOAD OF ACTIVE LOG SUSPENDED FROM RBA xxxxxx  
          TO RBA xxxxxx DUE TO I/O ERROR
```
 - If the second copy of the active log is satisfactory, the first copy is not stopped.
- If the error occurs during recovery, DB2 provides data from specific log RBAs that are requested from another copy or archive. If this is unsuccessful, recovery fails and the transaction cannot complete, but no log data sets are stopped. However, the table space that is being recovered is not accessible.

Resolving the problem

System programmer response:

- If the problem occurred during offload, determine which databases are affected by the active log problem, and take image copies of those. Then proceed with a new log data set.
- You can use the IDCAMS REPRO command to archive as much of the stopped active log data set as possible. Then run the change log inventory utility to notify the BSDS of the new archive log and its log RBA range. Repairing the active log does not solve the problem because offload does not go back to unload it.

- If the active log data set has been stopped, it is not used for logging. The data set is not deallocated; it is still used for reading.
- If the data set is not stopped, an active log data set should nevertheless be replaced if persistent errors occur. The operator is not told explicitly whether the data set has been stopped. To determine the status of the active log data set, run the print log map utility (DSNJU004).
- If you need to replace the data set:
 1. Ensure that the data is saved.

If you have dual active logs, the data is saved on the other active log, which becomes your new data set. Skip to step 4.

If you are not using dual active logs, take the following steps to determine whether the data set with the error has been offloaded:

 - a. Use the print log map utility (DSNJU004) to list information about the archive log data sets from the BSDS.
 - b. Search the list for a data set whose RBA range includes the range of the data set with the error.
 2. If the data set with the error has been offloaded (that is, if the value for high RBA offloaded in the print log map utility output is greater than the RBA range of the data set with the error), manually add a new archive log to the BSDS by using the change log inventory utility (DSNJU003). Use IDCAMS to define a new log that has the same LRECL and BLKSIZE values as defined in DSNZPxxx. You can use the access method services REPRO command to copy a data set with the error to the new archive log. If the archive log is not cataloged, DB2 can locate it from the UNIT and VOLSER values in the BSDS.
 3. If an active log data set has been stopped, an RBA range has not been offloaded; copy from the data set with the error to a new data set. If additional I/O errors prevent you from copying the entire data set, a gap occurs in the log and restart might fail, although the data still exists and is not overlaid. If this occurs, see “Recovering from BSDS or log failures during restart” on page 626.
 4. Stop DB2, and use the change log inventory utility to update information in the BSDS about the data set with the error.
 - a. Use DELETE to remove information about the bad data set.
 - b. Use NEWLOG to name the new data set as the new active log data set and to give it the RBA range that was successfully copied.

The DELETE and NEWLOG operations can be performed by the same job step; put DELETE before NEWLOG in the SYSIN input data set. This step clears the stopped status, and DB2 eventually archives it.
 - c. Delete the data set that is in error by using access method services.
 - d. Redefine the data set so that you can write to it. Use access method services DEFINE command to define the active log data sets. Run utility DSNJLOGF to initialize the active log data sets. If using dual logs, use access method services REPRO to copy the good log into the redefined data set so that you have two consistent, correct logs again.

Related reference

- “DSNJU004 (print log map)” (DB2 Utility Guide and Reference)
- “2. PRIMARY QUANTITY” (DB2 Installation Guide)

Recovering from archive log failures

You can recover from situations in which archive logging fails.

Symptoms

Archive log failures can result in a variety of DB2 and z/OS messages that identify problems with archive log data sets.

One specific symptom that might occur is message DSNJ104I, which indicates an open-close problem on the archive log.

Recovering from allocation problems with the archive log

You can recover from situations in which allocation problems occur for the archive log.

Symptoms

The following message is issued:

```
DSNJ103I - csect-name LOG ALLOCATION ERROR DSNAME=dsname,  
ERROR STATUS=eeeeiii, SMS REASON CODE=sssssss
```

z/OS dynamic allocation provides the ERROR STATUS information. If the allocation is for offload processing, the following message is also issued:

```
DSNJ115I - OFFLOAD FAILED, COULD NOT ALLOCATE AN ARCHIVE DATA SET
```

Causes

Archive log allocation problems can occur when various DB2 operations fail; for example:

- The RECOVER utility executes and requires an archive log. If neither archive log can be found or used, recovery fails.
- The active log becomes full, and an offload is scheduled. Offload tries again the next time it is triggered. The active log does not wrap around; therefore, if no more active logs are available, the offload fails, but data is not lost.
- The input is needed for restart, which fails. If this is the situation that you are experiencing, see “Recovering from BSDS or log failures during restart” on page 626

Resolving the problem

Operator response: Check the allocation error code for the cause of the problem, and correct it. Ensure that drives are available, and run the recovery job again. If a DFSMSdfp ACS user-exit filter exists for an archive log data set, be careful because this can cause the DB2 subsystem to fail on a device allocation error when DB2 attempts to read the archive log data set.

Recovering from write I/O errors during archive log offload

You can recover from write I/O errors that occur during the offload of an archive log.

Symptoms

No specific DB2 message is issued for write I/O errors. Only a z/OS error recovery program message is issued.

If DB2 message DSNJ128I is issued, an abend in the offload task occurred, in which case you should follow the instructions for this message.

Environment

- Offload abandons that output data set (no entry in BSDS).
- Offload dynamically allocates a new archive and restarts offloading from the point at which it was previously triggered. For dual archiving, the second copy waits.
- If an error occurs on the new data set, these additional actions occur:
 - For dual archive mode, the following DSNJ114I message is generated, and the offload processing changes to single mode.
DSNJ114I - ERROR ON ARCHIVE DATA SET, OFFLOAD CONTINUING
WITH ONLY ONE ARCHIVE DATA SET BEING GENERATED
 - For single mode, the offload process abandons the output data set. Another attempt to offload this RBA range is made the next time offload is triggered.
 - The active log does not wrap around; if no more active logs are available, data is not lost.

Resolving the problem

Operator response: Ensure that offload activity is allocated on a drive and control unit that are operational.

Related information

"DSNJ128I" (DB2 Messages)

Recovering from read I/O errors on an archive data set during recovery

You can recover from read I/O errors that occur on an archive log during recovery.

Symptoms

No specific DB2 message is issued; only the z/OS error recovery program message is issued.

Environment

- If a second copy of the archive log exists, the second copy is allocated and used.
- If a second copy of the archive log does not exist, recovery fails.

Resolving the problem

Operator response: If you recover from tape, try recovering by using a different drive. If this approach does not work, contact the system programmer.

System programmer response: Recover to the last image copy or to the RBA of the last quiesce point.

Related reference

"RECOVER" (DB2 Utility Guide and Reference)

Recovering from insufficient disk space for offload processing

If offload processing terminates unexpectedly when DB2 is offloading the active log data sets to disk, you can recover from this situation.

Symptoms

Prior to the failure, z/OS issues abend message IEC030I, IEC031I, or IEC032I. Offload processing terminates unexpectedly. DB2 issues the following message:
DSNJ128I - LOG OFFLOAD TASK FAILED FOR ACTIVE LOG *nnnn*

Additional z/OS abend messages might accompany message DSNJ128I.

Causes

The following situations can cause problems with insufficient disk space during DB2 offload processing:

- The size of the archive log data set is too small to contain the data from the active log data sets during offload processing. All secondary space allocations have been used.
- All available space on the disk volumes to which the archive data set is being written has been exhausted.
- The primary space allocation for the archive log data set (as specified in the load module for subsystem parameters) is too large to allocate to any available online disk device.

Environment

DB2 deallocates the data set on which the error occurred. If the subsystem is in dual archive mode, DB2 changes to single archive mode and continues the offload. If the offload cannot complete in single archive mode, the active log data sets cannot be offloaded, and the status of the active log data sets remains NOTREUSEABLE. Another attempt to offload the RBA range of the active log data sets is made the next time offload is invoked.

Resolving the problem

System programmer response: The actions that you take depend on what caused DB2 message DSNJ128I to be issued:

- If z/OS abend message IEC030I precedes DB2 message DSNJ128I, increase the primary or secondary allocations (or both) for the archive log data set in DSNZPxxx. Another option is to reduce the size of the active log data set. Modifications to DSNZPxxx require that you stop and start DB2 for the changes to take effect. If the data that is to be offloaded is particularly large, you can mount another online storage volume or make one available to DB2.
- If z/OS abend message IEC032I precedes message DSNJ128I, make space available on the disk volumes, or make another online storage volume available for DB2. After you make additional space available, issue the DB2 command ARCHIVE LOG CANCEL OFFLOAD. DB2 then retries the offload.
- If z/OS abend message IEC032I precedes DB2 message DSNJ128I, make space available on the disk volumes, or make available another online storage volume for DB2. If this approach is not possible, adjust the value of PRIQTY in the DSNZPxxx module to reduce the primary allocation. If the primary allocation is reduced, you might need to increase the size of the secondary space allocation to avoid future abends.

Related reference

"2. PRIMARY QUANTITY" (DB2 Installation Guide)

Recovering from BSDS failures

When the bootstrap data set (BSDS) is damaged, you need to recover that damaged BSDS, regardless of whether you are running DB2 in dual-BSDS or single-BSDS mode.

Symptoms

If a BSDS is damaged, DB2 issues one of the following message numbers: DSNJ126I, DSNJ100I, or DSNJ120I.

Related concepts

"Management of the bootstrap data set" on page 498

Recovering from an I/O error on the BSDS

When an I/O error occurs on the only copy of the BSDS, you need to recover the BSDS before DB2 can operate normally. If an I/O error occurs on one copy of the BSDS in a dual-BSDS mode environment, you need to recover that copy of the BSDS before the next restart.

Symptoms

The following message is issued:

```
DSNJ126I - BSDS ERROR FORCED SINGLE BSDS MODE
```

The following messages are then issued:

```
DSNJ107I - READ ERROR ON BSDS
          DSNAME=... ERROR STATUS=...
DSNJ108I - WRITE ERROR ON BSDS
          DSNAME=... ERROR STATUS=...
```

Causes

A write I/O error occurred on a BSDS.

Environment

If DB2 is in a dual-BSDS mode and one copy of the BSDS is damaged by an I/O error, the BSDS mode changes from dual-BSDS mode to single-BSDS mode. If DB2 is in a single-BSDS mode when the BSDS is damaged by an I/O error, DB2 terminates until the BSDS is recovered.

Resolving the problem

System programmer response:

1. Use access method services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the failing BSDS. You can find control statements in job DSNTIJIN.
2. Issue the DB2 command RECOVER BSDS to make a copy of the good BSDS in the newly allocated data set and to reinstate dual-BSDS mode.

Related tasks

"Recovering the BSDS from a backup copy" on page 623

Recovering from an error that occurs while opening the BSDS

You need to recover the BSDS if an error occurs when DB2 opens the bootstrap data set (BSDS).

Symptoms

The following message is issued:

```
DSNJ100I - ERROR OPENING BSDS n DSNAME=..., ERROR STATUS=eeii
```

Resolving the problem

System programmer response:

1. Use access method services to delete or rename the damaged data set, to define a replacement data set, and to copy (with the REPRO command) the remaining BSDS to the replacement.
2. Use the START DB2 command to start the DB2 subsystem.

Related tasks

“Recovering the BSDS from a backup copy” on page 623

Recovering from unequal timestamps on BSDSs

When timestamps on different copies of the BSDS differ, DB2 attempts to resynchronize the BSDSs and restore dual BSDS mode. If this attempt succeeds, DB2 restart continues automatically. If this attempt fails, you need to recover from the situation.

Symptoms

The following message is issued:

```
DSNJ120I - DUAL BSDS DATA SETS HAVE UNEQUAL TIMESTAMPS,  
          BSDS1 SYSTEM=..., UTILITY=..., BSDS2 SYSTEM=..., UTILITY=...
```

Causes

Unequal timestamps can occur for the following reasons:

- One of the volumes that contains the BSDS has been restored. All information of the restored volume is outdated. If the volume contains any active log data sets or DB2 data, their contents are also outdated. The outdated volume has the lower timestamp.
- Dual BSDS mode has degraded to single BSDS mode, and you are trying to start without recovering the bad copy of the BSDS.
- The DB2 subsystem abended after updating one copy of the BSDS, but prior to updating the second copy.

Resolving the problem

Operator response: If DB2 restart fails, notify the system programmer.

System programmer response: If DB2 fails to automatically resynchronize the BSDS data sets:

1. Run the print log map utility (DSNJU004) on both copies of the BSDS; compare the lists to determine which copy is accurate or current.
2. Rename the outdated data set, and define a replacement for it.

3. Copy the good data set to the replacement data set, using the REPRO command of access method services.
4. Use the access method services REPRO command to copy the current version of the active log to the outdated data set if all the following conditions are true:
 - The problem was caused by a restored outdated BSDS volume.
 - The restored volume contains active log data.
 - You were using dual active logs on separate volumes.

If you were not using dual active logs, cold start the subsystem.

If the restored volume contains database data, use the RECOVER utility to recover that data after successful restart.

Related troubleshooting information

“Recovering from a failure during a log RBA read request” on page 647

“Recovering from a failure resulting from total or excessive loss of log data” on page 651

Related tasks

“Recovering the BSDS from a backup copy”

Recovering the BSDS from a backup copy

In some situations, the bootstrap data set (BSDS) becomes damaged, and you need to recover the BSDS from a backup copy.

DB2 stops and does not restart until dual-BSDS mode is restored in the following situations:

- DB2 is operating in single-BSDS mode, and the BSDS is damaged.
- DB2 is operating in dual-BSDS mode, and both BSDSs are damaged.

To recover the BSDS from a backup copy:

1. Locate the BSDS that is associated with the most recent archive log data set. The data set name of the most recent archive log is displayed on the z/OS console in the last occurrence of message DSNJ003I, which indicates that offloading has successfully completed. In preparation for the rest of this procedure, keep a log of all successful archives that are noted by that message.
 - If archive logs are on disk, the BSDS is allocated on any available disk. The BSDS name is like the corresponding archive log data set name; change only the first letter of the last qualifier, from A to B, as in the following example:

Archive log name
DSN.ARCHLOG1.A0000001

BSDS copy name
DSN.ARCHLOG1.B0000001
 - If archive logs are on tape, the BSDS is the first data set of the first archive log volume. The BSDS is not repeated on later volumes.
2. If the most recent archive log data set has no copy of the BSDS (presumably because an error occurred during its offload), locate an earlier copy of the BSDS from an earlier offload.
3. Rename or delete any damaged BSDS.
 - To rename a damaged BSDS, use the access method services ALTER command with the NEWNAME option.
 - To delete a damaged BSDS, use the access method services DELETE command.

For each damaged BSDS, use access method services to define a new BSDS as a replacement data set. Job DSNTIJIN contains access method services control statements to define a new BSDS. The BSDS is a VSAM key-sequenced data set (KSDS) that has three components: cluster, index, and data. You must rename all components of the data set. Avoid changing the high-level qualifier.

4. Use the access method services REPRO command to copy the BSDS from the archive log to one of the replacement BSDSs that you defined in the prior step. Do not copy any data to the second replacement BSDS; data is placed in the second replacement BSDS in a later step in this procedure.
 - a. Use the print log map utility (DSNJU004) to print the contents of the replacement BSDS. You can then review the contents of the replacement BSDS before continuing your recovery work.
 - b. Update the archive log data set inventory in the replacement BSDS.

Examine the print log map output, and note that the replacement BSDS does not obtain a record of the archive log from which the BSDS was copied. If the replacement BSDS is a particularly old copy, it is missing all archive log data sets that were created later than the BSDS backup copy. Therefore, you need to update the BSDS inventory of the archive log data sets to reflect the current subsystem inventory.

Use the change log inventory utility (DSNJU003) NEWLOG statement to update the replacement BSDS, adding a record of the archive log from which the BSDS was copied. Ensure that the CATALOG option of the NEWLOG statement is properly set to CATALOG = YES if the archive log data set is cataloged. Also, use the NEWLOG statement to add any additional archive log data sets that were created later than the BSDS copy.
 - c. Update DDF information in the replacement BSDS. If the DB2 subsystem for your installation is part of a distributed network, the BSDS contains the DDF control record. You must review the contents of this record in the output of the print log map utility. If changes are required, use the change log inventory DDF statement to update the BSDS DDF record.
 - d. Update the active log data set inventory in the replacement BSDS.

In unusual circumstances, your installation might have added, deleted, or renamed active log data sets since the BSDS was copied. In this case, the replacement BSDS does not reflect the actual number or names of the active log data sets that your installation has currently in use.

If you must delete an active log data set from the replacement BSDS log inventory, use the change log inventory utility DELETE statement.

If you need to add an active log data set to the replacement BSDS log inventory, use the change log inventory utility NEWLOG statement. Ensure that the RBA range is specified correctly on the NEWLOG statement.

If you must rename an active log data set in the replacement BSDS log inventory, use the change log inventory utility DELETE statement, followed by the NEWLOG statement. Ensure that the RBA range is specified correctly on the NEWLOG statement.
 - e. Update the active log RBA ranges in the replacement BSDS. Later, when a restart is performed, DB2 compares the RBAs of the active log data sets that are listed in the BSDS with the RBAs that are found in the actual active log data sets. If the RBAs do not agree, DB2 does not restart. The problem is magnified when a particularly old copy of the BSDS is used. To resolve this problem, use the change log inventory utility to change the RBAs that are found in the BSDS to the RBAs in the actual active log data sets. Take the appropriate action, described below, to change RBAs in the BSDS:

- If you are not certain of the RBA range of a particular active log data set, use DSN1LOGP to print the contents of the active log data set. Obtain the logical starting and ending RBA values for the active log data set from the DSN1LOGP output. The STARTRBA value that you use in the change log inventory utility must be at the beginning of a control interval. Similarly, the ENDRBA value that you use must be at the end of a control interval. To get these values, round the starting RBA value from the DSN1LOGP output down so that it ends in X'000'. Round the ending RBA value up so that it ends in X'FFF'.
- When the RBAs of all active log data sets are known, compare the actual RBA ranges with the RBA ranges that are found in the BSDS (listed in the print log map utility output).

If the RBA ranges are equal for all active log data sets, you can proceed to step 4f without any additional work.

If the RBA ranges are not equal, adjust the values in the BSDS to reflect the actual values. For each active log data set for which you need to adjust the RBA range, use the change log inventory utility DELETE statement to delete the active log data set from the inventory in the replacement BSDS. Then use the NEWLOG statement to redefine the active log data set to the BSDS.

- f. If only two active log data sets are specified in the replacement BSDS, add a new active log data set for each copy of the active log, and define each new active log data set of the replacement BSDS log inventory.

If only two active log data sets are specified for each copy of the active log, DB2 might have difficulty during restart. The difficulty can arise when one of the active log data sets is full and has not been offloaded, whereas the second active log data set is close to filling. Adding a new active log data set for each copy of the active log can alleviate difficulties on restart in this situation.

To add a new active log data set for each copy of the active log, use the access method services DEFINE command. The control statements to accomplish this task can be found in job DSNTIJIN. After the active log data sets are physically defined and allocated, use the change log inventory utility NEWLOG statement to define the new active log data sets of the replacement BSDS. You do not need to specify the RBA ranges on the NEWLOG statement.

5. Copy the updated BSDS copy to the second new BSDS data set. The dual bootstrap data sets are now identical.
6. Optional: Use the print log map utility (DSNJU004) to print the contents of the second replacement BSDS at this point.
7. If you have lost your current active log data set, refer to the following topics:
 - “Recovering from BSDS or log failures during restart” on page 626
 - “Task 4: Truncate the log at the point of error” on page 638, which provides information about how to construct a conditional restart control record (CRCR).
8. Restart DB2, using the newly constructed BSDS. DB2 determines the current RBA and what active logs need to be archived.

Related information

 z/OS Internet Library

Recovering from BSDS or log failures during restart

When the bootstrap data set (BSDS) or part of the recovery log for DB2 is damaged or lost and that damage prevents restart, you need to recover from that situation. What you do to recover varies based on the particular circumstances.

If the problem is discovered at restart, begin with one of these recovery procedures:

- “Recovering from active log failures” on page 613
- “Recovering from archive log failures” on page 618
- “Recovering from BSDS failures” on page 621

If the problem persists, return to the procedures in this section.

When DB2 recovery log damage terminates restart processing, DB2 issues messages to the console to identify the damage and issue an abend reason code. (The SVC dump title includes a more specific abend reason code to assist in problem diagnosis.) If the explanations for the reason codes indicate that restart failed because of some problem that is not related to a log error, see DB2 Diagnosis Guide and Reference, and contact IBM Software Support.

To minimize log problems during restart, the system requires two copies of the BSDS. Dual logging is also recommended.

Basic approaches to recovery: The two basic approaches to recovery from problems with the log are:

- Restart DB2, bypassing the inaccessible portion of the log and rendering some data inconsistent. Then recover the inconsistent objects by using the RECOVER utility, or re-create the data by using REPAIR. Use the methods that are described following this procedure to recover the inconsistent data.
- Restore the entire DB2 subsystem to a prior point of consistency. The method requires that you have first prepared such a point; for suggestions, see “Preparing to recover to a prior point of consistency” on page 563. Methods of recovery are described under “Recovering from unresolvable BSDS or log data set problem during restart” on page 648.

Bypassing the damaged log

Even if the log is damaged, and DB2 is started by circumventing the damaged portion, the log is the most important source for determining what work was lost and what data is inconsistent.

Bypassing a damaged portion of the log generally proceeds with the following steps:

1. DB2 restart fails. A problem exists on the log, and a message identifies the location of the error. The following abend reason codes, which appear only in the dump title, can be issued for this type of problem. This is not an exhaustive list; other codes might occur.

00D10261	00D10264	00D10267	00D1032A	00D1032C
00D10262	00D10265	00D10268	00D1032B	00E80084
00D10263	00D10266	00D10329		

The following figure illustrates the general problem.

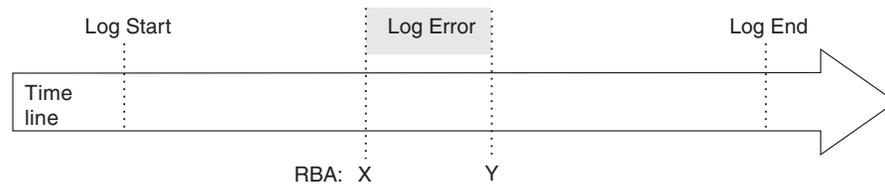


Figure 61. General problem of damaged DB2 log information

2. DB2 cannot skip over the damaged portion of the log and continue restart processing. Instead, you restrict processing to only a part of the log that is error free. For example, the damage shown in the preceding figure occurs in the log RBA range between X to Y. You can restrict restart to all of the log before X; then changes later than X are not made. Alternatively, you can restrict restart to all of the log after Y; then changes between X and Y are not made. In either case, some amount of data is inconsistent.
3. You identify the data that is made inconsistent by your restart decision. With the SUMMARY option, the DSN1LOGP utility scans the accessible portion of the log and identifies work that must be done at restart, namely, the units of recovery that are to be completed and the page sets that they modified.
Because a portion of the log is inaccessible, the summary information might not be complete. In some circumstances, your knowledge of work in progress is needed to identify potential inconsistencies.
4. You use the CHANGE LOG INVENTORY utility to identify the portion of the log to be used at restart, and to tell whether to bypass any phase of recovery. You can choose to do a cold start and bypass the entire log.
5. You restart DB2. Data that is unaffected by omitted portions of the log is available for immediate access.
6. Before you allow access to any data that is affected by the log damage, you resolve all data inconsistencies. That process is described under “Resolving inconsistencies resulting from a conditional restart” on page 655.

Where to start

The specific procedure depends on the phase of restart that was in control when the log problem was detected. On completion, each phase of restart writes a message to the console. You must find the last of those messages in the console log. The next phase after the one that is identified is the one that was in control when the log problem was detected. Accordingly, start at:

- “Recovering from failure during log initialization or current status rebuild” on page 628
- “Recovering from a failure during forward log recovery” on page 639
- “Recovering from a failure during backward log recovery” on page 645

As an alternative, determine which, if any, of the following messages was last received and follow the procedure for that message. Other DSN messages can also be issued.

Message ID	Procedure to use
DSNJ001I	“Recovering from failure during log initialization or current status rebuild” on page 628

Message ID	Procedure to use
DSNJ100I	"Recovering from unresolvable BSDS or log data set problem during restart" on page 648
DSNJ107	"Recovering from unresolvable BSDS or log data set problem during restart" on page 648
DSNJ119I	"Recovering from unresolvable BSDS or log data set problem during restart" on page 648
DSNR002I	None. Normal restart processing can be expected.
DSNR004I	"Recovering from a failure during forward log recovery" on page 639
DSNR005I	"Recovering from a failure during backward log recovery" on page 645
DSNR006I	None. Normal restart processing can be expected.
Other	"Recovering from failure during log initialization or current status rebuild"

Another procedure ("Recovering from a failure resulting from total or excessive loss of log data" on page 651) provides information to use if you determine (by using "Recovering from failure during log initialization or current status rebuild") that an excessive amount (or all) of DB2 log information (BSDS, active, and archive logs) has been lost.

The last procedure, "Resolving inconsistencies resulting from a conditional restart" on page 655, can be used to resolve inconsistencies introduced while using one of the restart procedures in this information. If you decide to use "Recovering from unresolvable BSDS or log data set problem during restart" on page 648, you do not need to use "Resolving inconsistencies resulting from a conditional restart" on page 655.

Because of the severity of the situations described, the procedures identify "Operations management action", rather than "Operator action". Operations management might not be performing all the steps in the procedures, but they must be involved in making the decisions about the steps to be performed.

Related reference

"DB2 codes" (DB2 Codes)

"Restarting a member with conditions" (DB2 Data Sharing: Planning and Administration)

"DSN1LOGP" (DB2 Utility Guide and Reference)

Recovering from failure during log initialization or current status rebuild

When a failure occurs during the log initialization or current status rebuild phase of restart, you need to recover from this situation.

Symptoms

An abend was issued, indicating that restart failed. In addition, either the last restart message that was received was a DSNJ001I message that indicates a failure during current status rebuild, or none of the following messages was issued:

- DSNJ001I
- DSNR004I
- DSNR005I

If none of the preceding messages was issued, the failure occurred during the log initialization phase of restart.

Environment

What happens in the environment depends on whether the failure occurred during log initialization or current status rebuild.

Failure during log initialization

DB2 terminates because a portion of the log is inaccessible, and DB2 cannot locate the end of the log during restart.

Failure during current status rebuild

DB2 terminates because a portion of the log is inaccessible, and DB2 cannot determine the state of the subsystem at the prior DB2 termination. Possible states include: outstanding units of recovery, outstanding database writes, and exception database conditions.

Resolving the problem

Operations management response: To correct the problem, choose one of the following approaches:

- Correct the problem that has made the log inaccessible, and start DB2 again. To determine if this approach is possible, read the relevant information about the messages and codes that you received. The explanations for the messages and codes identify the corrective action that can be taken to resolve the problem.
- Restore the DB2 log and all data to a prior consistent point, and then start DB2. This procedure is described in “Recovering from unresolvable BSDS or log data set problem during restart” on page 648.
- Start DB2 without completing some database changes. Using a combination of DB2 services and your own knowledge, determine what work is likely to be lost if you truncate the log. The procedure for determining the page sets that contain incomplete changes is described in “Restarting DB2 by truncating the log” on page 632.

Related reference

“DB2 codes” (DB2 Codes)

Related information

“DB2 messages” (DB2 Messages)

“DB2 messages” (DB2 Messages)

Description of failure during log initialization

When a failure occurs during log initialization, certain characteristics of the situation are evident.

The following figure illustrates the timeline of events that exist when a failure occurs during log initialization.

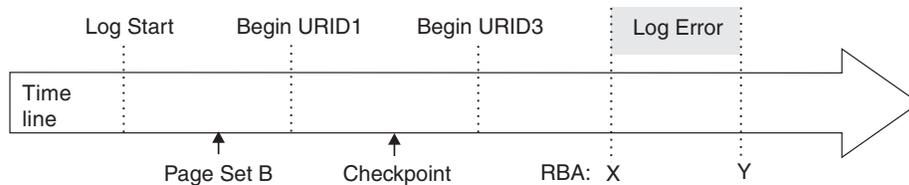


Figure 62. Failure during log initialization

The portion of the log between log RBAs X and Y is inaccessible. For failures that occur during the log initialization phase, the following activities occur:

1. DB2 allocates and opens each active log data set that is not in a stopped state.
2. DB2 reads the log until the last log record is located.
3. During this process, a problem with the log is encountered, preventing DB2 from locating the end of the log. DB2 terminates and issues an abend reason code. Some of the abend reason codes that might be issued include:
 - 00D10261
 - 00D10262
 - 00D10263
 - 00D10264
 - 00D10265
 - 00D10266
 - 00D10267
 - 00D10268
 - 00D10329
 - 00D1032A
 - 00D1032B
 - 00D1032C
 - 00E80084

During its operations, DB2 periodically records in the BSDS the RBA of the last log record that was written. This value is displayed in the print log map report as follows:

```
HIGHEST RBA WRITTEN:    00000742989E
```

Because this field is updated frequently in the BSDS, the “highest RBA written” can be interpreted as an approximation of the end of the log. The field is updated in the BSDS when any one of a variety of internal events occurs. In the absence of these internal events, the field is updated each time a complete cycle of log buffers is written. A complete cycle of log buffers occurs when the number of log buffers that are written equals the value of the OUTPUT BUFFER field of installation panel DSNTIPL. The value in the BSDS is, therefore, relatively close to the end of the log.

To find the actual end of the log at restart, DB2 reads the log forward sequentially, starting at the log RBA that approximates the end of the log and continuing until the actual end of the log is located.

Because the end of the log is inaccessible in this case, some information is lost:

- Units of recovery might have successfully committed or modified additional page sets past point X.
- Additional data might have been written, including those that are identified with writes that are pending in the accessible portion of the log.

- New units of recovery might have been created, and these might have modified data.

Because of the log error, DB2 cannot perceive these events.

A restart of DB2 in this situation requires truncation of the log. .

Related tasks

“Restarting DB2 by truncating the log” on page 632

Description of failure during current status rebuild

When a failure occurs during current status rebuild, certain characteristics of the situation are evident.

The following figure illustrates the timeline of events that exist when a failure occurs during current status rebuild.

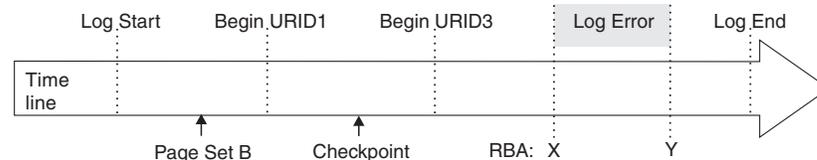


Figure 63. Failure during current status rebuild

The portion of the log between log RBAs X and Y is inaccessible. For failures that occur during the current status rebuild phase, the following activities occur:

1. Log initialization completes successfully.
2. DB2 locates the last checkpoint. (The BSDS contains a record of its location on the log.)
3. DB2 reads the log, beginning at the checkpoint and continuing to the end of the log.
4. DB2 reconstructs the status of the subsystem as it existed at the prior termination of DB2.
5. During this process, a problem with the log is encountered, preventing DB2 from reading all required log information. DB2 terminates and issues an abend reason code. Some of the abend reason codes that might be issued include:
 - 00D10261
 - 00D10262
 - 00D10263
 - 00D10264
 - 00D10265
 - 00D10266
 - 00D10267
 - 00D10268
 - 00D10329
 - 00D1032A
 - 00D1032B
 - 00D1032C
 - 00E80084

Because the end of the log is inaccessible in this case, some information is lost:

- Units of recovery might have successfully committed or modified additional page sets past point X.

- Additional data might have been written, including those that are identified with writes that are pending in the accessible portion of the log.
- New units of recovery might have been created, and these might have modified data.

Because of the log error, DB2 cannot perceive these events.

A restart of DB2 in this situation requires truncation of the log.

Related tasks

“Restarting DB2 by truncating the log”

Restarting DB2 by truncating the log

When a portion of the log is inaccessible, during the log initialization or current status rebuild phases of restart, DB2 cannot identify precisely what units of recovery failed to complete, what page sets had been modified, and what page sets have writes pending. You need to gather that information, and restart DB2.

Task 1: Find the log RBA after the inaccessible part of the log:

The first task in restarting DB2 by truncating the log is to locate the log RBA after the inaccessible part of the log.

The range of the log between RBAs X and Y is inaccessible to all DB2 processes.

To find the RBA after the inaccessible part of the log, take the action that is associated with the message number that you received (DSNJ007I, DSNJ012I, DSNJ103I, DSNJ104I, DSNJ106I, and DSNJ113I):

- **When message DSNJ007I is issued:**

The problem is that an operator canceled a request for archive mount. Reason code 00D1032B is associated with this situation and indicates that an entire data set is inaccessible.

For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The operator canceled a request for archive mount, resulting in the following message:

```
DSNJ007I OPERATOR CANCELED MOUNT OF ARCHIVE
          DSNCAT.ARCHLOG1.A0000009 VOLSER=5B225.
```

To determine the value of X, run the print log map utility (DSNJU004) to list the log inventory information. The output of this utility provides each log data set name and its associated log RBA range, the values of X and Y.

- **When message DSNJ012I is issued:**

The problem is that a log record is logically damaged. Message DSNJ012I identifies the log RBA of the first inaccessible log record that DB2 detects. The following reason codes are associated with this situation:

- 00D10261
- 00D10262
- 00D10263
- 00D10264
- 00D10265
- 00D10266
- 00D10267
- 00D10268

For example, the following message indicates a logical error in the log record at log RBA X'7429ABA'.

```
DSNJ012I ERROR D10265 READING RBA 000007429ABA
        IN DATA SET DSNCAT.LOGCOPY2.DS01
        CONNECTION-ID=DSN,
        CORRELATION-ID=DSN
```

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). DB2 stores logical records in blocks of physical records to improve efficiency. When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record are inaccessible. Therefore, the value of *X* is the log RBA that was reported in the message, rounded down to a 4-KB boundary. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ103I or DSNJ104I is issued:**

For message DSNJ103I, the underlying problem depends on the reason code that is issued:

- For reason code 00D1032B, an allocation error occurred for an archive log data set.
- For reason code 00E80084, an active log data set that is named in the BSDS could not be allocated during log initialization.

For message DSNJ104I, the underlying problem is that an open error occurred for an archive and active log data set.

In any of these cases, the message that accompanies the abend identifies an entire data set that is inaccessible. For example, the following DSNJ103I message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The STATUS field identifies the code that is associated with the reason for the data set being inaccessible.

```
DSNJ103I - csect-name LOG ALLOCATION ERROR
        DSNAME=DSNCAT.ARCHLOG1.A0000009,ERROR
        STATUS=04980004
        SMS REASON CODE=reasond-code
```

To determine the value of *X*, run the print log map utility (DSNJU004) to list the log inventory information. The output of the utility provides each log data set name and its associated log RBA range, the values of *X* and *Y*.

Verify the accuracy of the information in the print log map utility output for the active log data set with the lowest RBA range. For this active log data set only, the information in the BSDS is potentially inaccurate for the following reasons:

- When an active log data set is full, archiving is started. DB2 then selects another active log data set, usually the data set with the lowest RBA. This selection is made so that units of recovery do not need to wait for the archive operation to complete before logging can continue. However, if a data set has not been archived, nothing beyond it has been archived, and the procedure is ended.
- When logging has begun on a reusable data set, DB2 updates the BSDS with the new log RBA range for the active log data set and marks it as “Not Reusable.” The process of writing the new information to the BSDS might be delayed by other processing. Therefore, a possible outcome is for a failure to occur between the time that logging to a new active log data set begins and the time that the BSDS is updated. In this case, the BSDS information is not correct.

If the data set is marked “Not Reusable,” the log RBA that appears for the active log data set with the lowest RBA range in the print log map utility output is valid. If the data set is marked “Reusable,” you can assume for the purposes of this restart that the starting log RBA (*X*) for this data set is one greater than the highest log RBA that is listed in the BSDS for all other active log data sets.

- **When message DSNJ106I is issued:**

The problem is that an I/O error occurred while a log record was being read. The message identifies the log RBA of the first inaccessible log record that DB2 detects. Reason code 00D10329 is associated with this situation.

For example, the following message indicates an I/O error in the log at RBA X'7429ABA'.

```
DSNJ106I LOG READ ERROR DSNNAME=DSNCAT.LOGCOPY2.DS01,  
LOGRBA=000007429ABA,ERROR STATUS=0108320C
```

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record, and beyond it to the end of the log data set, are inaccessible. This is due to the log initialization or current status rebuild phase of restart. Therefore, the value of X is the log RBA that was reported in the message, rounded down to a 4-KB boundary. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ113E is issued:**

The problem is that the log RBA could not be found in the BSDS. Message DSNJ113E identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS. Reason code 00D1032B is associated with this situation.

For example, the following message indicates that the log RBA X'7429ABA' is not registered in the BSDS:

```
DSNJ113E RBA 000007429ABA NOT IN ANY ACTIVE OR ARCHIVE  
LOG DATA SET. CONNECTION-ID=DSN, CORRELATION-ID=DSN
```

Use the print log map utility (DSNJU004) to list the contents of the BSDS.

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during log initialization or current status rebuild, all log records within the physical log record are inaccessible.

Using the print log map output, locate the RBA that is closest to, but less than, X'7429ABA' for the value of X. If you do not find an RBA that is less than X'7429ABA', a considerable amount of log information has been lost. If this is the case, continue with "Recovering from a failure resulting from total or excessive loss of log data" on page 651. Otherwise, continue with the next topic.

Related concepts

"Description of failure during log initialization" on page 629

"Description of failure during current status rebuild" on page 631

Related reference

"DSNJU004 (print log map)" (DB2 Utility Guide and Reference)

Related information

"DSNJ103I" (DB2 Messages)

Task 2: Identify lost work and inconsistent data:

In certain recovery situations (such as when you recover by truncating the log), you need to identify what work was lost and what data is inconsistent.

To identify lost work and inconsistent data:

1. Obtain available information to help you determine the extent of the loss. DB2 cannot determine what units of recovery are not completed, what database state information is lost, or what data is inconsistent in this situation. The log contains all such information, but the information is not available. The steps below explain what to do to obtain the information that is available within DB2 to help you determine the extent of the loss. The steps also explain how to start DB2 in this situation.

After restart, data is inconsistent. Results of queries and any other operations on such data vary from incorrect results to abends. Abends that occur either identify an inconsistency in the data or incorrectly assume the existence of a problem in the DB2 internal algorithms.

Attention: If the inconsistent page sets are not identified and the problems in them are not resolved after starting DB2, be aware that following this procedure and allowing access to inconsistent data involves some risk.

- a. Run the print log map utility. The report that the utility produces includes a description of the last 100 checkpoints and provides, for each checkpoint the following information:
 - The location in the log of the checkpoint (begin and end RBA)
 - The date and time of day that the checkpoint was performed
- b. Locate the checkpoint on the log prior to the point of failure (X). Do that by finding the first checkpoint with an end RBA that is less than X.

Continue with the step 2 unless one of the following conditions exists:

- You cannot find such a checkpoint. This means that a considerable amount of log has been lost.
- You find the checkpoint, but the checkpoint is several days old, and DB2 has been operational during the interim.

In these two cases, use one of the following procedures:

- “Recovering from a failure resulting from total or excessive loss of log data” on page 651
- “Recovering from unresolvable BSDS or log data set problem during restart” on page 648

2. Determine what work is lost and what data is inconsistent. The portion of the log that represents activity that occurred before the failure provides information about work that was in progress at that point. From this information, you might be able to deduce what work was in progress within the inaccessible portion of the log. If use of DB2 was limited at the time or if DB2 was dedicated to a small number of activities (such as batch jobs that perform database loads or image copies), you might be able to accurately identify the page sets that were made inconsistent. To make the identification, extract a summary of the log activity up to the point of damage in the log by using the DSN1LOGP utility.

- Use the DSN1LOGP utility to specify the “BEGIN CHECKPOINT” RBA prior to the point of failure, which was determined in the previous task as the RBASTART. Terminate the DSN1LOGP scan prior to the point of failure on the log (X - 1) by using the RBAEND specification.
- Specify the last complete checkpoint. This is very important for ensuring that complete information is obtained from DSN1LOGP.
- Specify the SUMMARY(ONLY) option to produce a summary report.

The following figure is an example of a DSN1LOGP job that obtains summary information for the checkpoint that was described previously.

```

//ONE EXEC PGM=DSN1LOGP
//STEPLIB DD DSN=prefix.SDSNLOADSDSNLOAD,DISP=SHR
//SYSABEND DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSSUMRY DD SYSOUT=A
//BSDS DD DSN=DSNCAT.BSDS01,DISP=SHR
//SYSIN DD *
        RBASTART (7425468) RBAEND (7428FFF) SUMMARY (ONLY)
/*

```

Figure 64. Sample JCL for obtaining DSN1LOGP summary output for restart

3. Analyze the DSN1LOGP utility output.

Related reference

"DSN1LOGP" (DB2 Utility Guide and Reference)

DSN1LOGP summary report:

The DSN1LOGP utility generates a summary report, which is placed in the SYSSUMRY file. The report includes a summary of completed events and a restart summary. You can use the information in this report to identify lost work and inconsistent data that needs to be resolved.

The following figure shows an excerpt from the restart summary in a sample DSN1LOGP summary report. The report is described after the figure.

```

DSN1157I RESTART SUMMARY
DSN1153I DSN1LSIT CHECKPOINT
        STARTRBA=000007425468 ENDRBA=000007426C6C STARTLRSN=AA527AA809DF ENDLRSN=AA527AA829F4
        DATE=92.284 TIME=14:49:25
DSN1162I DSN1LPRT UR CONNID=BATCH CORRID=PROGRAM2 AUTHID=ADMF001 PLAN=TCEU02
        START DATE=92.284 TIME=11:12:01 DISP=INFLIGHT INFO=COMPLETE
        STARTRBA=0000063DA17B STARTLRSN=A974FAFF27FF NID=*
        LUWID=DB2NET.LUND0.A974FAFE6E77.0001 COORDINATOR=*
        PARTICIPANTS=*
        DATA MODIFIED:
                DATABASE=0101=STVDB02 PAGESET=0002=STVTS02
DSN1162I DSN1LPRT UR CONNID=BATCH CORRID=PROGRAM5 AUTHID=ADMF001 PLAN=TCEU02
        START DATE=92.284 TIME=11:21:02 DISP=INFLIGHT INFO=COMPLETE
        STARTRBA=000006A57C57 STARTLRSN=A974FAFF2801 NID=*
        LUWID=DB2NET.LUND0.A974FAFE6FFF.0003 COORDINATOR=*
        PARTICIPANTS=*
        DATA MODIFIED:
                DATABASE=0104=STVDB05 PAGESET=0002=STVTS05
DSN1162I DSN1LPRT UR CONNID=TEST0001 CORRID=CTHDCORID001 AUTHID=MULT002 PLAN=DONSQL1
        START DATE=92.278 TIME=06:49:33 DISP=INDOUBT INFO=PARTIAL
        STARTRBA=000005FBCC4F STARTLRSN=A974FBAF2302 NID=*
        LUWID=DB2NET.LUND0.B978FAFEFAB1.0000 COORDINATOR=*
        PARTICIPANTS=*
        NO DATA MODIFIED (BASED ON INCOMPLETE LOG INFORMATION)
DSN1162I UR CONNID=BATCH CORRID=PROGRAM2 AUTHID=ADMF001 PLAN=TCEU02
        START DATE=92.284 TIME=11:12:01 DISP=INFLIGHT INFO=COMPLETE
        START=0000063DA17B
DSN1160I DATABASE WRITES PENDING:
        DATABASE=0001=DSNDB01 PAGESET=004F=SYSUTIL START=000007425468
        DATABASE=0102 PAGESET=0015 START=000007425468

```

Figure 65. Partial sample of DSN1LOGP summary output

The following message acts as a heading, which is followed by messages that identify the units of recovery that have not yet completed and the page sets that they modified:

```
DSN1157I RESTART SUMMARY
```

Following the summary of outstanding units of recovery is a summary of page sets that have database writes that are pending.

In each case (units of recovery or databases with pending writes), the earliest required log record is identified by the START information. In this context, START information is the log RBA of the earliest log record that is required in order to complete outstanding writes for this page set.

Those units of recovery with a START log RBA equal to, or prior to, the point Y cannot be completed at restart. All page sets that were modified by these units of recovery are inconsistent after completion of restart when you attempt to identify lost work and inconsistent data.

All page sets that are identified in message DSN1160I with a START log RBA value equal to, or prior to, the point Y have database changes that cannot be written to disk. As in the previously described case, all of these page sets are inconsistent after completion of restart when you attempt to identify lost work and inconsistent data.

At this point, you need to identify only the page sets in preparation for restart. After restart, you need to resolve the problems in the page sets that are inconsistent.

Because the end of the log is inaccessible, some information is lost; therefore, the information is inaccurate. Some of the units of recovery that appear to be inflight might have successfully committed, or they might have modified additional page sets beyond point X. Additional data might have been written, including those page sets that are identified as having pending writes in the accessible portion of the log. New units of recovery might have been created, and these might have modified data. DB2 cannot detect that these events occurred.

From this and other information (such as system accounting information and console messages), you might be able to determine what work was actually outstanding and which page sets are likely to be inconsistent after you start DB2. This is because the record of each event contains the date and time to help you determine how recent the information is. In addition, the information is displayed in chronological sequence.

Task 3: Determine what status information is lost:

The third task in restarting DB2 by truncating the log is to determine what status information has been lost.

Depending on what was going on in your environment before the problem occurred, some amount of system status information might have been lost.

To determine what system status information is lost:

1. If you already know what system status information is lost (such as in the case in which utilities are in progress), you do not need to do anything. Continue with the next topic.
2. If you do not already know what system status information is lost, examine all relevant messages that provide details about the loss of status information (such as in the cases of deferred restart pending or write error ranges). If the messages provide adequate information about what information is lost, you do not need to do anything more. Continue with the next step.

3. If you find that all system status information is lost, try to reconstruct this information from recent console displays, messages, and abends that alerted you to these conditions. These page sets contain inconsistencies that you must resolve.

Task 4: Truncate the log at the point of error:

The fourth task in restarting DB2 by truncating the log is to truncate the log at the point of error.

No DB2 process, including the RECOVER utility, allows a gap in the log RBA sequence. You cannot process up to point X, skip over points X through Y, and continue after Y.

To truncate the log at the point of error:

Create a conditional restart control record (CRCR) in the BSDS by using the change log inventory utility. Specify the following options:

ENDRBA=*endrba*

The *endrba* value is the RBA at which DB2 begins writing new log records. If point X is X'7429000', specify ENDRBA=7429000 on the CRESTART control statement.

At restart, DB2 discards the portion of the log beyond X'7429000' before processing the log for completing work (such as units of recovery and database writes). Unless otherwise directed, DB2 performs normal restart processing within the scope of the log. Because log information is lost, DB2 errors might occur. For example, a unit of recovery that has actually been committed might be rolled back. Also, some changes that were made by that unit of recovery might not be rolled back because information about data changes is lost.

FORWARD=NO

Terminates forward-log recovery before log records are processed. This option and the BACKOUT=NO option minimize errors that might result from normal restart processing.

BACKOUT=NO

Terminates backward-log recovery before log records are processed. This option and the FORWARD=NO option minimize errors that might result from normal restart processing.

Recovering and backing out units of recovery with lost information might introduce more inconsistencies than the incomplete units of recovery.

The following example is a CRESTART control statement for the ENDRBA value of X'7429000':

```
CRESTART CREATE,ENDRBA=7429000,FORWARD=NO,BACKOUT=NO
```

Task 5: Start DB2 and resolve data inconsistencies:

The final task in restarting DB2 by truncating the log is to restart DB2 and resolve inconsistencies.

You must have a CRESTART control statement with the correct ENDRBA value and the FORWARD and BACKOUT options set to NO.

To start DB2 and resolve data inconsistencies:

1. Start DB2 with the following command:

```
-START DB2 ACCESS (MAINT)
```

In response to this command, DB2 performs the following actions:

- Discards from the checkpoint queue any entries with RBAs that are beyond the ENDRBA value in the CRCR (for example, X'7429000').
- Reconstructs the system status up to the point of log truncation.
- Performs pending database writes that the truncated log specifies and that have not already been applied to the data. You can use the DSN1LOGP utility to identify these writes. No forward recovery processing occurs for units of work in a FORWARD=NO conditional restart. All pending writes for in-commit and indoubt units of recovery are applied to the data. The standard forward-log recovery processing for the different unit of work states does not occur.
- Marks all units of recovery that have committed or are indoubt as complete on the log.
- Leaves inflight and in-abort units of recovery incomplete. Inconsistent data is left in tables that are modified by inflight or indoubt units of recovery. When you specify a BACKOUT=NO conditional restart, inflight and in-abort units of recovery are not backed out.

In a conditional restart that truncates the log, BACKOUT=NO minimizes DB2 errors for the following reasons:

- Inflight units of recovery might have been committed in the portion of the log that the conditional restart discarded. If these units of recovery are backed out (as would be normal during backward-log recovery) DB2 might back out database changes incompletely, which introduces additional errors.
- Data that is modified by in-abort units of recovery might have been modified again after the point of damage on the log. For in-abort units of recovery, DB2 might have written backout processing to disk after the point of log truncation. If these units of recovery are backed out (as would be normal during backward-log recovery), DB2 might introduce additional data inconsistencies by backing out units of recovery that are already partially or fully backed out.

At the end of restart, the conditional restart control record (CRCR) is marked "Deactivated" to prevent its use on a later restart. Until the restart completes successfully, the CRCR is in effect. Until data is consistent or page sets are stopped, start DB2 with the ACCESS (MAINT) option.

2. Resolve all data inconsistency problems.

Related concepts

"Phase 3: Forward log recovery" on page 505

"Phase 4: Backward log recovery" on page 506

Related tasks

"Resolving inconsistencies resulting from a conditional restart" on page 655

Recovering from a failure during forward log recovery

If a failure occurs during the forward-log recovery phase of restart, operations management can recover from this situation.

Symptoms

A DB2 abend occurred, indicating that restart had failed. In addition, the last restart message that was received was a DSNR004I message, which indicates that log initialization completed; therefore, the failure occurred during forward log recovery.

Environment

DB2 terminates because a portion of the log is inaccessible, and DB2 is therefore unable to guarantee the consistency of the data after restart.

Resolving the problem

Operations management response: To start DB2 successfully, choose one of the following approaches:

- Read the information about relevant messages and codes that you received to determine if this approach is possible. The explanations of the messages and codes identify any corrective action that you can take to resolve the problem. If it is possible, correct the problem that made the log inaccessible, and start DB2 again.
- Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in “Recovering from unresolvable BSDS or log data set problem during restart” on page 648.
- Start DB2 without completing some database changes. Do this only if the exact changes cannot be identified; all that can be determined is which page sets might have incomplete changes and which units of recovery made modifications to those page sets. The procedure for determining which page sets contain incomplete changes and which units of recovery made the modifications is described in “Recovering from BSDS or log failures during restart” on page 626. Other topics might help you better understand the problem.

Related reference

“DB2 codes” (DB2 Codes)

Related information

“DB2 messages” (DB2 Messages)

Forward log recovery failure

When a failure occurs during the forward-log recovery phase of DB2 restart, certain characteristics of the situation are evident.

The following figure illustrates the events surrounding a failure during the forward-log recovery phase of DB2 restart.

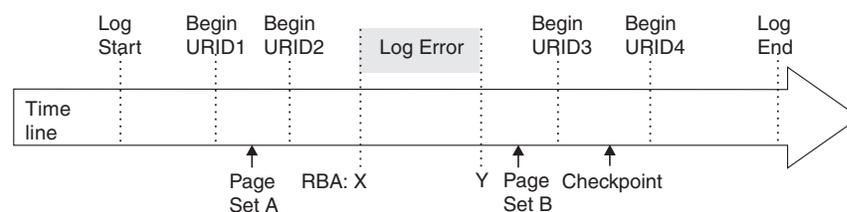


Figure 66. Illustration of failure during forward log recovery

The portion of the log between log RBA X and Y is inaccessible. The log initialization and current status rebuild phases of restart completed successfully. Restart processing was reading the log in a forward direction, beginning at some point prior to X and continuing to the end of the log. Because of the inaccessibility

of log data (between points X and Y), restart processing cannot guarantee the completion of any work that was outstanding at restart prior to point Y.

Assume that the following work was outstanding at restart:

- The unit of recovery that is identified as URID1 was in-commit.
- The unit of recovery that is identified as URID2 was inflight.
- The unit of recovery that is identified as URID3 was in-commit.
- The unit of recovery that is identified as URID4 was inflight.
- Page set A had writes that were pending prior to the error on the log, continuing to the end of the log.
- Page set B had writes that were pending after the error on the log, continuing to the end of the log.

The earliest log record for each unit of recovery is identified on the log line in Figure 66 on page 640. In order for DB2 to complete each unit of recovery, DB2 requires access to all log records from the beginning point for each unit of recovery to the end of the log.

The error on the log prevents DB2 from guaranteeing the completion of any outstanding work that began prior to point Y on the log. Consequently, database changes that are made by URID1 and URID2 might not be fully committed or backed out. Writes that were pending for page set A (from points in the log prior to Y) are lost.

Starting DB2 by limiting restart processing

When a portion of the log is inaccessible during forward recovery, starting DB2 is possible. You need to identify the units of recovery for which database changes cannot be fully guaranteed (either committed or backed out) and the page sets that these units of recovery changed.

You must determine which page sets are involved because after this procedure is used, the page sets will contain inconsistencies that you must resolve. In addition, using this procedure results in the completion of all database writes that are pending.

Related concepts

"Write operations" (DB2 Performance Monitoring and Tuning Guide)

Task 1: Find the log RBA after the inaccessible part of the log:

The first task in restarting DB2 by limiting restart processing is to locate the log RBA that is after the inaccessible part of the log.

The range of the log between RBAs X and Y is inaccessible to all DB2 processes.

To find the RBA after the inaccessible part of the log, take the action that is associated with the message number that you received (DSNJ007I, DSNJ012I, DSNJ103I, DSNJ104I, DSNJ106I, and DSNJ113I):

- **When message DSNJ007I is issued:**

The problem is that an operator canceled a request for archive mount. Reason code 00D1032B is associated with this situation and indicates that an entire data set is inaccessible.

For example, the following message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The operator canceled a request for archive mount, resulting in the following message:

```
DSNJ007I OPERATOR CANCELED MOUNT OF ARCHIVE
          DSNCAT.ARCHLOG1.A0000009 VOLSER=5B225.
```

To determine the value of X, run the print log map utility (DSNJU004) to list the log inventory information. The output of this utility provides each log data set name and its associated log RBA range, the values of X and Y.

- **When message DSNJ012I is issued:**

The problem is that a log record is logically damaged. Message DSNJ012I identifies the log RBA of the first inaccessible log record that DB2 detects. The following reason codes are associated with this situation:

- 00D10261
- 00D10262
- 00D10263
- 00D10264
- 00D10265
- 00D10266
- 00D10267
- 00D10268

For example, the following message indicates a logical error in the log record at log RBA X'7429ABA'.

```
DSNJ012I ERROR D10265 READING RBA 000007429ABA
          IN DATA SET DSNCAT.LOGCOPY2.DS01
          CONNECTION-ID=DSN,
          CORRELATION-ID=DSN
```

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). DB2 stores logical records in blocks of physical records to improve efficiency. When this type of an error on the log occurs during forward log recovery, all log records within the physical log record are inaccessible. Therefore, the value of X is the log RBA that was reported in the message, rounded down to a 4-KB boundary. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ103I or DSNJ104I is issued:**

For message DSNJ103I, the underlying problem depends on the reason code that is issued:

- For reason code 00D1032B, an allocation error occurred for an archive log data set.
- For reason code 00E80084, an active log data set that is named in the BSDS could not be allocated during log initialization.

For message DSNJ104I, the underlying problem is that an open error occurred for an archive and active log data set.

In any of these cases, the message that accompanies the abend identifies an entire data set that is inaccessible. For example, the following DSNJ103I message indicates that the archive log data set DSNCAT.ARCHLOG1.A0000009 is not accessible. The STATUS field identifies the code that is associated with the reason for the data set being inaccessible.

```
DSNJ103I - csect-name LOG ALLOCATION ERROR
          DSNAME=DSNCAT.ARCHLOG1.A0000009,ERROR
          STATUS=04980004
          SMS REASON CODE=reason-code
```

To determine the value of *X*, run the print log map utility (DSNJU004) to list the log inventory information. The output of the utility provides each log data set name and its associated log RBA range, the values of *X* and *Y*.

- **When message DSNJ106I is issued:**

The problem is that an I/O error occurred while a log record was being read. The message identifies the log RBA of the first inaccessible log record that DB2 detects. Reason code 00D10329 is associated with this situation.

For example, the following message indicates an I/O error in the log at RBA X'7429ABA'.

```
DSNJ106I LOG READ ERROR DSNNAME=DSNCAT.LOGCOPY2.DS01,  
          LOGRBA=000007429ABA,ERROR STATUS=0108320C
```

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during forward log recovery, all log records within the physical log record, and beyond it to the end of the log data set, are inaccessible to the forward log recovery phase of restart. This is due to the log initialization or current status rebuild phase of restart. Therefore, the value of *X* is the log RBA that was reported in the message, rounded down to a 4-KB boundary. (For the example message above, the rounded 4-KB boundary value would be X'7429000'.)

- **When message DSNJ113E is issued:**

The problem is that the log RBA could not be found in the BSDS. Message DSNJ113E identifies the log RBA of the inaccessible log record. This log RBA is not registered in the BSDS. Reason code 00D1032B is associated with this situation.

For example, the following message indicates that the log RBA X'7429ABA' is not registered in the BSDS:

```
DSNJ113E RBA 000007429ABA NOT IN ANY ACTIVE OR ARCHIVE  
          LOG DATA SET. CONNECTION-ID=DSN, CORRELATION-ID=DSN
```

Use the print log map utility (DSNJU004) to list the contents of the BSDS.

A given physical log record is actually a set of logical log records (the log records that are generally spoken of) and the log control interval definition (LCID). When this type of an error on the log occurs during forward log recovery, all log records within the physical log record are inaccessible.

Using the print log map output, locate the RBA that is closest to, but less than, X'7429ABA' for the value of *X*. If you do not find an RBA that is less than X'7429ABA', the value of *X* is zero. Locate the RBA that is closest to, by greater than, X'7429ABA'. This is the value of *Y*.

Related concepts

"Forward log recovery failure" on page 640

Related reference

"DSNJU004 (print log map)" (DB2 Utility Guide and Reference)

Related information

"DSNJ103I" (DB2 Messages)

Task 2: Identify incomplete units of recovery and inconsistent page sets:

The second task in restarting DB2 by limiting restart processing is to identify incomplete units of recovery and inconsistent page sets.

Units of recovery that cannot be fully processed are considered *incomplete units of recovery*. Page sets that will be inconsistent after completion of restart are considered *inconsistent page sets*.

To identify incomplete units of recovery and inconsistent page sets:

1. Determine the location of the latest checkpoint on the log by looking at one of the following sources, whichever is more convenient:
 - The operator's console contains the following message, identifying the location of the start of the last checkpoint on the log at log RBA X'876B355'. For example:

```
DSNR003I RESTART ... PRIOR CHECKPOINT
                        RBA=00007425468
```
 - The print log map utility output identifies the last checkpoint, including its BEGIN CHECKPOINT RBA.
2. Obtain a report of the outstanding work that is to be completed at the next restart of DB2 by running the DSN1LOGP utility. When you run the DSN1LOGP utility, specify the checkpoint RBA as the STARTRBA and the SUMMARY(ONLY) option. In order to obtain complete information, be sure to include the last complete checkpoint from running DSN1LOGP.
3. Analyze the output of the DSN1LOGP utility. The summary report that is placed in the SYSSUMRY file contains two sections of information: a complete summary of completed events and a restart summary.

Related concepts

"DSN1LOGP summary report" on page 636

Task 3: Restrict restart processing to the part of the log after the damage:

The third task in restarting DB2 by limiting restart processing is to restrict restart processing to the part of the log that is after the damage.

To restrict restart processing to the part of the log after the damage:

Create a conditional restart control record (CRCR) in the BSDS by using the change log inventory utility. Identify the accessible portion of the log beyond the damage by using the STARTRBA specification, which will be used at the next restart. Specify the value Y+1 (that is, if Y is X'7429FFF', specify STARTRBA=742A000). Restart restricts its processing to the portion of the log beginning with the specified STARTRBA and continuing to the end of the log. For example:

```
CRESTART CREATE,STARTRBA=742A000
```

Task 4: Start DB2 and resolve inconsistent data:

The final task in restarting DB2 by limiting restart processing is to start DB2 and resolve problems with inconsistent data.

To start DB2 and resolve data inconsistencies:

At the end of restart, the CRCR is marked DEACTIVATED to prevent its use on a subsequent restart. Until the restart is complete, the CRCR will be in effect. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.

1. Start DB2 with the following command:

```
-START DB2 ACCESS (MAINT)
```

At the end of restart, the conditional restart control record (CRCR) is marked "Deactivated" to prevent its use on a later restart. Until the restart completes successfully, the CRCR is in effect. Until data is consistent or page sets are stopped, start DB2 with the ACCESS (MAINT) option.

2. Resolve all data inconsistency problems.

Related tasks

"Resolving inconsistencies resulting from a conditional restart" on page 655

Recovering from a failure during backward log recovery

When a failure occurs during backward log recovery, DB2 terminates because it cannot access a portion of the log that it needs. Operations management can recover from this situation.

Symptoms

An abend is issued to indicate that restart failed because of a log problem. In addition, the last restart message that is received is a DSNR005I message, indicating that forward log recovery completed and that the failure occurred during backward log recovery.

Environment

Because a portion of the log is inaccessible, DB2 needs to roll back some database changes during restart.

Resolving the problem

Operations management response: To start DB2, choose one of the following approaches:

- Read the information about relevant messages and codes that you received to determine if this approach is possible. The explanations of the messages and codes identify any corrective action that you can take to resolve the problem. If it is possible, correct the problem that made the log inaccessible, and start DB2 again.
- Restore the DB2 log and all data to a prior consistent point and start DB2. This procedure is described in "Recovering from unresolvable BSDS or log data set problem during restart" on page 648.
- Start DB2 without completing some database changes. Do this only if the exact changes cannot be identified; all that can be determined is which page sets might have incomplete changes. The procedure for determining which page sets contain incomplete changes is described in "Bypassing backout before restarting" on page 646. Other related topics might help you better understand the problem.

Related reference

"DB2 codes" (DB2 Codes)

Related information

"DB2 messages" (DB2 Messages)

Backward log recovery failure

If a failure occurs during the backward-log recovery phase of restart, operations management can recover from this situation.

When a failure occurs during the backward log recovery phase, certain characteristics of the situation are evident, as the following figure shows.

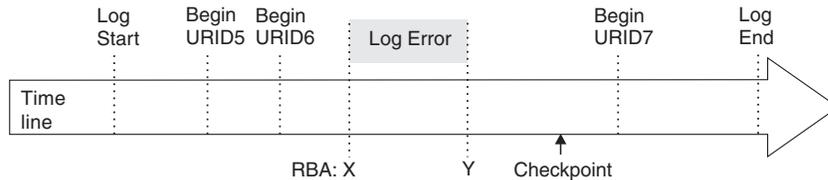


Figure 67. Illustration of failure during backward log recovery

The portion of the log between log RBA X and Y is inaccessible. The restart process was reading the log in a backward direction, beginning at the end of the log and continuing backward to the point marked by Begin URID5 in order to back out the changes that were made by URID5, URID6, and URID7. You can assume that DB2 determined that these units of recovery were in-flight or in-abort. The portion of the log from point Y to the end of the log has been processed. However, the portion of the log from Begin URID5 to point Y has not been processed and cannot be processed by restart. Consequently, database changes that were made by URID5 and URID6 might not be fully backed out. All database changes made by URID7 have been fully backed out, but these database changes might not have been written to disk. A subsequent restart of DB2 causes these changes to be written to disk during forward recovery.

Related concepts

“Recommendations for changing the BSDS log inventory” on page 499

Bypassing backout before restarting

A portion of the log becomes inaccessible when a failure occurs during backward log recovery. Operations management can recover from this situation by starting DB2 in a certain way and then identifying the page sets that are inconsistent because of the incomplete units of recovery.

To bypass backout before recovery:

1. Determine the units of recovery that cannot be backed out and the page sets that will be inconsistent after the completion of restart. To do this:
 - a. Determine the location of the latest checkpoint on the log by looking at one of the following sources, whichever is more convenient:
 - The operator’s console contains message DSNR003I, which identifies the location of the start of the last checkpoint on the log at log RBA X’7425468’.
 - b. Obtain a report of the outstanding work that is to be completed at the next DB2 restart by running the DSN1LOGP. When you run DSN1LOGP, specify the checkpoint RBA as the RBASTART and the SUMMARY(ONLY) option. Include the last complete checkpoint in the execution of DSN1LOGP in order to obtain complete information.

Analyze the output of the DSN1LOGP utility. The summary report that is placed in the SYSSUMRY file contains two sections of information. The heading of first section of the output is the following message:

```
DSN1150I SUMMARY OF COMPLETED EVENTS
```

That message is followed by other messages that identify completed events, such as completed units of recovery. That section of the output does not apply to this procedure.

The heading of the second section of the output is the following message:

```
DSN1157I RESTART SUMMARY
```

That message is followed by others that identify units of recovery that are not yet completed and the page sets that they modified. After the summary of outstanding units of recovery is a summary of page sets with database writes that are pending.

The restart processing that failed was able to complete all units of recovery processing within the accessible scope of the log after point Y. Database writes for these units of recovery are completed during the forward recovery phase of restart on the next restart. Therefore, do not bypass the forward recovery phase. All units of recovery that can be backed out have been backed out.

All remaining units of recovery that are to be backed out (DISP=INFLIGHT or DISP=IN-ABORT) are bypassed on the next restart because their STARTRBA values are less than the RBA of point Y. Therefore, all page sets that were modified by those units of recovery are inconsistent after restart. This means that some changes to data might not be backed out. At this point, you only need to identify the page sets in preparation for restart.

2. Use the change log inventory utility to create a conditional restart control record (CRCR) in the BSDS, and direct restart to bypass backward recovery processing during the subsequent restart by using the BACKOUT specification. At restart, all units of recovery that require backout are declared complete by DB2, and log records are generated to note the end of the unit of recovery. The change log inventory utility control statement is:

```
CRESTART CREATE,BACKOUT=NO
```
3. Start DB2. At the end of restart, the CRCR is marked “Deactivated” to prevent its use on a subsequent restart. Until the restart is complete, the CRCR is in effect. Use START DB2 ACCESS(MAINT) until data is consistent or page sets are stopped.
4. Resolve all inconsistent data problems. After the successful start of DB2, resolve all data inconsistency problems. “Resolving inconsistencies resulting from a conditional restart” on page 655 describes how to do this. At this time, make all other data available for use.

Related concepts

“DSN1LOGP summary report” on page 636

Recovering from a failure during a log RBA read request

A failure might occur during a log RBA read request. For example, because of problems with the BSDS, the requested RBA, which contains the dropped log data set, cannot be read. You can recover from the situation.

Symptoms

Abend code 00D1032A is issued, and message DSNJ113E is displayed:

```
DSNJ113E RBA log-rba NOT IN ANY ACTIVE OR ARCHIVE  
LOG DATA SET. CONNECTION-ID=aaaaaaaa, CORRELATION-ID=aaaaaaaa
```

Causes

The BSDS is wrapping around too frequently when log RBA read requests are submitted; when the last archive log data sets were added to the BSDS, the maximum allowable number of log data sets in the BSDS was exceeded. This caused the earliest data sets in the BSDS to be displaced by the new entry. Subsequently, the requested RBA containing the dropped log data set cannot be read after the wrap occurs.

Resolving the problem

System programmer response:

1. Stop DB2 with the STOP DB2 command, if it has not already been stopped automatically as a result of the problem.
2. Check any other messages and reason codes that are displayed, and correct the errors that are indicated. Locate the output from an old execution of the print log map utility, and identify the data set that contains the missing RBA. If the data set has not been reused, run the change log inventory utility to add this data set back into the inventory of log data sets.
3. Increase the maximum number of archive log volumes that can be recorded in the BSDS. To do this, update the MAXARCH system parameter value as follows:
 - a. Start the installation CLIST.
 - b. On panel DSNTIPA1, select UPDATE mode.
 - c. On panel DSNTIPT, change any data set names that are not correct.
 - d. On panel DSNTIPB, select the ARCHIVE LOG DATA SET PARAMETERS option.
 - e. On panel DSNTIPA, increase the value of RECORDING MAX.
 - f. When the installation CLIST editing completes, rerun job DSNTIJUZ to recompile the system parameters.
4. Start DB2 with the START DB2 command.

Related concepts

"The update process" (DB2 Installation Guide)

Related reference

"10. RECORDING MAX" (DB2 Installation Guide)

"DSNJU003 (change log inventory)" (DB2 Utility Guide and Reference)

Recovering from unresolvable BSDS or log data set problem during restart

During a restart of DB2, serious problems with the bootstrap data set (BSDS) or log data sets might occur, but operations management can recover from these problems. Use of dual logging (active logs, archive logs, and bootstrap data sets) can reduce your efforts in resolving these sorts of problems.

Symptoms

The following messages are issued:

- DSNJ100I
- DSNJ107I
- DSNJ119I

Causes

Any of the following problems might cause problems with the BSDS or log data sets during restart:

- A log data set is physically damaged.
- Both copies of a log data set are physically damaged in the case of dual logging mode.
- A log data set is lost.
- An archive log volume was reused even though it was still needed.
- A log data set contains records that are not recognized by DB2 because they are logically broken.

Environment

DB2 cannot be started until this procedure is performed.

Resolving the problem

Operations management response: Serious cases such as this sometimes necessitate a fallback to a prior shutdown level.

- If you decide to fall back (because the amount of lost information is not excessive):

1. See “Preparing to recover to a prior point of consistency” on page 563.
2. Follow the procedure in “Falling back to a prior shutdown point.”

If you use do a fallback, all database changes between the shutdown point and the present are lost. However, all the data that is retained will be consistent within DB2.

- If you decide not to fall back (because too much log information has been lost), use the alternative approach that is described in “Recovering from a failure resulting from total or excessive loss of log data” on page 651.

Falling back to a prior shutdown point

When a failure occurs in your environment, you might decide to fall back to a prior shutdown point.

When a failure occurs and you decide to fall back, use the following procedure to complete the fallback:

1. Use the print log map utility on the most current copy of the BSDS. Even if you are not able to do this, continue with the next step. (If you are unable to do this, an error message is issued.)
2. Use the access method services IMPORT command to restore the backed-up versions of the BSDS and active log data sets.
3. Use the print log map utility on the copy of the BSDS with which DB2 is to be restarted.
4. Determine whether any archive log data sets must be deleted.
 - If you have a copy of the most current BSDS, compare it to the BSDS with which DB2 is to be restarted. Delete and uncatalog any archive log data sets that are listed in the most current BSDS but are not listed in the previous one. These archive log data sets are normal physical sequential (SAM) data sets. If you are able to do this step, continue with step 5 on page 650.
 - If you were not able to print a copy of the most current BSDS and the archive logs are cataloged, use access method services LISTCAT to check for archive logs with a higher sequence number than the last archive log that is shown in the BSDS that is being used to restart DB2.

- If no archive log data sets with a higher sequence number exist, you do not need to delete or uncatalog any data sets, and you can continue with step 5.
- Delete and uncatalog all archive log data sets that have a higher sequence number than the last archive log data set in the BSDS that is being used to restart DB2. These archive log data sets are SAM data sets. Continue with the next step.

If the archive logs are not cataloged, you do not need to uncatalog them.

5. Issue the START DB2 ACCESS(MAINT) command until data is consistent or page sets are stopped.
6. Determine what data needs to be recovered, what data needs to be dropped, what data can remain unchanged, and what data needs to be recovered to the prior shutdown point.
 - For table spaces and indexes that might have been changed after the shutdown point, use the DB2 RECOVER utility to recover these table spaces and indexes. They must be recovered in the proper order.
 - For data that has not been changed after the shutdown point (data used with RO access), you do not need to use RECOVER or DROP.
 - For table spaces that were deleted after the shutdown point, issue the DROP statement. These table spaces will not be recovered.
 - Re-create any objects that were created after the shutdown point.

You must recover all data that has potentially been modified after the shutdown point. If you do not use the RECOVER utility to recover modified data, serious problems might occur because of data inconsistency.

If you try to access inconsistent data, any of the following events can occur (and the list is not comprehensive):

- You can successfully access the correct data.
 - You can access data without DB2 recognizing any problem, but it might not be the data that you want (the index might be pointing to the wrong data).
 - DB2 might recognize that a page is logically incorrect and, as a result, abend the subsystem with an X'04E' abend completion code and an abend reason code of X'00C90102'.
 - DB2 might notice that a page was updated after the shutdown point and, as a result, abend the requester with an X'04E' abend completion code and an abend reason code of X'00C200C1'.
7. Analyze the CICS log and the IMS log to determine the work that must be redone (work that was lost because of shutdown at the previous point). Inform all users (TSO users, QMF users, and batch users for whom no transaction log tracking has been performed) about the decision to fall back to a previous point.
 8. When DB2 is started after being shut down, indoubt units of recovery might exist. This occurs if transactions are indoubt when the STOP DB2 MODE (QUIESCE) command is issued. When DB2 is started again, these transactions will still be indoubt to DB2. IMS and CICS cannot know the disposition of these units of recovery.
To resolve these indoubt units of recovery, use the RECOVER INDOUBT command.
 9. If a table space was dropped and re-created after the shutdown point, drop and re-create it again after DB2 is restarted. To do this, use SQL DROP and SQL CREATE statements.

Do not use the RECOVER utility to accomplish this, because it will result in the old version (which might contain inconsistent data) that is being recovered.

10. If any table spaces and indexes were created after the shutdown point, re-create these after DB2 is restarted. You can accomplish this in these ways:
 - For data sets that are defined in DB2 storage groups, use the CREATE TABLESPACE statement, and specify the appropriate storage group names. DB2 automatically deletes the old data set and redefines a new one.
 - For user-defined data sets, use the access method services DELETE command to delete the old data sets. After these data sets have been deleted, use the access method services DEFINE command to redefine them; then use the CREATE TABLESPACE statement.

Related reference

"RECOVER" (DB2 Utility Guide and Reference)

Recovering from a failure resulting from total or excessive loss of log data

If a situation occurs that causes the entire log or an excessive amount of log data to be lost or destroyed, operations management needs to recover from that situation.

Symptoms

This situation is generally accompanied by messages or abend reason codes that indicate that an excessive amount of log information, or the entire log, has been lost.

Diagnosing the problem

In this situation, you need to rely on your own sources to determine what data is inconsistent as a result of the failure; DB2 cannot provide any hints of inconsistencies. For example, you might know that DB2 was dedicated to a few processes (such as utilities) during the DB2 session, and you might be able to identify the page sets that they modified. If you cannot identify the page sets that are inconsistent, you must decide whether you are willing to assume the risk that is involved in restarting DB2 under those conditions.

Resolving the problem

Operations management response: If you decide that a restart is needed, restart DB2 without any log data by using the appropriate procedure, depending on whether the log is totally or partially (but excessively) lost.

Recovering from a total loss of the log

If all system and user table spaces remain intact and you have a recent copy of the BSDS, you can recover from a total loss of the log. DB2 can still be restarted, and data that belongs to that DB2 subsystem can still be accessed.

All system and user table spaces must be intact, and you must have a recent copy of the BSDS. Other VSAM clusters on disk, such as the system databases DSNDB01, DSNDB04, and DSNB06, and also user databases are assumed to exist.

To restart DB2 when the entire log is lost:

1. Define and initialize the BSDSs by recovering the BSDS from a backup copy.

2. Define the active log data sets by using the access method services DEFINE command. Run utility DSNJLOGF to initialize the new active log data sets.
3. Prepare to restart DB2 with no log data. Each data and index page contains the log RBA of the last log record that was applied against the page. Safeguards within DB2 disallow a modification to a page that contains a log RBA that is higher than the current end of the log. You have two choices:
 - Determine the highest possible log RBA of the prior log. From previous console logs that were written when DB2 was operational, locate the last DSNJ001I message. When DB2 switches to a new active log data set, this message is written to the console, identifying the data set name and the highest potential log RBA that can be written for that data set. Assume that this is the value X'8BFFF'. Add one to this value (X'8C000'), and create a conditional restart control record that specifies the following change log inventory control statement:


```
CRESTART CREATE,STARTRBA=8C000,ENDRBA=8C000
```

 When DB2 starts, all phases of restart are bypassed, and logging begins at log RBA X'8C000'. If you choose this method, you do not need to use the RESET option of the DSN1COPY utility, and you can save a lot of time.
 - Run the DSN1COPY utility, specifying the RESET option to reset the log RBA in every data and index page. Depending on the amount of data in the subsystem, this process might take quite a long time. Because the BSDS has been redefined and reinitialized, logging begins at log RBA 0 when DB2 starts.

If the BSDS is not reinitialized, you can force logging to begin at log RBA 0 by constructing a conditional restart control record (CRCR) that specifies a STARTRBA and ENDRBA that are both equal to 0, as the following command shows:

```
CRESTART CREATE,STARTRBA=0,ENDRBA=0
```
4. Start DB2. Use the START DB2 ACCESS(MAINT) command until data is consistent or page sets are stopped.
5. After restart, resolve all inconsistent data as described in "Resolving inconsistencies resulting from a conditional restart" on page 655.

Related tasks

"Recovering the BSDS from a backup copy" on page 623

"Deferring restart processing" on page 510

Related reference

"DSNJLOGF (preformat active log)" (DB2 Utility Guide and Reference)

Recovering from an excessive loss of active log data

When your site experiences an excessive loss of active log data, you can develop a procedure for restarting in this situation. Do not redefine the BSDS.

You can recover from an excessive loss of active log data in one of two ways.

Recovering DB2 by creating a gap in the active log:

If your site experiences an excessive loss of active log data, you can recover by creating a gap in the active log.

To recover by creating a gap in the active log:

1. Use the print log map utility (DSNJU004) on the copy of the BSDS with which DB2 is to be restarted.

2. Use the print log map output to obtain the data set names of all active log data sets. Use the access method services LISTCAT command to determine which active log data sets are no longer available or usable.
3. Use the access method services DELETE command to delete all active log data sets that are no longer usable.
4. Use the access method services DEFINE command to define new active log data sets. Run the DSNJLOGF utility to initialize the new active log data sets. Define one active log data set for each one that is found to be no longer available or usable in step 2. Use the active log data set name that is found in the BSDS as the data set name for the access method services DEFINE command.
5. Refer to the print log map utility (DSNJU004) output, and note whether an archive log data set exists that contains the RBA range of the redefined active log data set. To do this, note the starting and ending RBA values for the active log data set that was recently redefined, and look for an archive log data set with the same starting and ending RBA values.

If no such archive log data sets exist:

- a. Use the change log inventory utility (DSNJU003) DELETE statement to delete the recently redefined active log data sets from the BSDS active log data set inventory.
- b. Use the change log inventory utility (DSNJU003) NEWLOG statement to add the active log data set to the BSDS active log data set inventory. Do not specify RBA ranges on the NEWLOG statement.

If the corresponding archive log data sets exist, two courses of action are available:

- If you want to minimize the number of potential read operations on the archive log data sets, use the access method services REPRO command to copy the data from each archive log data set into the corresponding active log data set. Ensure that you copy the proper RBA range into the active log data set.

Ensure that the active log data set is large enough to hold all the data from the archive log data set. When DB2 does an archive operation, it copies the log data from the active log data set to the archive log data set, and then pads the archive log data set with binary zeroes to fill a block. In order for the access method services REPRO command to be able to copy all of the data from the archive log data set to a recently defined active log data set, the new active log data set might need to be larger than the original one.

For example, if the block size of the archive log data set is 28 KB, and the active log data set contains 80 KB of data, DB2 copies the 80 KB and pads the archive log data set with 4 KB of nulls to fill the last block. Thus, the archive log data set now contains 84 KB of data instead of 80 KB. In order for the access method services REPRO command to complete successfully, the active log data set must be able to hold 84 KB, rather than just 80 KB of data.

- If you are not concerned about read operations against the archive log data sets, complete the two steps that appear in the steps 5a and 5b (as though the archive data sets did not exist).
6. Choose the appropriate point for DB2 to start logging. To do this, determine the highest possible log RBA of the prior log. From previous console logs that were written when DB2 was operational, locate the last DSNJ001I message. When DB2 switches to a new active log data set, this message is written to the console, identifying the data set name and the highest potential log RBA that can be written for that data set. Assume that this is the value X'8BFFF'. Add

one to this value (X'8C000'), and create a conditional restart control record that specifies the following change log inventory control statement:

```
CRESTART CREATE,STARTRBA=8C000,ENDRBA=8C000
```

When DB2 starts, all phases of restart are bypassed, and logging begins at log RBA X'8C000'. If you choose this method, you do not need to use the RESET option of the DSN1COPY utility, and you can save a lot of time.

7. To restart DB2 without using any log data, create a conditional restart control record for the change log inventory utility (DSNJU003).
8. Start DB2. Use the START DB2 ACCESS(MAINT) command until data is consistent or page sets are stopped.
9. After restart, resolve all inconsistent data as described in "Resolving inconsistencies resulting from a conditional restart" on page 655.

This procedure causes all phases of restart to be bypassed and logging to begin at the point in the log RBA that you identified in step 6 on page 653 (X'8C000' in the example given in this procedure). This procedure creates a gap in the log between the highest RBA kept in the BSDS and, in this example, X'8C000', and that portion of the log is inaccessible.

Because no DB2 process can tolerate a gap, including RECOVER, you need to take image copies of all data after a cold start, even data that you know is consistent.

Related reference

"DSNJU003 (change log inventory)" (DB2 Utility Guide and Reference)

Recovering DB2 without creating a gap in the active log:

You can do a cold start without creating a gap in the log. Although this approach does eliminate the gap in the physical log record, you cannot use a cold start to resolve the logical inconsistencies.

To recover without creating a gap in the active log:

1. Locate the last valid log record by using the DSN1LOGP utility to scan the log. Message DSN1213I identifies the last valid log RBA.
2. Identify the last RBA that is known to be valid by examining message DSN1213I. For example, if message DSN1213I indicates that the last valid log RBA is X'89158', round this value up to the next 4-KB boundary, which in this example is X'8A000'.
3. Create a conditional restart control record (CRCR). For example:

```
CRESTART CREATE,STARTRBA=8A000,ENDRBA=8A000
```
4. Start DB2 with the START DB2 ACCESS(MAINT) command until data is consistent or page sets are stopped.
5. Take image copies of all data for which data modifications were recorded beyond the log RBA that was used in the CRESTART statement (in this example, X'8A000'). If you do not know what data was modified, take image copies of all data.

If you do not take image copies of data that has been modified beyond the log RBA that was used in the CRESTART statement, future RECOVER utility operations might fail or result in inconsistent data.

After restart, resolve all inconsistent data as described in "Resolving inconsistencies resulting from a conditional restart" on page 655.

Resolving inconsistencies resulting from a conditional restart

When a conditional restart of the DB2 subsystem is done, several problems might occur. Recovery from these problems is possible and varies based on the specific situation.

The following problems might occur after a conditional restart of DB2:

- Some amount of work is left incomplete.
- Some data is left inconsistent.
- Information about the status of objects within the DB2 subsystem is made unusable.

Inconsistencies in a distributed environment

In a distributed environment, when a DB2 subsystem restarts, it indicates its restart status and the name of its recovery log to the systems with which it communicates. The two possible conditions for restart status are warm and cold.

A cold status is associated with a *cold start*, which is a process by which a DB2 subsystem restarts without processing any log records. DB2 has no memory of previous connections with its partner. The general process that occurs with a cold start in a distributed environment is as follows:

1. The partner (for example CICS) accepts the cold start connection and remembers the recovery log name of the DB2 subsystem that experienced the cold start.
2. If the partner has indoubt thread resolution requirements with the cold-starting DB2 subsystem, those requirements cannot be satisfied.
3. The partner terminates its indoubt resolution responsibility with the cold-starting DB2 subsystem. However, as a participant, the partner has indoubt logical units of work that must be resolved manually.
4. Because the DB2 subsystem has an incomplete record of resolution responsibilities, DB2 attempts to reconstruct as much resynchronization information as possible.
5. DB2 displays the information that it was able to reconstruct in one or more DSNL438 or DSNL439 messages.
6. DB2 then discards the synchronization information that it was able to reconstruct and removes any restrictive states that are maintained on the object.

Resolving inconsistencies

In some problem situations, you need to determine what you must do in order to resolve any data inconsistencies that exist.

To resolve inconsistencies:

1. Determine the scope of any inconsistencies that are introduced by the situation.
 - a. If the situation is either a cold start that is beyond the current end of the log or a conditional restart that skips backout or forward log recovery, use the DSN1LOGP utility to determine what units of work have not been backed out and which objects are involved. For a cold start that is beyond the end of the log, you can also use DSN1LOGP to help identify any restrictive object states that have been lost.
 - b. If a conditional restart truncates the log in a non-data sharing environment, recover all data and indexes to the new current point in time, and rebuild the data and indexes as needed. You need to recover or rebuild (or both recover and rebuild) the data and indexes because data and index updates

might exist without being reflected in the DB2 log. When this situation occurs, a variety of inconsistency errors might occur, including DB2 abends with reason code 00C200C1.

2. Decide what approach to take to resolve inconsistencies by reading related topics about the approaches:
 - Recovery to a prior point of consistency
 - Restoration of a table space
 - Use of the REPAIR utility on the data

The first two approaches are less complex than, and therefore preferred over, the third approach.

3. If one or more of the following conditions are applicable, take image copies of all DB2 table spaces:
 - You did a cold start.
 - You did a conditional restart that altered or truncated the log.
 - The log is damaged.
 - Part of the log is no longer accessible.

When a portion of the DB2 recovery log becomes inaccessible, all DB2 recovery processes have difficulty operating successfully, including restart, RECOVER, and deferred restart processing. Conditional restart allows circumvention of the problem during the restart process. To ensure that RECOVER does not attempt to access the inaccessible portions of the log, secure a copy (either full or incremental) that does not require such access. A failure occurs any time a DB2 process (such as the RECOVER utility) attempts to access an inaccessible portion of the log. You cannot be sure which DB2 processes must use that portion of the recovery log. Therefore, you need to assume that all data recovery activity requires that portion of the log.

A cold start might cause down-level page set errors, which you can find out about in different ways:

- Message DSNB232I is sometimes displayed during DB2 restart, once for each down-level page set that DB2 detects. After you restart DB2, check the console log for down-level page set messages.
 - If a small number of those messages exist, run DSN1COPY with the RESET option to correct the errors to the data before you take image copies of the affected data sets.
 - If a large number of those messages exist, the actual problem is not that page sets are down-level but that the conditional restart inadvertently caused a high volume of DSNB232I messages. In this case, temporarily turn off down-level detection by turning off the DLDFREQ ZPARM. (See 14. LEVELID UPDATE FREQ.)

In either case, continue with step 4 on page 657.

- If you run the COPY utility with the SHRLEVEL REFERENCE option to make image copies, the COPY utility sometimes issues message DSNB232I about down-level page sets that DB2 does not detect during restart. If any of those messages were issued when you are making image copies, correct the errors, and continue making image copies of the affected data sets.
- If you use some other method to make image copies, you will find out about down-level page set errors during normal operation. In this case, you need to correct the errors by using the information in “Recovering from a down-level page set problem” on page 662.

4. For any DB2 (catalog and directory) system table spaces that are inconsistent, recover them in the proper order. You might need to recover to a prior point in time, prior to the conditional restart.
5. For any objects that you suspect might be inconsistent, resolve the database inconsistencies before proceeding.
 - First, resolve inconsistencies in DB2 system databases DSNDB01 and DSNDB06. Catalog and directory inconsistencies need to be resolved before inconsistencies in other databases because the subsystem databases describe all other databases, and access to other databases requires information from DSNDB01 and DSNDB06.
 - If you determine that the existing inconsistencies involve indexes only (not data), use the REBUILD INDEX utility to rebuild the affected indexes. Alternatively, you can use the RECOVER utility to recover the index if rebuilding the indexes is not possible.
 - For a table space that cannot be recovered (and is thus inconsistent), determine the importance of the data and whether it can be reloaded. If the data is not critical or can be reloaded, drop the table after you restart DB2, and reload the data rather than trying to resolve the inconsistencies.

Related concepts

“Recovery of data to a prior point of consistency” on page 556

Related tasks

“Recovering catalog and directory objects” (DB2 Utility Guide and Reference)

Related reference

“RECOVER” (DB2 Utility Guide and Reference)

“DSN1COPY” (DB2 Utility Guide and Reference)

Restoring the table space:

You can restore the table space by reloading data into it or by re-creating the table space, which requires advance planning. Either of these methods is easier than using REPAIR.

Reloading the table space is the preferred approach, when it is possible, because reloading is easier and requires less advance planning than re-creating a table space. Re-creating a table space involves dropping and then re-creating the table space and associated tables, indexes, authorities, and views, which are implicitly dropped when the table space is dropped. Therefore, re-creating the objects means that you need to plan ahead so that you will be prepared to re-establish indexes, views, authorizations, and the data content itself.

Restriction: You cannot drop DB2 system tables, such as the catalog and directory. For these system tables, follow one of these procedures instead of this one:

- “Recovery of data to a prior point of consistency” on page 556
- “Using the REPAIR utility on inconsistent data” on page 658

To restore the table space:

1. Decide whether you can reload the table space or must drop and re-create it.
 - If you can reload the table space, run the appropriate LOAD utility jobs to do so; specify the REPLACE option. After you load the content of the table space, skip to step 6 on page 658.
 - If you cannot reload the table space, continue with step 2 on page 658.

2. Issue an SQL DROP TABLESPACE statement for the table space that is suspected of being involved in the problem.
3. Re-create the table space, tables, indexes, synonyms, and views by using SQL CREATE statements.
4. Grant access to these objects the same way that access was granted prior to the time of the error.
5. Reconstruct the data in the tables.
6. Run the RUNSTATS utility on the data.
7. Use the COPY utility to acquire a full image copy of all data.
8. Use the REBIND command on all plans that use the tables or views that are involved in this activity.

Related concepts

“Recovery of data to a prior point of consistency” on page 556

Related tasks

“Using the REPAIR utility on inconsistent data”

Related reference

“LOAD” (DB2 Utility Guide and Reference)

Using the REPAIR utility on inconsistent data:

You can resolve inconsistencies with the REPAIR utility. However, using REPAIR is not recommended unless the inconsistency is limited to a small number of data or index pages.

DB2 does not provide a mechanism to automatically inform users about data that is physically inconsistent or damaged. When you use SQL to access data that is physically damaged, DB2 issues messages to indicate that data is not available due to a physical inconsistency.

However, DB2 includes several utilities that can help you identify data that is physically inconsistent before you try to access it. These utilities are:

- CHECK DATA
- CHECK INDEX
- CHECK LOB
- COPY with the CHECKPAGE option
- DSN1COPY with the CHECK option

Attention: If you decide to use this method to resolve data inconsistencies, use extreme care. Use of the REPAIR utility to correct inconsistencies requires in-depth knowledge of DB2 data structures. Incorrect use of the REPAIR utility can cause further corruption and loss of data. Read this topic carefully because it contains information that is important to the successful resolution of the inconsistencies.

Recommendation: Avoid using this procedure if you are experiencing extensive data inconsistency because it is more time-consuming and complex (and therefore prone to error) than recovering to a point in time or re-creating the table spaces. If possible, use those alternative procedures instead.

Restrictions:

- Although the DSN1LOGP utility can identify page sets that contain inconsistencies, this utility cannot identify the specific data modifications that are involved in the inconsistencies within a given page set.

- Any pages that are on the logical page list (perhaps caused by this restart) cannot be accessed by using the REPAIR utility.

To use the REPAIR utility to resolve the inconsistency:

1. Issue the following command to start DB2 and allow access to data:

```
START DATABASE (dbase) SPACENAM (space) ACCESS(FORCE)
```

In this command, *space* identifies the table space that is involved.

2. If any system data is inconsistent, use the REPAIR utility to resolve those inconsistencies. DB2 system data (such as data that is in the catalog and directory) exists in interrelated tables and table spaces. Data in DB2 system databases cannot be modified with SQL, so use of the REPAIR utility is necessary to resolve the inconsistencies that are identified.
3. Determine if you have any structural violations in data pages. DB2 stores data in data pages. The structure of data in a data page must conform to a set of rules for DB2 to be able to process the data accurately. Using a conditional restart process does not cause violations to this set of rules; but, if violations existed prior to conditional restart, they continue to exist after conditional restart.
4. Use the DSN1COPY utility with the CHECK option to identify any violations that you detected in the previous step, and then resolve the problems, possibly by recovering or rebuilding the object or by dropping and re-creating it.
5. Examine the various types of pointers that DB2 uses to access data (indexes, hashes, and links), and identify inconsistencies that need to be manually corrected.

Hash and link pointers exist in the DB2 directory database; link pointers also exist in the catalog database. DB2 uses these pointers to access data. During a conditional restart, data pages might be modified without update of the corresponding pointers. When this occurs, one of the following actions might occur:

- If a pointer addresses data that is nonexistent or incorrect, DB2 abends the request. If SQL is used to access the data, a message that identifies the condition, and the page in question is issued.
- If data exists but no pointer addresses it, that data is virtually invisible to all functions that attempt to access it by using the damaged hash or link pointer. The data might, however, be visible and accessible by some functions, such as SQL functions that use another pointer that was not damaged. This situation can result in inconsistencies.

If a row that contains a varying-length field is updated, it can increase in size. If the page in which the row is stored does not contain enough available space to store the additional data, the row is placed in another data page, and a pointer to the new data page is stored in the original data page. After a conditional restart, one of the following conditions might exist.

- The row of data exists, but the pointer to that row does not exist. In this case, the row is invisible, and the data cannot be accessed.
 - The pointer to the row exists, but the row itself no longer exists. DB2 abends the requester when any operation (for instance, a SELECT) attempts to access the data. If termination occurs, one or more messages are issued to identify the condition and the page that contains the pointer.
6. Use the REPAIR utility to resolve any inconsistencies that you detected in the previous step.
 7. Reset the log RBA in every data and index page set that are to be corrected with this procedure by using the DSN1COPY RESET option. This step is

necessary for the following reason. If the log was truncated, changing data by using the REPAIR utility can cause problems. Each data and index page contains the log RBA of the last recovery log record that was applied against the page. DB2 does not allow modification of a page that contains a log RBA that is higher than the current end of the log.

8. When all known inconsistencies have been resolved, take full image copies of all modified table spaces, in order to use the RECOVER utility to recover from any future problems. This last step is imperative.

Related concepts

“Recovery of data to a prior point of consistency” on page 556

Related tasks

“Restoring the table space” on page 657

Related reference

“REPAIR” (DB2 Utility Guide and Reference)

Related information

DB2 Diagnosis Guide and Reference

Recovering from DB2 database failure

If a DB2 failure occurs because of an allocation or open problem, you can recover from this situation.

Symptoms

The symptoms vary based on whether the failure was an allocation or an open problem:

Allocation problem

The following message indicates an allocation problem:

```
DSNB207I - DYNAMIC ALLOCATION OF DATA SET FAILED.  
          REASON=rrrr DSNAME=dsn
```

The *rrrr* is a z/OS dynamic allocation reason code.

Open problem

The following messages indicate an open problem:

```
IEC161I rc[(sfi)] - ccc, iii, sss, ddn,  
          ddd, ser, xxx, dsn, cat
```

```
DSNB204I - OPEN OF DATA SET FAILED. DSNAME = dsn
```

In the IEC161I message:

<i>rc</i>	Is a return code.
<i>sfi</i>	Is subfunction information, which is displayed only with certain return codes.
<i>ccc</i>	Is a function code.
<i>iii</i>	Is a job name.
<i>sss</i>	Is a step name.
<i>ddn</i>	Is a DD name.
<i>ddd</i>	Is a device number (if the error is related to a specific device).
<i>ser</i>	Is a volume serial number (if the error is related to a specific volume).
<i>xxx</i>	Is a VSAM cluster name.
<i>dsn</i>	Is a data set name.
<i>cat</i>	Is a catalog name.

Environment

When this type of problem occurs:

- The table space is automatically stopped.
- Programs receive a -904 SQLCODE (SQLSTATE '57011').
- If the problem occurs during restart, the table space is marked for deferred restart, and restart continues. The changes are applied later when the table space is started.

Resolving the problem

Operator response:

1. Check reason code, and correct problems.
2. Ensure that drives are available for allocation.
3. Enter the command START DATABASE.

Recovering a DB2 subsystem to a prior point in time

You can recover a DB2 subsystem and data sharing group to a prior point in time by using the BACKUP SYSTEM and RESTORE SYSTEM utilities.

In this recovery procedure, you create and populate a table that contains data that is both valid and invalid. You need to restore your DB2 subsystem to a point in time before the invalid data was inserted into the table, but after the point in time when the valid data was inserted. Also, you create an additional table space and table that DB2 will re-create during the log-apply phase of the restore process.

To insert data into a table, determine the point in time that you want to recover to, and then recover the DB2 subsystem to a prior point in time:

1. Issue the START DB2 command to start DB2 and all quiesced members of the data sharing group. Quiesced members are ones that you removed from the data sharing group either temporarily or permanently. Quiesced members remain dormant until you restart them.
2. Issue SQL statements to create a database, a table space, and two tables with one index for each table.
3. Issue the BACKUP SYSTEM DATA ONLY utility control statement to create a backup copy of only the database copy pool for a DB2 subsystem or data sharing group.
4. Issue an SQL statement to first insert rows into one of the tables, and then update some of the rows.
5. Use the LOAD utility with the LOG NO attribute to load the second table.
6. Issue SQL statements to create an additional table space, table, and index in an existing database. DB2 will re-create the additional table space and table during the log-apply phase of the restore process.
7. Issue the SET LOG SUSPEND command or the SET LOG RESUME command to obtain a log truncation point, *logpoint1*, which is the point you want to recover to. For a non-data sharing group, use the RBA value. For a data sharing group, use the lowest log record sequence number (LRSN) from the active members.

The following example shows sample output for the SET LOG SUSPEND command:

```

09.47.52          @set log suspend
09.47.52 STC00066 DSN9022I @ DSNJC001 '-SET LOG' NORMAL COMPLETION
*09.47.55 STC00066 *DSNJ372I @ DSNJC09A UPDATE ACTIVITY HAS BEEN
*SUSPENDED FOR VA1A AT RBA 00004777B710, LRSN C31E5141E0C4, PRIOR
*CHECKPOINT RBA 000047778090

```

8. Issue an SQL statement to first insert rows into one of the tables and then to update and delete some rows.
9. Issue the STOP DB2 command to stop DB2 and all active members of the data sharing group.
10. Run the DSNJU003 change log inventory utility to create a SYSPITR CRCR record (CRESTART CREATE SYSPITR=*logpoint1*). The log truncation point is the value that you obtained from issuing either the SET LOG SUSPEND command, or the SET LOG RESUME command.
11. For a data sharing group, delete all of the coupling facility structures.
12. Issue the START DB2 command to restart DB2 and all members of the data sharing group.
13. Run the RESTORE SYSTEM utility. For a data sharing group, this utility can be run only on one member. If the utility stops and you must restart it, you can restart the utility only on the member on which it was initially run.
14. After the RESTORE SYSTEM utility completes successfully, issue the STOP DB2 command to stop DB2 and all active members of the data sharing group. The DB2 subsystem resets to RECOVER-pending status.
15. Issue the START DB2 command to restart DB2 and all members of the data sharing group.
16. Issue the DISPLAY command to identify the utilities that are active and the objects that are restricted. For example:

```

-DIS UTIL(*)
-DIS DB(DSNDB01) SP(*)
-DIS DB(DSNDB06) SP(*) LIMIT(*)
-DIS DB(DSNDB06) SP(*) LIMIT(*)RESTRICT

```
17. Stop all of the active utilities that you identified in the previous step.
18. Recover any objects that are in RECOVER-pending status or REBUILD-pending status from the table that you created in step 6 on page 661.

Recovering from a down-level page set problem

When using a stand-alone or non-DB2 utility, you might inadvertently replace a DB2 data set with an incorrect or outdated copy. Such a copy is called *down-level*. Using a down-level page set can cause complex problems; therefore, you need to recover from this situation.

Symptoms

The following message is issued:

```
DSNB232I csect-name - UNEXPECTED DATA SET LEVEL ID ENCOUNTERED
```

The message also contains the level ID of the data set, the level ID that DB2 expects, and the name of the data set.

Causes

A down-level page set can be caused by:

- A DB2 data set is inadvertently replaced by an incorrect or outdated copy. Usually this happens in conjunction with use of a stand-alone or non-DB2 utility, such as DSN1COPY or DFSMSHsm.
- A cold start of DB2 occurs.
- A VSAM high-used RBA of a table space becomes corrupted.

DB2 associates a level ID with every page set or partition. Most operations detect a down-level ID, and return an error condition, when the page set or partition is first opened for mainline or restart processing. The exceptions are the following operations, which do not use the level ID data:

- LOAD REPLACE
- RECOVER
- REBUILD INDEX
- DSN1COPY
- DSN1PRNT

Environment

- If the error was reported during mainline processing, DB2 sends a "resource unavailable" SQLCODE and a reason code to the application to explain the error.
- If the error was detected while a utility was processing, the utility generates a return code 8.

Diagnosing the problem

Determine whether the message was issued during restart or at some other time during normal operation. This information is important for determining what steps to take below

Resolving the problem

System programmer response: The actions that you need to do to recover depend on when the message was issued:

- If the message was issued during restart, take one of the following actions:
 - Replace the data set with one that is at the proper level, by using DSN1COPY, DFSMSHsm, or some equivalent method. To check the level ID of the new data set, run the stand-alone utility DSN1PRNT on it, with the options PRINT(0) (to print only the header page) and FORMAT. The formatted print output identifies the level ID.
 - Recover the data set to the current time, or to a prior time, by using the RECOVER utility.
 - Replace the contents of the data set, by using LOAD REPLACE.
- If the message was issued during normal operation (not during restart):
 1. Take one of the actions that are listed for situations when the message was issued during restart.
 2. Accept the down-level data set by changing its level ID. You can use the REPAIR utility with the LEVELID statement. (You cannot use the LEVELID option in the same job step with any other REPAIR utility control statement.)

Attention: If you accept a down-level data set or disable down-level detection, your data might be inconsistent.

Related system programmer actions: Consider taking the following actions, which might help you minimize or deal with down-level page set problems in the future:

- To control how often the level ID of a page set or partition is updated, specify a value between 0 and 32767 on the LEVELID UPDATE FREQ field of panel DSNTIPL.
- To disable down-level detection, specify 0 in the LEVELID UPDATE FREQ field of panel DSNTIPL.
- To control how often level ID updates are taken, specify a value between 1 and 32767.

Related reference

"DSN1COPY" (DB2 Utility Guide and Reference)

"DSN1PRNT" (DB2 Utility Guide and Reference)

"LOAD" (DB2 Utility Guide and Reference)

"RECOVER" (DB2 Utility Guide and Reference)

"REPAIR" (DB2 Utility Guide and Reference)

"14. LEVELID UPDATE FREQ" (DB2 Installation Guide)

Recovering from a problem with invalid LOBs

If a LOB table space is defined with LOG NO and you need to recover that table space, you can recover the LOB data to the point that you made your last image copy of the table space.

Unless your LOBs are fairly small, specifying LOG NO for LOB objects is recommended for the best performance. However, to maximize recoverability, specifying LOG YES is recommended. The performance cost of logging exceeds the benefits that you can receive from logging such large amounts of data. If no changes are made to LOB data, the logging cost is not an issue. However, you should make image copies of the LOB table space to prepare for failures. The frequency with which you make image copies is based on how often you update LOB data.

To recover LOB data from a LOB table space that is defined with LOG NO:

1. Run the RECOVER utility as you do for other table spaces:

```
RECOVER TABLESPACE dbname.lobts
```

If changes were made after the image copy, DB2 puts the table space in auxiliary warning status, which indicates that some of your LOBs are invalid. Applications that try to retrieve the values of those LOBs receive SQLCODE -904. Applications can still access other LOBs in the LOB table space.

2. Get a report of the invalid LOBs by running CHECK LOB on the LOB table space:

```
CHECK LOB TABLESPACE dbname.lobts
```

DB2 generates the following messages:

```
LOB WITH ROWID = 'xxxxxxx' VERSION = n IS INVALID
```

3.  Fix the invalid LOBs, by updating the LOBs or setting them to the null value. For example, suppose that you determine from the CHECK LOB utility that the row of the EMP_PHOTO_RESUME table with ROWID X'C1BDC4652940D40A81C201AA0A28' has an invalid value for column RESUME. If host variable *hlob* contains the correct value for RESUME, you can use this statement to correct the value:

```
UPDATE DSN8910. EMP_PHOTO_RESUME
SET RESUME = :hvlob
WHERE EMP_ROWID = ROWID(X'C1BDC4652940D40A81C201AA0A28');
```

GUI

Recovering from table space input-output errors

You can recover a table space after I/O errors have occurred and caused the table space to fail.

Symptoms

The following message is issued, where *ddddddd* is a table space name:

```
DSNU086I  DSNUCDA1 READ I/O ERRORS ON SPACE=ddddddd.
          DATA SET NUMBER=nnn.
          I/O ERROR PAGE RANGE=aaaaaa, bbbbbb.
```

Any table spaces that are identified in DSNU086I messages must be recovered. Follow the steps later in this topic.

Environment

DB2 remains active.

Resolving the problem

Operator response:

1. Fix the error range:
 - a. Use the command STOP DATABASE to stop the failing table space.
 - b. Use the command START DATABASE ACCESS (UT) to start the table space for utility-only access.
 - c. Start a RECOVER utility step to recover the error range by using the following statement:

```
DB2 RECOVER (ddddddd) ERROR RANGE
```

If you receive message DSNU086I again to indicate that the error range recovery cannot be performed, continue with step 2.
 - d. Issue the command START DATABASE to start the table space for RO or RW access, whichever is appropriate. If the table space is recovered, you do not need to continue with the following procedure.
2. If error range recovery fails because of a hardware problem:
 - a. Use the command STOP DATABASE to stop the table space or table space partition that contains the error range. As a result of this command, all in-storage data buffers that are associated with the data set are externalized to ensure data consistency during the subsequent steps.
 - b. Use the INSPECT function of the IBM Device Support Facility, ICKDSF, to check for track defects and to assign alternate tracks as necessary. Determine the physical location of the defects by analyzing the output of messages DSNB224I, DSNU086I, and IOS000I. These messages are displayed on the system operator's console at the time that the error range was created. If damaged storage media is suspected, request assistance from IBM Hardware Support before proceeding.

- c. Use the command START DATABASE to start the table space with ACCESS(UT) or ACCESS(RW).
- d. Run the RECOVER utility with the ERROR RANGE option. Specify an error range that, from image copies, locates, allocates, and applies the pages within the tracks that are affected by the error ranges.

Related information

 z/OS Internet Library

Recovering from DB2 catalog or directory I/O errors

When the DB2 catalog or directory fails because of I/O errors, you need to recover from this situation so that processing can return to normal.

Symptoms

The following message is issued, where *ddddddd* is the name of the table space from the catalog or directory that failed (for example, SYSIBM.SYSCOPY):

```
DSNU086I DSNUCDA1 READ I/O ERRORS ON SPACE=ddddddd.
          DATA SET NUMBER=NNN.
          I/O ERROR PAGE RANGE=aaaaaa, bbbbbb
```

This message can indicate either read or write errors. You might also receive a DSNB224I or DSNB225I message, which indicates an input or output error for the catalog or directory.

Environment

DB2 remains active.

If the DB2 directory or any catalog table is damaged, only user IDs with the RECOVERDB privilege in DSNDB06, or an authority that includes that privilege, can perform the recovery. Furthermore, until the recovery takes place, only those IDs can do anything with the subsystem. If an ID without proper authorization attempts to recover the catalog or directory, message DSNU060I is displayed. If the authorization tables are unavailable, message DSNT500I is displayed to indicate that the resource is unavailable.

Resolving the problem

Operator response: Recover each table space in the failing DB2 catalog or directory. If multiple table spaces need to be recovered, recover them in the recommended order as defined in the information about the RECOVER utility. (See Recovering catalog and directory objects in DB2 Utility Guide and Reference.)

1. Stop the failing table spaces.
2. Determine the name of the data set that failed by using one of the following methods:
 - Check *prefix*.SDSNSAMP (DSNTIJIN), which contains the JCL for installing DB2. Find the fully qualified name of the data set that failed by searching for the name of the table space that failed (the one that is identified in the message as SPACE=*ddddddd*).
 - Construct the data set name by performing one of the following actions:
 - If the table space is in the DB2 catalog, the data set name format is:
DSNC910.DSNDBC.DSNDB06.*ddddddd*.I0001.A001

The *ddddddd* is the name of the table space that failed.

- If the table space is in the DB2 directory, the data set name format is:

DSNC910.DSNDBC.DSNDB01.*ddddddd*.I0001.A001

The *ddddddd* is the name of the table space that failed.

If you do not use the default (IBM-supplied) formats, the formats for data set names might be different.

3. Use the access method services DELETE command to delete the data set, specifying the fully qualified data set name.
4. After the data set is deleted, use the access method services DEFINE command with the REUSE parameter to redefine the same data set, again specifying the same fully qualified data set name. Use the JCL for installing DB2 to determine the appropriate parameters.
5. Issue the command START DATABASE ACCESS(UT), naming the table space that is involved.
6. Use the RECOVER utility to recover the table space that failed.
7. Issue the command START DATABASE, specifying the table space name and RO or RW access, whichever is appropriate.

Recovering from integrated catalog facility failure

Some problems with VSAM data sets might occur. For example, VSAM volume data sets might be out of space or destroyed, or you might experience problems with other VSAM data sets being out of space or unable to be extended any further. You can respond to these problems

Symptoms

The symptoms for integrated catalog facility problems vary according to the underlying problems.

Recovering VSAM volume data sets that are out of space or destroyed

If the VSAM volume data set (VVDS) is out of space or destroyed, you can recover from the situation by using DB2 commands and utilities and access method services commands.

Symptoms

DB2 sends the following message to the master console:

```
DSNP012I - DSNPCT0 - ERROR IN VSAM CATALOG LOCATE FUNCTION
          FOR data_set_name
          CTLGRC=50
          CTLGRSN=zzzRRRR
          CONNECTION-ID=xxxxxxx,
          CORRELATION-ID=yyyyyyyyyy
          LUW-ID=logical-unit-of-work-id=token
```

VSAM might also issue the following message:

```
IDC3009I VSAM CATALOG RETURN CODE IS 50, REASON CODE IS
          IGGOCLaa - yy
```

In this VSAM message, *yy* is 28, 30, or 32 for an out-of-space condition. Any other values for *yy* indicate a damaged VVDS.

Environment

Your program is terminated abnormally, and one or more messages are issued.

Resolving the problem

Operator response: Begin by determining whether the VSAM volume data set is out of space or has been destroyed. Then follow these steps:

1. Determine the names of all table spaces that reside on the same volume as the VVDS. To determine the table space names, look at the VTOC entries list for that volume, which indicates the names of all the data sets on that volume.
2. Use the DB2 COPY utility to take image copies of all table spaces of the volume. Taking image copies minimizes reliance on the DB2 recovery log and can speed up the processing of the DB2 RECOVER utility (to be mentioned in a subsequent step).

If you cannot use the COPY utility, continue with this procedure. Be aware that processing time increases because more information must be obtained from the DB2 recovery log.

3. Use the command STOP DATABASE for all the table spaces that reside on the volume, or use the command STOP DB2 to stop the entire DB2 subsystem if an unusually large number or critical set of table spaces are involved.
4. If possible, use access method services to export all non-DB2 data sets that resides on that volume.
5. Use access method services to recover all non-DB2 data sets that resides on that volume.
6. Use access method services DELETE and DEFINE commands to delete and redefine the data sets for all user-defined table spaces and DB2-defined data sets when the physical data set has been destroyed. DB2 automatically deletes and redefines all other STOGROUP-defined table spaces.
You do not need to delete and redefine table spaces that are STOGROUP-defined because DB2 takes care of them automatically.
7. Issue the DB2 START DATABASE command to restart all the table spaces that were stopped in step 3. If the entire DB2 subsystem was stopped, issue the START DB2 command.
8. Use the DB2 RECOVER utility to recover any table spaces and indexes.

Related tasks

Chapter 19, "Backing up and recovering your data," on page 531

Related information

"DSNP012I" (DB2 Messages)

 z/OS Internet Library

 z/OS Internet Library

Recovering from out-of-disk-space or extent-limit problems

When a volume on which a data set is stored has insufficient space, or when the data set reaches its maximum size or its maximum number of VSAM extents, you can recover from this situation by following the appropriate procedure.

Symptoms

The symptoms vary based on the specific situation. The following messages and codes might be issued:

- DSNP007I
- DSNP001I
- -904 SQL return code (SQLSTATE '57011')

Environment

For a demand request failure during restart, the object that is supported by the data set (an index space or a table space) is stopped with deferred restart pending. Otherwise, the state of the object remains unchanged.

Diagnosing the problem

Read the following descriptions of possible problems, and determine which problem you are experiencing.

- Extend request failures: When an insert or update requires additional space, but the space is not available in the current table space or index space, DB2 issues the following message:

```
DSNP007I - DSNPmmmm - EXTEND FAILED FOR
           data-set-name. RC=rrrrrrrr
           CONNECTION-ID=xxxxxxx,
           CORRELATION-ID=yyyyyyyyyy
           LUWID-ID=logical-unit-of-work-id=token
```

- Look-ahead warning: A look-ahead warning occurs when enough space is available for a few inserts or updates, but the index space or table space is almost full. On an insert or update at the end of a page set, DB2 determines whether the data set has enough available space. DB2 uses the following values in this space calculation:
 - The primary space quantity from the integrated catalog facility (ICF) catalog
 - The secondary space quantity from the ICF catalog
 - The allocation unit size

If enough space does not exist, DB2 tries to extend the data set. If the extend request fails, DB2 issues the following message:

```
DSNP001I - DSNPmmmm - data-set-name IS WITHIN
           nK BYTES OF AVAILABLE SPACE.
           RC=rrrrrrrr
           CONNECTION-ID=xxxxxxx,
           CORRELATION-ID=yyyyyyyyyy
           LUWID-ID=logical-unit-of-work-id=token
```

Resolving the problem

What you need to do depends on your particular circumstances.

- In most cases, if the data set has not reached its maximum size, as described below, you can enlarge it. The maximum size of a data set can be:
 - Data set of a simple space: maximum size is 2 GB.
 - Data set that contains a partition: maximum size is 1, 2, or 4 GB.
 - Data set of a large partitioned table space: maximum is 4 GB.
 - Data set of an index on a large partitioned table space: maximum is 4 GB.
- If the data set has reached its maximum size, you need to follow the appropriate procedure, depending on the situation you face.

Extending a data set

If a user-defined data set reaches the maximum number of VSAM extents, you can extend the data set by adding volumes.

To extend a user-defined data set:

1. If possible, delete unneeded data on the current volume.
2. If deleting data from the volume does not solve the problem, add volumes to the data set in one of the following ways:
 - If the data set is defined in a DB2 storage group, add more volumes to the storage group by using the SQL ALTER STOGROUP statement.
 - If the data set is not defined in a DB2 storage group, add volumes to the data set by using the access method services ALTER ADDVOLUMES command.

Enlarging a fully extended user-managed data set

If a user-managed data set reaches the maximum number of VSAM extents, you can enlarge the data set.

To enlarge a user-managed data set:

1. To allow for recovery in case of failure during this procedure, ensure that you have a recent full image copy of your table spaces and indexes. Use the DSNUM option to identify the data set for table spaces or partitioning indexes.
2. Issue the command STOP DATABASE SPACENAM for the last data set of the supported object.
3. Delete the last data set by using access method services.
4. Redefine the data set, and enlarge it as necessary. The object must be a user-defined linear data set, and it should not have reached the maximum number of 32 data sets for a nonpartitioned table space (or 254 data sets for LOB table spaces). For a partitioned table space, a partitioning index, or a nonpartitioning index on a partitioned table space, the maximum is 4096 data sets.
5. Issue the command START DATABASE ACCESS (UT) to start the object for utility-only access.
6. To recover the data set that was redefined, use the RECOVER utility on the table space or index, and identify the data set by the DSNUM option (specify this DSNUM option for table spaces or partitioning indexes only).

The RECOVER utility enables you to specify a single data set number for a table space. Therefore, you need to redefine and recover only the last data set (the one that needs extension). This approach can be better than using the REORG utility if the table space is very large and contains multiple data sets, and if the extension must be done quickly.

If you do not copy your indexes, use the REBUILD INDEX utility.
7. Issue the command START DATABASE to start the object for either RO or RW access, whichever is appropriate.

Enlarging a fully extended DB2-managed data set

If a DB2-managed data set reaches the maximum number of VSAM extents, you can enlarge the data set.

To enlarge a DB2-managed data set:

1. Use the SQL statement ALTER TABLESPACE or ALTER INDEX with a USING clause. (You do not need to stop the table space before you use ALTER TABLESPACE.) You can give new values of PRIQTY and SECQTY in either the same or a new DB2 storage group.
2. Use one of the following procedures. No movement of data occurs until this step is completed.

- For indexes: If you have taken full image copies of the index, run the RECOVER INDEX utility. Otherwise, run the REBUILD INDEX utility.
- For table spaces other than LOB table spaces: Run one of the following utilities on the table space: REORG, RECOVER, or LOAD REPLACE.
- For LOB table spaces that are defined with LOG YES: Run the RECOVER utility on the table space.
- For LOB table spaces that are defined with LOG NO:
 - a. Start the table space in read-only (RO) mode to ensure that no updates are made during this process.
 - b. Make an image copy of the table space.
 - c. Run the RECOVER utility on the table space.
 - d. Start the table space in read-write (RW) mode.

Adding a data set

If a user-defined simple data set has reached its maximum size, you can use access method services to define another data set.

To add another data set:

1. Use access method services to define another data set. The name of the new data set must follow the naming sequence of the existing data sets that support the object. The last four characters of each name are a relative data set number: If the last name ends with A001, the next name must end with A002, and so on. Also, be sure to add either the character "I" or the character "J" to the name of the data set. If the object is defined in a DB2 storage group, DB2 automatically tries to create an additional data set. If that fails, access method services messages are sent to an operator to indicate the cause of the problem.
2. If necessary, correct the problem (identified in the access method services messages) to obtain additional space.

Redefining a partition (index-based partitioning)

In a partitioned object in which each partition is restricted to a single data set, if the data set reaches its maximum size, you need to redefine the partitions. Redefining a partition in an index-based partitioning environment is different than in a table-based partitioning environment.

To redefine the partitions in an index-based partitioning environment:

1. Use the SQL statement ALTER INDEX ALTER PARTITION to alter the key range values of the partitioning index.
2. Use the REORG utility with inline statistics on the partitions that are affected by the change in key range.
3. Use the RUNSTATS utility on the nonpartitioned indexes.
4. Rebind the dependent packages and plans.

Redefining a partition (table-based partitioning)

In a partitioned object in which each partition is restricted to a single data set, if the data set reaches its maximum size, you need to redefine the partitions. Redefining a partition in a table-based partitioning environment is different than in an index-based partitioning environment.

To redefine the partitions in a table-based partitioning environment:

1. Use the SQL statement ALTER TABLE ALTER PARTITION to alter the partition boundaries.

2. Use the REORG utility with inline statistics on the partitions that are affected by the change in partition boundaries.
3. Use the RUNSTATS utility on the indexes.
4. Rebind the dependent packages and plans.

Enlarging a fully extended data set for the work file database

If you have an out-of-disk-space or extent limit problem with the work file database (DSNDB07), you need to add space to the data set.

To enlarge a fully extended data set for the work file database:

Add space to the DB2 storage group, choosing one of the following approaches:

- Use SQL to create more table spaces in database DSNDB07.
- Execute these steps:
 1. Use the command STOP DATABASE(DSNDB07) to ensure that no users are accessing the database.
 2. Use SQL to alter the storage group, adding volumes as necessary.
 3. Use the command START DATABASE(DSNDB07) to allow access to the database.

Recovering from referential constraint violation

When a referential constraint is violated, the table space is available for some actions, but you cannot run certain utilities or use SQL to update the data in the table space until you recover from this situation.

Symptoms

One of the following messages is issued at the end of utility processing, depending on whether the table space is partitioned:

```
DSNU561I csect-name - TABLESPACE=tablespace-name PARTITION=partnum
          IS IN CHECK PENDING
DSNU563I csect-name - TABLESPACE=tablespace-name IS IN CHECK PENDING
```

Causes

DB2 detected one or more referential constraint violations.

Environment

The table space is still generally available. However, it is not available to the COPY, REORG, and QUIESCE utilities, or to SQL select, insert, delete, or update operations that involve tables in the table space.

Resolving the problem

Operator response:

1. Use the START DATABASE ACCESS (UT) command to start the table space for utility-only access.
2. Run the CHECK DATA utility on the table space. Consider these recommendations:
 - If you do not believe that violations exist, specify DELETE NO. If violations do not exist, specifying DELETE NO resets the CHECK-pending status; however, if violations do exist, the status is not reset.

- If you believe that violations exist, specify the DELETE YES option and an appropriate exception table. Specifying DELETE YES results in deletion of all rows that are in violation, copies them to an exception table, and resets the CHECK-pending status.
 - If the CHECK-pending status was set during execution of the LOAD utility, specify the SCOPE PENDING option. This checks only those rows that are added to the table space by LOAD, rather than every row in the table space.
3. Correct the rows in the exception table, if necessary, and use the SQL INSERT statement to insert them into the original table.
 4. Issue the command START DATABASE to start the table space for RO or RW access, whichever is appropriate. The table space is no longer in CHECK-pending status and is available for use. If you use the ACCESS (FORCE) option of this command, the CHECK-pending status is reset. However, using ACCESS (FORCE) is not recommended because it does not correct the underlying violations of referential constraints.

Related reference

"CHECK DATA" (DB2 Utility Guide and Reference)

Recovering from distributed data facility failure

You can recover from various problems that occur for the distributed data facility (DDF).

Symptoms

The symptoms for DDF failures vary based on the precise problems. The symptoms include messages, SQL return codes, and apparent wait states.

Recovering from conversation failure

A VTAM APPC or TCP/IP conversation might fail during or after allocation. The conversation is not available for use until you recover from the situation.

Symptoms

VTAM or TCP/IP returns a resource-unavailable condition along with the appropriate diagnostic reason code and message. A DSNL500 or DSNL511 (conversation failed) message is sent to the console for the first failure to a location for a specific logical unit (LU) mode or TCP/IP address. All other threads that detect a failure from that LU mode or IP address are suppressed until communications to that LU that use that mode are successful. DB2 returns messages DSNL501I and DSNL502I. Message DSNL501I usually means that the other subsystem is not operational. When the error is detected, it is reported by a console message, and the application receives an SQL return code. For DB2 private protocol access, SQLCODE -904 (SQLSTATE '57011') is returned with resource type 1001, 1002, or 1003. The resource name in the SQLCA contains VTAM return codes such as RTNCD, FDBK2, RCPRI, and RCSEC, and any SNA SENSE information.

If you use application-directed access or DRDA as the database protocols, SQLCODE -30080 is returned to the application. The SQLCA contains the VTAM diagnostic information, which contains only the RCPRI and RCSEC codes. For SNA communications errors, SQLCODE -30080 is returned. For TCP/IP connections, SQLCODE -30081 is returned.

Environment

The application can choose to request rollback or commit, both of which deallocate all but the first conversation between the allied thread and the remote database access thread. A commit or rollback message is sent over this remaining conversation.

Errors during the deallocation process of the conversation are reported through messages, but they do not stop the commit or rollback processing. If the conversation that is used for the commit or rollback message fails, the error is reported. If the error occurred during a commit process and if the remote database access was read-only, the commit process continues. Otherwise the commit process is rolled back.

Diagnosing the problem

System programmer response: Review the VTAM or TCP/IP return codes, and possibly discuss the problem with a communications expert. Many VTAM or TCP/IP errors, besides the error of an inactive remote LU or TCP/IP errors, require a person who has a knowledge of VTAM or TCP/IP and the network configuration to diagnose them.

Resolving the problem

Operator response: Correct the cause of the unavailable-resource condition by taking the action that is required by the diagnostic messages that are displayed on the console.

Recovering from communications database failure

You need to recover the communications database (CDB) when a failure occurs during an attempt to access the CDB.

Symptoms

A DSNL700I message, which indicates that a resource-unavailable condition exists, is sent to the console. Other messages that describe the cause of the failure are also sent to the console.

Environment

If the distributed data facility (DDF) has already started when an individual CDB table becomes unavailable, DDF does not terminate. Depending on the severity of the failure, threads are affected as follows:

- The threads receive a -904 SQL return code (SQLSTATE '57011') with resource type 1004 (CDB).
- The threads continue using VTAM default values.

The only threads that receive a -904 SQL return code are those that access locations that have not had any prior threads. DB2 and DDF remain operational.

Resolving the problem

Operator response:

1. Examine the messages to determine the source of the error.
2. Correct the error, and then stop and restart DDF.

Recovering from a problem with a communications database that is incorrectly defined

You need to recover from a situation in which the communications database (CDB) is not correctly defined. This problem occurs when distributed data facility (DDF) is started and the DB2 catalog is accessed to verify the CDB definitions.

Symptoms

A DSNL701I, DSNL702I, DSNL703I, DSNL704I, or DSNL705I message is issued to identify the problem. Other messages that describe the cause of the failure are also sent to the console.

Environment

DDF fails to start. DB2 continues to run.

Resolving the problem

Operator response:

1. Examine the messages to determine the source of the error.
2. Correct the error, and then restart DDF.

Recovering from database access thread failure

When a database access thread is deallocated, a conversation failure occurs, and you need to recover from this situation.

Symptoms

In the event of a failure of a database access thread, the DB2 server terminates the database access thread only if a unit of recovery exists. The server deallocates the database access thread and then deallocates the conversation with an abnormal indication (a negative SQL code), which is subsequently returned to the requesting application. The returned SQL code depends on the type of remote access:

- DRDA access

For a database access thread or non-DB2 server, a DDM error message is sent to the requesting site, and the conversation is deallocated normally. The SQL error status code is a -30020 with a resource type 1232 (agent permanent error received from the server).

- DB2 private protocol access

The application program receives a -904 SQL return code (SQLSTATE '57011') with a resource type 1005 at the requesting site. The SNA sense code in the resource name contains the DB2 reason code that describes the failure.

Environment

Normal DB2 error recovery mechanisms apply, with the following exceptions:

- Errors that are encountered in the functional recovery routine are automatically converted to rollback situations. The allied thread experiences conversation failures.
- Errors that occur during commit, rollback, and deallocate within the DDF function do not normally cause DB2 to abend. Conversations are deallocated, and the database access thread is terminated. The allied thread experiences conversation failures.

Diagnosing the problem

System programmer response: Collect all diagnostic information that is related to the failure at the serving site. For a DB2 database access thread (DBAT), a dump is produced at the server.

Resolving the problem

Operator response: Communicate with the operator at the other site to take the appropriate corrective action, regarding the messages that are displayed at both the requesting and responding sites. Ensure that you and operators at the other sites gather the appropriate diagnostic information and give it to the programmer for diagnosis.

Recovering from VTAM failure

When VTAM terminates or fails, you need to recover from the situation.

Symptoms

VTAM messages and DB2 messages are issued to indicate that distributed data facility (DDF) is terminating and to explain why.

Causes

Environment

DDF terminates. An abnormal VTAM failure or termination causes DDF to issue a STOP DDF MODE(FORCE) command. The VTAM commands Z NET,QUICK and Z NET,CANCEL cause an abnormal VTAM termination. A Z NET,HALT causes a STOP DDF MODE(QUIESCE) to be issued by DDF.

Resolving the problem

Operator response: Correct the condition that is described in the messages that are received at the console, and restart VTAM and DDF.

Recovering from TCP/IP failure

When TCP/IP terminates or fails, you need to recover from this situation.

Symptoms

TCP/IP messages and DB2 messages are issued to indicate that TCP/IP is unavailable.

Environment

Distributed data facility (DDF) periodically attempts to reconnect to TCP/IP. If the TCP/IP listener fails, DDF automatically tries to re-establish the TCP/IP listener for the DRDA SQL port or the resync port every three minutes. TCP/IP connections cannot be established until the TCP/IP listener is re-established.

Resolving the problem

Operator response:

1. Examine the messages that are received at the console to determine the cause of the problem.
2. Correct the condition.
3. Restart TCP/IP. You do not need to restart DDF after a TCP/IP failure.

Recovering from remote logical unit failure

When a series of conversation or change number of sessions (CNOS) failures occur from a remote logical unit (LU), you need to recover from this situation.

Symptoms

Message DSNL501I is issued when a CNOS request to a remote LU fails. The CNOS request is the first attempt to connect to the remote site and must be negotiated before any conversations can be allocated. Consequently, if the remote LU is not active, message DSNL501I is displayed to indicate that the CNOS request cannot be negotiated. Message DSNL500I is issued only once for all the SQL conversations that fail as a result of a remote LU failure.

Message DSNL502I is issued for system conversations that are active to the remote LU at the time of the failure. This message contains the VTAM diagnostic information about the cause of the failure.

Environment

Any application communications with a failed LU receives a message to indicate a resource-unavailable condition. The application programs receive SQL return code -30080 for DRDA access and SQL return code -904 (SQLSTATE '57011') for DB2 private protocol access. Any attempt to establish communication with such an LU fails.

Resolving the problem

System programmer response: Communicate with the other involved sites regarding the unavailable-resource condition, and request that appropriate corrective action be taken. If a DSNL502 message is received, activate the remote LU, or ask another operator to do so.

Recovering from an indefinite wait condition

You need to recover from a situation in which an allied thread is waiting indefinitely for a response from a remote subsystem, or a database access thread is waiting for a response from the local subsystem.

Symptoms

An application is in an indefinitely long wait condition. This can cause other DB2 threads to fail due to resources that are held by the waiting thread. DB2 sends an error message to the console, and the application program receives an SQL return code.

Environment

DB2 does not respond.

Diagnosing the problem

Operator response: To check for very long waits, look to see if the conversation timestamp is changing from the last time it was used. If it is changing, the conversation thread is not hung, but it is taking more time for a long query. Also, look for conversation state changes, and determine what they mean.

Resolving the problem

Operator response:

1. Use the DISPLAY THREAD command with the LOCATION and DETAIL options to identify the LUWID and the session allocation for the waiting thread.
2. Use the CANCEL DDF THREAD command to cancel the waiting thread.
3. If the CANCEL DDF THREAD command fails to break the wait (because the thread is not suspended in DB2), try using VTAM commands such as VARY TERM,SID=xxx.

Related tasks

“Canceling threads” on page 466

Recovering database access threads after security failure

During database access thread allocation, the remote might not have the proper security to access DB2 through distributed data facility (DDF). When this happens, you can recover from the situation.

Symptoms

Message DSNL500I is issued at the requester for VTAM conversations (if it is a DB2 subsystem) with return codes RTNCD=0, FDBK2=B, RCPRI=4, and RCSEC=5. These return codes indicate that a security violation has occurred. The server has deallocated the conversation because the user is not allowed to access the server. For conversations that use DRDA access, LU 6.2 communications protocols present specific reasons for why the user access failed, and these reasons are communicated to the application. If the server is a DB2 database access thread, message DSNL030I is issued to describe what caused the user to be denied access into DB2 through DDF. No message is issued for TCP/IP connections.

If the server is a DB2 subsystem, message DSNL030I is issued. Otherwise, the system programmer needs to refer to the documentation of the server. For the application uses DRDA access, SQLCODE -30082 is returned. If the application uses DB2 private protocol access, it receives SQLCODE -904 (SQLSTATE '57011') with reason code 00D3103D, to indicate that a resource is unavailable.

Causes

This problem is caused by a remote user who attempts to access DB2 through DDF without the necessary security authority.

Resolving the problem

Operator response:

1. Read about the DB2 code 00D3103D.
2. Take the appropriate action:
 - If the security failure involves a DB2 database access thread, provide the DSNL030I message to the system programmer.

- If the security failure does not involve a DB2 server, work with the operator or programmer at the server to get diagnostic information that is needed by the system programmer.

Related reference

"00D3103D" (DB2 Codes)

Performing remote-site disaster recovery

When your local system experiences damage or disruption that prevents recovery from that site, you can recover by using a remote site that you have set up for this purpose.

Symptoms

The specific symptoms of a disaster that affects your local system hardware vary, but when this happens, the affected DB2 subsystem is not operational.

Causes

Your local system hardware has suffered physical damage.

Resolving the problem

System programmer response: Coordinate the activities that are detailed in "Restoring data from image copies and archive logs."

Operator response: At the remote-site, the disaster-recovery procedures differ from other recovery procedures because you cannot use the hardware at your local DB2 site to recover data. Instead, you use hardware at a remote site to recover after a disaster by using one of a variety of methods.

Recovering from a disaster by using system-level backups

If you have recent system-level backups, you can use those backups along with one of several utilities to recover after a disaster.

To recover from a disaster by using system-level backups:

For a remote site recovery procedure where tape volumes that contain system data are sent from the production site, specify the dump class that is available at the remote site by using the following installation options on installation panel DSNTIP6:

- Either RESTORE FROM DUMP or RECOVER FROM DUMP
- DUMP CLASS NAME

Restoring data from image copies and archive logs

Follow the appropriate procedure for restoring from image copies and archive logs, depending on whether you are in a data sharing environment. Both procedures assume that all logs, copies, and reports are available at the recovery site.

Related information

 z/OS Internet Library

Restoring data in a non-data sharing environment

If you are in a non-data sharing environment, you might need to recover from a disaster by restoring data from image copies and logs. The procedure assumes that all logs, copies, and reports are available at the recovery site.

To recover from a disaster in a non-data sharing environment by using image copies and archive logs:

1. If an integrated catalog facility catalog does not already exist, run job DSNTIJCA to create a user catalog.
2. Use the access method services IMPORT command to import the integrated catalog facility catalog.
3. Restore DB2 libraries. Some examples of libraries that you might need to restore include:
 - DB2 SMP/E libraries
 - User program libraries
 - User DBRM libraries
 - DB2 CLIST libraries
 - DB2 libraries that contain customized installation jobs
 - JCL for creating user-defined table spaces
4. Use IDCAMS DELETE NOSCRATCH to delete all catalog and user objects. (Because step 2 imports a user ICF catalog, the catalog reflects data sets that do not exist on disk.)
5. Obtain a copy of installation job DSNTIJIN, which creates DB2 VSAM and non-VSAM data sets. Change the volume serial numbers in the job to volume serial numbers that exist at the recovery site. Comment out the steps that create DB2 non-VSAM data sets, if these data sets already exist. Run DSNTIJIN. However, do not run DSNTIJID.
6. Recover the BSDS:
 - a. Use the access method services REPRO command to restore the contents of one BSDS data set (allocated in step 5). You can find the most recent BSDS image in the last file (archive log with the highest number) on the latest archive log tape.
 - b. Determine the RBA range for this archive log by using the print log map utility (DSNJU004) to list the current BSDS contents. Find the most recent archive log in the BSDS listing, and add 1 to its ENDRBA value. Use this as the STARTRBA. Find the active log in the BSDS listing that starts with this RBA, and use its ENDRBA as the ENDRBA.
 - c. Register this latest archive log tape data set in the archive log inventory of the BSDS that you just restored by using the change log inventory utility (DSNJU003). This step is necessary because the BSDS image on an archive log tape does not reflect the archive log data set that resides on that tape. After these archive logs are registered, use the print log map utility (DSNJU004) to list the contents of the BSDS.
 - d. Adjust the active logs in the BSDS by using the change log inventory utility (DSNJU003), as necessary:
 - 1) To delete all active logs in the BSDS, use the DELETE option of DSNJU003. Use the BSDS listing that is produced in step 6c to determine the active log data set names.

- 2) To add the active log data sets to the BSDS, use the NEWLOG statement of DSNJU003. Do not specify a STARTRBA or ENDRBA in the NEWLOG statement. This specification indicates to DB2 that the new active logs are empty.
 - e. If you are using the DB2 distributed data facility, update the LOCATION and the LUNAME values in the BSDS by running the change log inventory utility with the DDF statement.
 - f. List the new BSDS contents by using the print log map utility (DSNJU004). Ensure that the BSDS correctly reflects the active and archive log data set inventories. In particular, ensure that:
 - All active logs show a status of NEW and REUSABLE.
 - The archive log inventory is complete and correct (for example, the start and end RBAs are correct).
 - g. If you are using dual BSDSs, make a copy of the newly restored BSDS data set to the second BSDS data set.
7. Optional: Restore archive logs to disk. Archive logs are typically stored on tape, but restoring them to disk might speed later steps. If you elect this option, and the archive log data sets are not cataloged in the primary integrated catalog facility catalog, use the change log inventory utility to update the BSDS. If the archive logs are listed as cataloged in the BSDS, DB2 allocates them by using the integrated catalog and not the unit or VOLSER that is specified in the BSDS. If you are using dual BSDSs, remember to update both copies.
 8. Use the DSN1LOGP utility to determine which transactions were in process at the end of the last archive log. Use the following job control language where *yyyyyyyyyyyyyy* is the STARTRBA of the last complete checkpoint within the RBA range on the last archive log from the previous print log map:

```
//SAMP EXEC PGM=DSN1LOGP
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//ARCHIVE DD DSN=last-archive, DISP=(OLD,KEEP),UNIT=TAPE,
            LABEL=(2,SL),VOL=SER=volsr1
            (NOTE FILE 1 is BSDS COPY)
//SYSIN DD *
STARTRBA(yyyyyyyyyyyyyy) SUMMARY(ONLY)
/*
```

DSN1LOGP generates a report.

9. Examine the DSN1LOGP output, and identify any utilities that were executing at the end of the last archive log. Determine the appropriate recovery action to take on each table space that is involved in a utility job. If DSN1LOGP output showed that utilities are inflight (PLAN=DSNUTIL), examine SYSUTILX to identify the utility status and determine the appropriate recovery approach.
10. Modify DSNZPxxx parameters:
 - a. Run the DSNTINST CLIST in UPDATE mode.
 - b. To defer processing of all databases, select DATABASES TO START AUTOMATICALLY from panel DSNTIPB. Panel DSNTIPS opens. On panel DSNTIPS, type DEFER in the first field and ALL in the second field; then press **Enter**. You are returned to panel DSNTIPB.
 - c. To specify where you are recovering, select OPERATOR FUNCTIONS from panel DSNTIPB. Panel DSNTIPO opens. From panel DSNTIPO, type RECOVERYSITE in the SITE TYPE field. Press **Enter** to continue.
 - d. Optional: Specify which archive log to use by selecting OPERATOR FUNCTIONS from panel DSNTIPB. Panel DSNTIPO opens. From panel

DSNTIPO, type YES in the READ ARCHIVE COPY2 field if you are using dual archive logging and want to use the second copy of the archive logs. Press **Enter** to continue.

- e. Reassemble DSNZPxxx by using job DSNTIJUZ (produced by the CLIST started in the first step of this procedure).

At this point, you have the log, but the table spaces have not been recovered. With DEFER ALL, DB2 assumes that the table spaces are unavailable but does the necessary processing to the log. This step also handles the units of recovery that are in process.

- 11. Create a conditional restart control record by using the change log inventory utility with one of the following forms of the CRESTART statement:

- CRESTART CREATE, ENDRBA=nnnnnnnn000

The *nnnnnnnn000* equals a value that is one more than the ENDRBA of the latest archive log.

- CRESTART CREATE, ENDTIME=nnnnnnnnnnnn

The *nnnnnnnnnnnn* is the end time of the log record. Log records with a timestamp later than *nnnnnnnnnnnn* are truncated.

- 12. Enter the command START DB2 ACCESS(MAINT).

Even though DB2 marks all table spaces for deferred restart, log records are written so that in-abort and inflight units of recovery are backed out. In-commit units of recovery are completed, but no additional log records are written at restart to cause this. This happens when the original redo log records are applied by the RECOVER utility.

At the primary site, DB2 probably committed or aborted the inflight units of recovery, but you have no way of knowing.

During restart, DB2 accesses two table spaces that result in DSNT501I, DSNT500I, and DSNL700I resource unavailable messages, regardless of DEFER status. The messages are normal and expected, and you can ignore them.

The following return codes can accompany the message. Other codes are also possible.

00C90081

This return code is issued for activity against the object that occurs during restart as a result of a unit of recovery or pending writes. In this case, the status that is shown as a result of DISPLAY is STOP,DEFER.

00C90094

Because the table space is currently only a defined VSAM data set, it is in a state that DB2 does not expect.

00C900A9

An attempt was made to allocate a deferred resource.

- 13. Resolve the indoubt units of recovery. The RECOVER utility, which you run in a subsequent step, fails on any table space that has indoubt units of recovery. Because of this, you must resolve them first. Determine the proper action to take (commit or abort) for each unit of recovery. To resolve indoubt units of recovery, see "Resolving indoubt units of recovery" on page 522. From an install SYSADM authorization ID, enter the RECOVER INDOUBT command for all affected transactions.

- 14. Recover the catalog and directory. The RECOVER function includes: RECOVER TABLESPACE, RECOVER INDEX, or REBUILD INDEX. If you

have an image copy of an index, use RECOVER INDEX. If you do not have an image copy of an index, use REBUILD INDEX to reconstruct the index from the recovered table space.

- a. Recover DSNDB01.SYSUTILX. This must be a separate job step.
 - b. Recover all indexes on SYSUTILX. This must be a separate job step.
 - c. Determine whether a utility was running at the time the latest archive log was created by entering the DISPLAY UTILITY(*) command, and record the name and current phase of any utility that is running. (You cannot restart a utility at the recovery site that was interrupted at the disaster site. You must use the TERM UTILITY command to terminate it. Use the TERM UTILITY command on a utility that is operating on any object except DSNDB01.SYSUTILX.)
 - d. Run the DIAGNOSE utility with the DISPLAY SYSUTILX option. The output consists of information about each active utility, including the table space name (in most cases). This is the only way to correlate the object name with the utility. Message DSNU866I gives information about the utility, and DSNU867I gives the database and table space name in USUDBNAM and USUSPNAM, respectively.
 - e. Use the TERM UTILITY command to terminate any utilities that are in progress on catalog or directory table spaces.
 - f. Recover the rest of the catalog and directory objects, starting with DBD01, in the order shown in the description of the RECOVER utility. (See Recovering catalog and directory objects in DB2 Utility Guide and Reference.)
15. Define and initialize the work file database:
 - a. Define temporary work files. Use installation job DSNTIJTM as a model.
 - b. Issue the command START DATABASE(*work-file-database*) to start the work file database.
 16. Use any method that you want to verify the integrity of the DB2 catalog and directory. Use the catalog queries in member DSNTESQ of data set DSN910.SDSNSAMP after the work file database is defined and initialized.
 17. If you use data definition control support, recover the objects in the data definition control support database.
 18. If you use the resource limit facility, recover the objects in the resource limit control facility database.
 19. Modify DSNZPxxx to restart all databases:
 - a. Run the DSNTINST CLIST in UPDATE mode.
 - b. From panel DSNTIPB, select DATABASES TO START AUTOMATICALLY. Panel DSNTIPS opens. Type RESTART in the first field and ALL in the second field, and press **Enter**. You are returned to DSNTIPB.
 - c. Reassemble DSNZPxxx by using job DSNTIJUZ (produced by the CLIST started in step 3 on page 680).
 20. Stop DB2.
 21. Start DB2.
 22. Make a full image copy of the catalog and directory.
 23. Recover user table spaces and index spaces. If utilities were running on any table spaces or index spaces, see “What to do about utilities that were in progress at time of failure” on page 691. You cannot restart a utility at the recovery site that was interrupted at the disaster site. Use the TERM UTILITY command to terminate any utilities that are running against user table spaces or index spaces.

- a. To determine which, if any, of your table spaces or index spaces are user-managed, perform the following queries for table spaces and index spaces.

- Table spaces:

```
SELECT * FROM SYSIBM.SYSTABLEPART WHERE STORTYPE='E';
```

- Index spaces:

```
SELECT * FROM SYSIBM.SYSINDEXPART WHERE STORTYPE='E';
```

To allocate user-managed table spaces or index spaces, use the access method services DEFINE CLUSTER command. To find the correct IPREFIX for the DEFINE CLUSTER command, perform the following queries for table spaces and index spaces.

- Table spaces:

```
SELECT DBNAME, TSNAME, PARTITION, IPREFIX FROM SYSIBM.SYSTABLEPART  
WHERE DBNAME=dbname AND TSNAME=tsname  
ORDER BY PARTITION;
```

- Index spaces:

```
SELECT IXNAME, PARTITION, IPREFIX FROM SYSIBM.SYSINDEXPART  
WHERE IXCREATOR=ixcreator AND IXNAME=ixname  
ORDER BY PARTITION;
```

Now you can perform the DEFINE CLUSTER command with the correct IPREFIX (I or J) in the data set name:

```
catname.DSNDBx.dbname.psname.y0001.znnn
```

The *y* can be either I or J, *x* is C (for VSAM clusters) or D (for VSAM data components), and *psname* is either the table space or index space name.

- b. If your user table spaces or index spaces are STOGROUP-defined, and if the volume serial numbers at the recovery site are different from those at the local site, use the SQL statement ALTER STOGROUP to change them in the DB2 catalog.
- c. Recover all user table spaces and index spaces from the appropriate image copies. If you do not copy your indexes, use the REBUILD INDEX utility to reconstruct the indexes.
- d. Start all user table spaces and index spaces for read-write processing by issuing the command START DATABASE with the ACCESS(RW) option.
- e. Resolve any remaining CHECK-pending states that would prevent COPY execution.
- f. Run queries for which the results are known.
24. Make full image copies of all table spaces and indexes with the COPY YES attribute.
25. Finally, compensate for work that was lost since the last archive was created by rerunning online transactions and batch jobs.

Determine what to do about any utilities that were in progress at the time of failure.

Related concepts

“What to do about utilities that were in progress at time of failure” on page 691

“Preparations for disaster recovery” on page 545

“Migration step 1: Actions that you must perform before migration” (DB2 Installation Guide)

Related tasks

“Defining data sets” on page 32

“Invoking the CLIST” (DB2 Installation Guide and Reference)

Related reference

"DSN1LOGP" (DB2 Utility Guide and Reference)

Restoring data in a data sharing environment

If you are in a data sharing environment, you might need to recover from a disaster by restoring data from image copies and logs. The procedure assumes that all logs, copies, and reports are available at the recovery site.

Additional recovery procedures for data sharing environments are also available.

To recover from a disaster by using image copies and archive logs:

1. If you have information in your coupling facility from practice startups, remove old information from the coupling facility. If you do not have old information in your coupling facility, continue with the step 2.
 - a. Enter the following z/OS command to display the structures for this data sharing group:

```
D XCF,STRUCTURE,STRNAME=grpname*
```

- b. For group buffer pools and the lock structure, enter the following command to force off the connection of those structures:

```
SETXCF FORCE,CONNECTION,STRNAME=strname,CONNNAME=ALL
```

Connections for the SCA are not held at termination; therefore you do not need to force off any SCA connections.

- c. Delete all the DB2 coupling facility structures by using the following command for each structure:

```
SETXCF FORCE,STRUCTURE,STRNAME=strname
```

This step is necessary to remove old information that exists in the coupling facility from your practice startup when you installed the group.

2. If an integrated catalog facility catalog does not already exist, run job DSNTIJCA to create a user catalog.
3. Use the access method services IMPORT command to import the integrated catalog facility catalog.
4. Restore DB2 libraries. Some examples of libraries that you might need to restore include:
 - DB2 SMP/E libraries
 - User program libraries
 - User DBRM libraries
 - DB2 CLIST libraries
 - DB2 libraries that contain customized installation jobs
 - JCL for creating user-defined table spaces
5. Use IDCAMS DELETE NOSCRATCH to delete all catalog and user objects. (Because step 3 imports a user ICF catalog, the catalog reflects data sets that do not exist on disk.)
6. Obtain a copy of the installation job DSNTIJIN, which creates DB2 VSAM and non-VSAM data sets, for the first data sharing member. Change the volume serial numbers in the job to volume serial numbers that exist at the recovery site. Comment out the steps that create DB2 non-VSAM data sets, if these data sets already exist. Run DSNTIJIN on the first data sharing member. However, do not run DSNTIJID.

For subsequent members of the data sharing group, run the DSNTIJIN that defines the BSDS and logs.

7. Recover the BSDS by following these steps for each member in the data sharing group:
 - a. Use the access method services REPRO command to restore the contents of one BSDS data set (allocated in step 6 on page 685) on each member. You can find the most recent BSDS image in the last file (archive log with the highest number) on the latest archive log tape.
 - b. Determine the RBA and LRSN ranges for this archive log by using the print log map utility (DSNJU004) to list the current BSDS contents. Find the most recent archive log in the BSDS listing, and add 1 to its ENDRBA value. Use this as the STARTRBA. Find the active log in the BSDS listing that starts with this RBA, and use its ENDRBA as the ENDRBA. Use the STARTLRSN and ENDLRSN of this active log data set as the LRSN range (STARTLRSN and ENDLRSN) for this archive log.
 - c. Register this latest archive log tape data set in the archive log inventory of the BSDS that you just restored by using the change log inventory utility (DSNJU003). This step is necessary because the BSDS image on an archive log tape does not reflect the archive log data set that resides on that tape. Running DSNJU003 is critical for data sharing groups. Include the group buffer pool checkpoint information that is stored in the BSDS from the most recent archive log.

After these archive logs are registered, use the print log map utility (DSNJU004) with the GROUP option to list the contents of all BSDSs. You receive output that includes the start and end LRSN and RBA values for the latest active log data sets (shown as NOTREUSABLE). If you did not save the values from the DSNJ003I message, you can get those values by running DSNJU004, which creates output as shown below

The following sample DSNJU004 output shows the (partial) information for the archive log member DB1G.

ACTIVE LOG COPY 1 DATA SETS					
START RBA/LRSN/TIME	END RBA/LRSN/TIME	DATE	LTIME	DATA SET INFORMATION	
-----	-----	-----	-----	-----	
000001C20000 ADFA0FB26C6D 1996.361 23:37:48.4	000001C67FFF ADFA208AA36B 1996.362 00:53:10.1	1996.358	17:25	DSN=DSNDB0G.DB1G.LOGCOPY1.DS03 STATUS=TRUNCATED, REUSABLE	
000001C68000 ADFA208AA36C 1996.362 00:53:10.1	000001D4FFFF AE3C45273A77 1997.048 15:28:23.5	1996.358	17:25	DSN=DSNDB0G.DB1G.LOGCOPY1.DS01 STATUS=TRUNCATED, NOTREUSABLE	
000001D50000 AE3C45273A78 1997.048 15:28:23.5	0000020D3FFF	1996.358	17:25	DSN=DSNDB0G.DB1G.LOGCOPY1.DS02 STATUS=NOTREUSABLE	

The following sample DSNJU004 output shows the (partial) information for the archive log member DB2G.

ACTIVE LOG COPY 1 DATA SETS					
START RBA/LRSN/TIME	END RBA/LRSN/TIME	DATE	LTIME	DATA SET INFORMATION	
-----	-----	-----	-----	-----	
000000000000 0000.000 00:00:00.0	000000000000 0000.000 00:00:00.0	1996.361	14:14	DSN=DSNDB0G.DB2G.LOGCOPY1.DS03 STATUS=NEW, REUSABLE	
000000000000 ADFA00BB70FB 1996.361 22:30:51.4	0000000D6FFF AE3C45276DD7 1997.048 15:28:23.7	1996.361	14:14	DSN=DSNDB0G.DB2G.LOGCOPY1.DS01 STATUS=TRUNCATED, NOTREUSABLE	
0000000D7000 AE3C45276DD8 1997.048 15:28:23.7	00000045AFFE	1996.361	14:14	DSN=DSNDB0G.DB2G.LOGCOPY1.DS02 STATUS=NOTREUSABLE	

- d. Adjust the active logs in the BSDS by using the change log inventory utility (DSNJU003), as necessary:
 - 1) To delete all active logs in the BSDS, use the DELETE option of DSNJU003. Use the BSDS listing that is produced in step 7c on page 686 to determine the active log data set names.
 - 2) To add the active log data sets to the BSDS, use the NEWLOG statement of DSNJU003. Do not specify a STARTRBA or ENDRBA in the NEWLOG statement. This specification indicates to DB2 that the new active logs are empty.
 - e. If you are using the DB2 distributed data facility, update the LOCATION and the LUNAME values in the BSDS by running the change log inventory utility with the DDF statement.
 - f. List the new BSDS contents by using the print log map utility (DSNJU004). Ensure that the BSDS correctly reflects the active and archive log data set inventories. In particular, ensure that:
 - All active logs show a status of NEW and REUSABLE.
 - The archive log inventory is complete and correct (for example, the start and end RBAs are correct).
 - g. If you are using dual BSDSs, make a copy of the newly restored BSDS data set to the second BSDS data set.
8. Optional: Restore archive logs to disk for each member. Archive logs are typically stored on tape, but restoring them to disk might speed later steps. If you elect this option, and the archive log data sets are not cataloged in the primary integrated catalog facility catalog, use the change log inventory utility to update the BSDS. If the archive logs are listed as cataloged in the BSDS, DB2 allocates them by using the integrated catalog and not the unit or VOLSER that is specified in the BSDS. If you are using dual BSDSs, remember to update both copies.
 9. Use the DSN1LOGP utility to determine, for each member of the data sharing group, which transactions were in process at the end of the last archive log. Use the following job control language where *yyyyyyyyyyyyyy* is the STARTRBA of the last complete checkpoint within the RBA range on the last archive log from the previous print log map:


```
//SAMP EXEC PGM=DSN1LOGP
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//ARCHIVE DD DSN=last-archive, DISP=(OLD,KEEP),UNIT=TAPE,
           LABEL=(2,SL),VOL=SER=volser1
           (NOTE FILE 1 is BSDS COPY
//SYSIN DD *
STARTRBA(yyyyyyyyyyyyyy) SUMMARY(ONLY)
/*
```

DSN1LOGP generates a report.

10. Examine the DSN1LOGP output for each data sharing member, and identify any utilities that were executing at the end of the last archive log. Determine the appropriate recovery action to take on each table space that is involved in a utility job. If DSN1LOGP output showed that utilities are inflight (PLAN=DSNUTIL), examine SYSUTILX to identify the utility status and determine the appropriate recovery approach.
11. Modify DSNZPxxx parameters for each member of the data sharing group:
 - a. Run the DSNTINST CLIST in UPDATE mode.
 - b. To defer processing of all databases, select DATABASES TO START AUTOMATICALLY from panel DSNTIPB. Panel DSNTIPS opens. On panel

DSNTIPS, type DEFER in the first field and ALL in the second field; then press **Enter**. You are returned to panel DSNTIPB.

- c. To specify where you are recovering, select OPERATOR FUNCTIONS from panel DSNTIPB. Panel DSNTIPO opens. From panel DSNTIPO, type RECOVERYSITE in the SITE TYPE field. Press **Enter** to continue.
- d. Optional: Specify which archive log to use by selecting OPERATOR FUNCTIONS from panel DSNTIPB. Panel DSNTIPO opens. From panel DSNTIPO, type YES in the READ ARCHIVE COPY2 field if you are using dual archive logging and want to use the second copy of the archive logs. Press **Enter** to continue.
- e. Reassemble DSNZPxxx by using job DSNTIJUZ (produced by the CLIST started in the first step of this procedure).

At this point, you have the log, but the table spaces have not been recovered. With DEFER ALL, DB2 assumes that the table spaces are unavailable but does the necessary processing to the log. This step also handles the units of recovery that are in process.

12. Create a conditional restart control record for each data sharing member by using the change log inventory utility with one of the following forms of the CRESTART statement:

- CRESTART CREATE,ENDLRSN=nnnnnnnnnnnn

The *nnnnnnnnnnnn* is the LRSN of the last log record that is to be used during restart.

- CRESTART CREATE,ENDTIME=nnnnnnnnnnnn

The *nnnnnnnnnnnn* is the end time of the log record. Log records with a timestamp later than *nnnnnnnnnnnn* are truncated.

Use the same LRSN or GMT TIMESTAMP value for all members in a data sharing group. Determine the ENDLRSN value by using one of the following methods:

- Use the DSN1LOGP summary utility. In the “Summary of Completed Events” section, find the lowest LRSN value that is listed in the DSN1213I message for the data sharing group. Use this value for the ENDLRSN in the CRESTART statement.
- Use the print log map utility (DSNJU004) to list the BSDS contents. Find the ENDLRSN of the last log record that is available for each active member of the data sharing group. Subtract 1 from the lowest ENDLRSN in the data sharing group. Use this value for the ENDLRSN in the CRESTART statement. (In the sample output that is shown in step 7c on page 686, the value is AE3C45273A77 - 1, which is AE3C45273A76.)
- If only the console logs are available, use the archive offload message (DSNJ003I) to obtain the ENDLRSN. Compare the ending LRSN values for the archive logs of all members. Subtract 1 from the lowest LRSN in the data sharing group. Use this value for the ENDLRSN in the CRESTART statement. (In the sample output that is shown in step 7c on page 686, the value is AE3C45273A77 - 1, which is AE3C45273A76.)

DB2 discards any log information in the bootstrap data set and the active logs with an RBA greater than or equal to *nnnnnnnnnn000* or an LRSN greater than *nnnnnnnnnnnn* as listed in the preceding CRESTART statements.

Use the print log map utility to verify that the conditional restart control record that you created in the previous step is active.

13. Enter the command START DB2 ACCESS(MAINT). If a discrepancy exists among the print log map reports as to the number of members in the group, which would be an unlikely occurrence, record the one that shows the highest

number of members. Start this DB2 subsystem first using ACCESS(MAINT). DB2 prompts you to start each additional DB2 subsystem in the group.

After all additional members are successfully restarted, and if you are going to run single-system data sharing at the recovery site, stop all except one of the DB2 subsystems by using the STOP DB2 command with MODE(QUIESCE).

If you planned to use the light mode when starting the DB2 group, add the LIGHT parameter to the START command. Start the members that run in LIGHT(NO) mode first, followed by the LIGHT(YES) members.

Even though DB2 marks all table spaces for deferred restart, log records are written so that in-abort and inflight units of recovery are backed out.

In-commit units of recovery are completed, but no additional log records are written at restart to cause this. This happens when the original redo log records are applied by the RECOVER utility.

At the primary site, DB2 probably committed or aborted the inflight units of recovery, but you have no way of knowing.

During restart, DB2 accesses two table spaces that result in DSNT501I, DSNT500I, and DSNL700I resource unavailable messages, regardless of DEFER status. The messages are normal and expected, and you can ignore them.

The following return codes can accompany the message. Other codes are also possible.

00C90081

This return code is issued for activity against the object that occurs during restart as a result of a unit of recovery or pending writes. In this case, the status that is shown as a result of DISPLAY is STOP,DEFER.

00C90094

Because the table space is currently only a defined VSAM data set, it is in a state that DB2 does not expect.

00C900A9

An attempt was made to allocate a deferred resource.

14. Resolve the indoubt units of recovery. The RECOVER utility, which you run in a subsequent step, fails on any table space that has indoubt units of recovery. Because of this, you must resolve them first. Determine the proper action to take (commit or abort) for each unit of recovery. To resolve indoubt units of recovery, see "Resolving indoubt units of recovery" on page 522. From an install SYSADM authorization ID, enter the RECOVER INDOUBT command for all affected transactions.
15. Recover the catalog and directory. The RECOVER function includes: RECOVER TABLESPACE, RECOVER INDEX, or REBUILD INDEX. If you have an image copy of an index, use RECOVER INDEX. If you do not have an image copy of an index, use REBUILD INDEX to reconstruct the index from the recovered table space.
 - a. Recover DSNDB01.SYSUTILX. This must be a separate job step.
 - b. Recover all indexes on SYSUTILX. This must be a separate job step.
 - c. Determine whether a utility was running at the time the latest archive log was created by entering the DISPLAY UTILITY(*) command, and record the name and current phase of any utility that is running. (You cannot restart a utility at the recovery site that was interrupted at the disaster site. You must use the TERM UTILITY command to terminate it. Use the TERM UTILITY command on a utility that is operating on any object except DSNDB01.SYSUTILX.)

- d. Run the DIAGNOSE utility with the DISPLAY SYSUTILX option. The output consists of information about each active utility, including the table space name (in most cases). This is the only way to correlate the object name with the utility. Message DSNU866I gives information about the utility, and DSNU867I gives the database and table space name in USUDBNAM and USUSPNAM, respectively.
 - e. Use the TERM UTILITY command to terminate any utilities that are in progress on catalog or directory table spaces.
 - f. Recover the rest of the catalog and directory objects, starting with DBD01, in the order shown in the description of the RECOVER utility. (See Recovering catalog and directory objects in DB2 Utility Guide and Reference.)
16. Define and initialize the work file database
 - a. Define temporary work files. Use installation job DSNTIJTM as a model.
 - b. Issue the command START DATABASE(*work-file-database*) to start the work file database.
 17. Use any method that you want to verify the integrity of the DB2 catalog and directory. Use the catalog queries in member DSNTESQ of data set DSN910.SDSNSAMP after the work file database is defined and initialized.
 18. If you use data definition control support, recover the objects in the data definition control support database.
 19. If you use the resource limit facility, recover the objects in the resource limit control facility database.
 20. Modify DSNZPxxx to restart all databases on each member of the data sharing group:
 - a. Run the DSNTINST CLIST in UPDATE mode.
 - b. From panel DSNTIPB, select DATABASES TO START AUTOMATICALLY. Panel DSNTIPS opens. Type RESTART in the first field and ALL in the second field, and press **Enter**. You are returned to DSNTIPB.
 - c. Reassemble DSNZPxxx by using job DSNTIJUZ (produced by the CLIST started in step 4 on page 685).
 21. Stop DB2.
 22. Start DB2.
 23. Make a full image copy of the catalog and directory.
 24. Recover user table spaces and index spaces. If utilities were running on any table spaces or index spaces, see “What to do about utilities that were in progress at time of failure” on page 691. You cannot restart a utility at the recovery site that was interrupted at the disaster site. Use the TERM UTILITY command to terminate any utilities that are running against user table spaces or index spaces.
 - a. To determine which, if any, of your table spaces or index spaces are user-managed, perform the following queries for table spaces and index spaces.
 - Table spaces:
SELECT * FROM SYSIBM.SYSTABLEPART WHERE STORYPE='E';
 - Index spaces:
SELECT * FROM SYSIBM.SYSINDEXPART WHERE STORYPE='E';
 To allocate user-managed table spaces or index spaces, use the access method services DEFINE CLUSTER command. To find the correct IPREFIX for the DEFINE CLUSTER command, perform the following queries for table spaces and index spaces.

- Table spaces:

```
SELECT DBNAME, TSNAME, PARTITION, IPREFIX FROM SYSIBM.SYSTABLEPART
WHERE DBNAME=dbname AND TSNAME=tsname
ORDER BY PARTITION;
```

- Index spaces:

```
SELECT IXNAME, PARTITION, IPREFIX FROM SYSIBM.SYSINDEXPART
WHERE IXCREATOR=ixcreator AND IXNAME=ixname
ORDER BY PARTITION;
```

Now you can perform the DEFINE CLUSTER command with the correct IPREFIX (I or J) in the data set name:

```
catname.DSNDBx.dbname.psname.y0001.znnn
```

The *y* can be either I or J, *x* is C (for VSAM clusters) or D (for VSAM data components), and *sname* is either the table space or index space name.

- If your user table spaces or index spaces are STOGROUP-defined, and if the volume serial numbers at the recovery site are different from those at the local site, use the SQL statement ALTER STOGROUP to change them in the DB2 catalog.
 - Recover all user table spaces and index spaces from the appropriate image copies. If you do not copy your indexes, use the REBUILD INDEX utility to reconstruct the indexes.
 - Start all user table spaces and index spaces for read-write processing by issuing the command START DATABASE with the ACCESS(RW) option.
 - Resolve any remaining CHECK-pending states that would prevent COPY execution.
 - Run queries for which the results are known.
- Make full image copies of all table spaces and indexes with the COPY YES attribute.
 - Finally, compensate for work that was lost since the last archive was created by rerunning online transactions and batch jobs.

Determine what to do about any utilities that were in progress at the time of failure.

Related concepts

"What to do about utilities that were in progress at time of failure"

"Preparations for disaster recovery" on page 545

"Migration step 1: Actions that you must perform before migration" (DB2 Installation Guide)

Related tasks

"Recovering data" (DB2 Data Sharing: Planning and Administration)

"Invoking the CLIST" (DB2 Installation Guide and Reference)

Related reference

"DSN1LOGP" (DB2 Utility Guide and Reference)

What to do about utilities that were in progress at time of failure

After you restore data from image copies and archives, you need to determine what to do about any utilities that were in progress at the time of the failure. If any utility jobs were running after the last time that the log was offloaded before the disaster, you might need to take some additional steps.

After restarting DB2, only certain utilities need to be terminated with the TERM UTILITY command.

Allowing the RECOVER utility to reset pending states is preferable. However, you might occasionally need to use the REPAIR utility to reset them. Do not start the table space with ACCESS(FORCE) because FORCE resets any page set exception conditions described in “Database page set controls.”

For the following utility jobs, perform the indicated actions:

CHECK DATA

Terminate the utility, and run it again after recovery is complete.

COPY After you enter the TERM UTILITY command, DB2 places a record in the SYSCOPY catalog table to indicate that the COPY utility job was terminated. This makes it necessary for you to make a full image copy. When you copy your environment at the completion of the disaster recovery scenario, you fulfill that requirement.

LOAD

Find the options that you specified in the following table, and perform the specified actions.

Table 109. Actions to take when a LOAD utility job is interrupted

LOAD options specified	What to do
LOG YES	If the RELOAD phase completed, recover to the current time. Recover the indexes. If the RELOAD phase did not complete, recover to a prior point in time. The SYSCOPY record that is inserted at the beginning of the RELOAD phase contains the RBA or LRSN.
LOG NO and <i>copy-spec</i>	If the RELOAD phase completed, the table space is complete after you recover it to the current time. Recover the indexes. If the RELOAD phase did not complete, recover the table space to a prior point in time. Recover the indexes.
LOG NO, <i>copy-spec</i> , and SORTKEYS <i>integer</i> ¹	If the BUILD or SORTBLD phase completed, recover to the current time, and recover the indexes. If the BUILD or SORTBLD phase did not complete, recover to a prior point in time. Recover the indexes.
LOG NO	Recover the table space to a prior point in time. You can use the TOCOPY option of the RECOVER utility to do this.

Note:

1. You must specify a value that is greater than zero for *integer*. If you specify zero for *integer*, the SORTKEYS option does not apply.

To avoid extra loss of data in a future disaster situation, run the QUIESCE utility on table spaces before invoking the LOAD utility. This enables you to recover a table space by using the TOLOGPOINT option instead of TOCOPY.

REORG

For a user table space, find the options that you specified in the following table, and perform the specified actions.

Recommendation: Make full image copies of the catalog and directory before you run REORG on them.

Table 110. Actions to take when the REORG utility is interrupted

REORG options specified	What to do
LOG YES	<p>If the RELOAD phase completed, recover to the current time. Recover the indexes.</p> <p>If the RELOAD phase did not complete, recover to the current time to restore the table space to the point before the REORG job began. Recover the indexes.</p>
LOG NO	<p>If the build or SORTBLD phase completed, recover to the current time, and recover the indexes.</p> <p>If the build or SORTBLD phase did not complete, recover to the current time to restore the table space to the point before the REORG job began. Recover the indexes.</p>
SHRLEVEL CHANGE or SHRLEVEL REFERENCE	<p>If the SWITCH phase completed, terminate the utility. Recover the table space to the current time. Recover the indexes.</p> <p>If the SWITCH phase did not complete, recover the table space to the current time. Recover the indexes.</p>

For a catalog or directory table space, the instructions are somewhat different. For those table spaces that were using online REORG, find the options that you specified in the preceding table, and perform the specified actions.

If you have no image copies from immediately before REORG failed, use this procedure:

1. From your DISPLAY UTILITY command and DIAGNOSE utility output, determine what phase the REORG job was in and which table space it was reorganizing when the disaster occurred.
2. Run the RECOVER utility on the catalog and directory in the correct order. Recover all table spaces to the current time, except the table space that was being reorganized at the time of the disaster. If the RELOAD phase of the REORG job on that table space had not completed when the disaster occurred, recover the table space to the current time. Because REORG does not generate any log records prior to the RELOAD phase for catalog and directory objects, a recovery to the current time restores the data to the state that it was in before the REORG job. If the RELOAD phase completed, perform the following actions:
 - a. Run the DSN1LOGP utility against the archive log data sets from the disaster site.
 - b. Find the begin-UR log record for the REORG job that failed in the DSN1LOGP output.
 - c. Run the RECOVER utility with the TOLOGPOINT option on the table space that was being reorganized. Use the URID of the begin-UR record as the TOLOGPOINT value.
3. Recover or rebuild all indexes.

If you have image copies from immediately before the REORG job failed, run the RECOVER utility with the TOCOPY option to recover the catalog and directory, in the correct order.

Related tasks

"Recovering catalog and directory objects" (DB2 Utility Guide and Reference)

Recovering from disasters by using a tracker site

You can use a tracker site for disaster recovery. A DB2 *tracker site* is a separate DB2 subsystem or data sharing group that exists solely for the purpose of keeping shadow copies of the data at your primary site.

Using a tracker site for disaster recovery is somewhat similar to other methods.

Recommendation: Test and document a disaster procedure that is customized for your location.

From the primary site, you transfer the BSDS and the archive logs, and that tracker site runs periodic LOGONLY recoveries to keep the shadow data up-to-date. If a disaster occurs at the primary site, the tracker site becomes the *takeover* site. Because the tracker site has been shadowing the activity on the primary site, you do not need to constantly ship image copies; the takeover time for the tracker site might be faster because DB2 recovery does not need to use image copies.

Characteristics of a tracker site

A DB2 *tracker site* is a separate DB2 subsystem or data sharing group that exists solely for the purpose of keeping shadow copies of the data at your primary site.

Because the tracker site must use only the primary site logs for recovery, you must not update the catalog and directory or the data at the tracker site. The DB2 subsystem at the tracker site disallows updates.

- The following SQL statements are not allowed at a tracker site:
 - GRANT or REVOKE
 - DROP, ALTER, or CREATE
 - UPDATE, INSERT, or DELETE

Dynamic read-only SELECT statements are allowed, but not recommended. At the end of each tracker site recovery cycle, databases might contain uncommitted data, and indexes might be inconsistent with the data at the tracker site.

- The only online utilities that are allowed are REPORT, DIAGNOSE, RECOVER, REBUILD, and RESTORE SYSTEM LOGONLY. Recovery to a prior point in time is not allowed.
- BIND is not allowed.
- TERM UTIL is not allowed for LOAD, REORG, REPAIR, and COPY.
- The START DATABASE command is not allowed when LPL or GRECP status exists for the object of the command. Use of the START DATABASE command is not necessary to clear LPL or GRECP conditions because you are going to be running RECOVER jobs that clear the conditions.
- The START DATABASE command with ACCESS(FORCE) is not allowed.
- Down-level detection is disabled.
- Log archiving is disabled.
- Real-time statistics are disabled.

Setting up a tracker site

To set up a tracker site for disaster recovery purposes, you create a mirror image of your primary DB2 subsystem and then take steps so that the tracker site continues to be synchronized with the primary site.

To set up the tracker site:

1. Create a mirror image of your primary DB2 subsystem or data sharing group. This process is described in steps 1 through 4 of the normal disaster recovery procedure, which includes creating catalogs and restoring DB2 libraries.
2. Modify the subsystem parameters as follows:
 - Set the TRKSITE subsystem parameter to YES.
 - Optionally, set the SITETYP parameter to RECOVERYSITE if the full image copies that this site is to receive are created as remote site copies.
3. Use the access method services DEFINE CLUSTER command to allocate data sets for all user-managed table spaces that you plan to send over from the primary site.
4. Optional: Allocate data sets for user-managed indexes that you want to rebuild during recovery cycles. The main reason that you rebuild indexes during recovery cycles is for running efficient queries on the tracker site. If you do not require indexes, you do not need to rebuild them for recovery cycles. For nonpartitioning indexes on very large tables, you can include indexes for LOGONLY recovery during the recovery cycle, which can reduce the amount of time that it takes to bring up the disaster site. Be sure that you define data sets with the proper prefix (either I or J) for both indexes and table spaces.
5. Send full image copies of all DB2 data at the primary site to the tracker site. Optionally, you can use the BACKUP SYSTEM utility with the DATA ONLY option and send copies of the database copy pool to the tracker site. If you send copies that the BACKUP SYSTEM utility creates, this step completes the tracker site setup procedure.
6. If you did not use the BACKUP SYSTEM utility in the prior, tailor installation job DSNTIJIN to create DB2 catalog data sets.

Important: Do not attempt to start the tracker site when you are setting it up. You must follow the procedure described in “Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY.”

Related reference

“REORG INDEX” (DB2 Utility Guide and Reference)

“REORG TABLESPACE” (DB2 Utility Guide and Reference)

Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY

Each time that you restore the logs and the BSDS from the primary site at your tracker site, you establish a new recovery cycle. One way to establish a recovery cycle is to use the RESTORE SYSTEM utility with the LOGONLY option.

Full image copies of all the data at the primary site must be available at the tracker site.

Using the LOGONLY option of the RESTORE SYSTEM utility enables you to periodically apply the active log, archive logs, and the BSDS from the primary site at the tracker site.

To establish a recovery cycle at your tracker site by using the RESTORE SYSTEM utility:

1. While your primary site continues its usual workload, send a copy of the primary site active log, archive logs, and BSDS to the tracker site. Send full image copies for the following objects:
 - Table spaces or partitions that are reorganized, loaded, or repaired with the LOG NO option after the latest recovery cycle

- Objects that, after the latest recovery cycle, have been recovered to a point in time

Recommendation: If you are taking incremental image copies, run the MERGECOPY utility at the primary site before sending the copy to the tracker site.

2. At the tracker site, restore the BSDS that was received from the primary site by following these steps:
 - a. Locate the BSDS in the latest archive log that is now at the tracker site.
 - b. Register this archive log in the archive log inventory of the new BSDS by using the change log inventory utility (DSNJU003).
 - c. Register the primary site active log in the new BSDS by using the change log inventory utility (DSNJU003).
3. Use the change log inventory utility (DSNJU003) with the following CRESTART control statement:


```
CRESTART CREATE, ENDRBA=nnnnnnnn000, FORWARD=NO, BACKOUT=NO
```

In this control statement, *nnnnnnnnnn* equals the RBA at which the latest archive log record ends +1. Do not specify the RBA at which the archive log begins because you cannot cold start or skip logs in tracker mode.

Data sharing

If you are recovering a data sharing group, you must use the following CRESTART control statement on all members of the data sharing group. The ENDLRSN value must be the same for all members.

```
CRESTART CREATE, ENDLRSN=nnnnnnnnnnnn, FORWARD=NO, BACKOUT=NO
```

In this control statement, *nnnnnnnnnnnn* is the lowest LRSN of all the members that are to be read during restart. Specify one of the following values for the ENDLRSN:

- If you receive the ENDLRSN from the output of the print log map utility (DSNJU004) or from the console logs using message DSNJ003I, you must use ENDLRSN -1 as the input to the conditional restart.
- If you receive the ENDLRSN from the output of the DSN1LOGP utility (message DSN1213I), you can use the displayed value.

The ENDLRSN or ENDRBA value indicates the end log point for data recovery and for truncating the archive log. With ENDLRSN, the missing log records between the lowest and highest ENDLRSN values for all the members are applied during the next recovery cycle.

4. If the tracker site is a data sharing group, delete all DB2 coupling facility structures before restarting the tracker members.
5. If you used the DSN1COPY utility to create a copy of SYSUTILX during the last tracker cycle, restore this copy with DSN1COPY.

Data sharing

For data sharing, restart every member of the data sharing group.

6. At the tracker site, restart DB2 to begin a tracker site recovery cycle.
7. At the tracker site, run the RESTORE SYSTEM utility with the LOGONLY option to apply the logs (both archive and active) to the data at the tracker site.

8. If the RESTORE SYSTEM utility issues a return code of 4, use the DSN1COPY utility to make a copy of SYSUTILX and of indexes that are associated with SYSUTILX before you recover or rebuild those objects. DSN1COPY issues a return code of 4 if application of the log results in one or more DB2 objects being marked as RECP or RBDP.
9. Restart DB2 at the tracker site.
10. Issue the DISPLAY DATABASE RESTRICT command to display objects that are marked RECP, RBDP, or LPL and to identify which objects are in a utility progress state (such as UTUT or UTRO). Run the RECOVER or REBUILD INDEX utility on these objects, or record which objects are in an exception state so that you can recover them at a later time. The exception states of these objects are not retained in the next recovery cycle.
11. After all recovery activity complete at the tracker site, shut down the DB2 tracker site.
12. Optional: Stop and start the DB2 tracker site several times before completing a recovery cycle.

Related concepts

“Media failures during LOGONLY recovery” on page 699

Related tasks

“Establishing a recovery cycle by using the RECOVER utility”

“Restoring data from image copies and archive logs” on page 679

Establishing a recovery cycle by using the RECOVER utility

Each time that you restore the logs and the BSDS from the primary site at your tracker site, you establish a new recovery cycle. One way to establish a recovery cycle is to use the RECOVER utility.

To establish a recovery cycle by using the RECOVER utility:

1. While your primary site continues its usual workload, send a copy of the primary site active log, archive logs, and BSDS to the tracker site. Send full image copies for the following objects:
 - Table spaces or partitions that are reorganized, loaded, or repaired with the LOG NO option after the latest recovery cycle.
 - Objects that, after the latest recovery cycle, have been recovered to a point in time.
 - SYSUTILX. Send a full image copy to DSNDB01.SYSUTILX for normal (full image copy and log) recoveries. For LOGONLY recoveries, create a copy of DSNDB01.SYSUTILX by using the DSN1COPY utility.

DB2 does not write SYSLGRNX entries for DSNDB01.SYSUTILX, which can lead to long recovery times at the tracker site. In addition, SYSUTILX and its indexes are updated during the tracker cycle when you run your recoveries. Because SYSUTILX must remain consistent with the SYSUTILX at the primary site, discard the tracker cycle updates before the next tracker cycle.

Recommendation: If you are taking incremental image copies, run the MERGECOPY utility at the primary site before sending the copy to the tracker site.

2. At the tracker site, restore the BSDS that was received from the primary site by using one of the following methods:
 - Locate the BSDS in the latest archive log that is now at the tracker site.
 - Register this archive log in the archive log inventory of the new BSDS by using the change log inventory utility (DSNJU003).

- Register the primary site active log in the new BSDS by using the change log inventory utility (DSNJU003).
3. Use the change log inventory utility (DSNJU003) with the following CRESTART control statement:

```
CRESTART CREATE,ENDRBA=nnnnnnnn000, FORWARD=NO,BACKOUT=NO
```

In this control statement, *nnnnnnnnn000* equals the value of the ENDRBA of the latest archive log plus 1. Do not specify STARTRBA because you cannot cold start or skip logs in a tracker system.

Data sharing

If you are recovering a data sharing group, you must use the following CRESTART control statement on all members of the data sharing group. The ENDLRSN value must be the same for all members.

```
CRESTART CREATE,ENDLRSN=nnnnnnnnnnn, FORWARD=NO,BACKOUT=NO
```

In this control statement, *nnnnnnnnnnnn* is the lowest ENDLRSN of all the members that are to be read during restart. Specify one of the following values for the ENDLRSN:

- If you receive the ENDLRSN from the output of the print log map utility (DSNJU004) or from message DSNJ003I at the console logs use ENDLRSN -1 as the input to the conditional restart.
- If you receive the ENDLRSN from the output of the DSN1LOGP utility (DSN1213I message), use the displayed value.

The ENDLRSN or ENDRBA value indicates the end log point for data recovery and for truncating the archive log. With ENDLRSN, the missing log records between the lowest and highest ENDLRSN values for all the members are applied during the next recovery cycle.

4. If the tracker site is a data sharing group, delete all DB2 coupling facility structures before restarting the tracker members.
5. At the tracker site, restart DB2 to begin a tracker site recovery cycle.

Data sharing

For data sharing, restart every member of the data sharing group.

6. At the tracker site, submit RECOVER utility jobs to recover database objects. Run the RECOVER utility with the LOGONLY option on all database objects that do not require recovery from an image copy.

You must recover database objects as the following procedure specifies:

- a. Restore the full image copy or DSN1COPY of SYSUTILX.

If you are doing a LOGONLY recovery on SYSUTILX from a previous DSN1COPY backup, make another DSN1COPY copy of that table space after the LOGONLY recovery is complete and before recovering any other catalog or directory objects.

After you recover SYSUTILX and either recover or rebuild its indexes, and before you recover other system and user table spaces, determine what utilities were running at the primary site.

- b. Recover the catalog and directory in the correct order.

If you have user-defined catalog indexes, rebuilding them is optional until the tracker DB2 site becomes the takeover DB2 site. (You might want to rebuild them sooner if you require them for catalog query performance.) However, if you are recovering user-defined catalog indexes, do the recovery in this step.

- c. If needed, recover other system data such as the data definition control support table spaces and the resource limit facility table spaces.
- d. Recover user data and, optionally, rebuild your indexes.

You do not need to rebuild indexes unless you intend to run dynamic queries on the data at the tracker site.

For a tracker site, DB2 stores the conditional restart ENDRBA or ENDLRSN in the page set after each recovery completes successfully. By storing the log truncation value in the page set, DB2 ensures that it does not skip any log records between recovery cycles.

7. Issue the DISPLAY UTILITY(*) command for a list of currently running utilities.
8. Run the DIAGNOSE utility with the DISPLAY SYSUTIL statement to determine the names of the object on which the utilities are running. Installation SYSOPR authority is required.
9. Perform the following actions for objects at the tracker site on which utilities are pending. Restrictions apply to these objects because DB2 prevents you from using the TERM UTILITY command to remove pending statuses at a tracker site.
 - If a LOAD, REORG, REPAIR, or COPY utility job is in progress on any catalog or directory object at the primary site, shut down DB2 subsystem. You cannot continue recovering by using the list of catalog and directory objects. Therefore, you cannot recover any user data. At the next recovery cycle, send a full image copy of the object from the primary site. At the tracker site, use the RECOVER utility to restore the object.
 - If a LOAD, REORG, REPAIR, or COPY utility job is in progress on any user data, at the next recovery cycle, send a full image copy of the object from the primary site. At the tracker site, use the RECOVER utility to restore the object.
 - If an object is in the restart-pending state, use LOGONLY recovery to recover the object when that object is no longer in a restart-pending state.

Data sharing

If read/write shared data (GPB-dependent data) is in the advisory recovery pending state, the tracker DB2 site performs recovery processing. Because the tracker DB2 site always performs a conditional restart, the postponed indoubt units of recovery are not recognized after the tracker DB2 site restarts.

10. After all recovery has completed at the tracker site, shut down the tracker DB2 site. This is the end of the tracker site recovery cycle.
11. Optional: Stop and start the tracker DB2 site several times before completing a recovery cycle.

Related concepts

“Media failures during LOGONLY recovery”

Related tasks

“Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY” on page 695

“Restoring data from image copies and archive logs” on page 679

Media failures during LOGONLY recovery

If an I/O error occurs during a LOGONLY recovery, you can recover the object by using the image copies and logs after you correct the media failure.

If an entire volume is corrupted and you are using DB2 storage groups, you cannot use the ALTER STOGROUP statement to remove the corrupted volume and add another. (This is possible, however, for a non-tracker system.) Instead, you must remove the corrupted volume and re-initialize another volume with the same volume serial number before you invoke the RECOVER utility for all table spaces and indexes on that volume.

Maintaining a tracker site

If you want to have a tracker site for possible disaster recovery needs, you need to maintain it so that it can operate as you need it to.

To maintain a tracker site:

1. Keep the tracker site and primary site be at the same maintenance level to avoid unexpected problems.
2. Between recovery cycles, apply maintenance as you normally do, by stopping and restarting the DB2 subsystem or a DB2 data sharing member.
3. If a tracker site fails, restart it as you normally do.
4. Save your complete tracker site prior to testing a takeover site. This step is necessary because bringing up a tracker site as the takeover site destroys the tracker site environment. After testing the takeover site, you can restore the tracker site and resume the recovery cycles.

When restarting a data sharing group, the first member that starts during a recovery cycle puts the ENDLRSN value in the shared communications area (SCA) of the coupling facility. If an SCA failure occurs during a recovery cycle, you must go through the recovery cycle again, using the same ENDLRSN value for your conditional restart.

Making the tracker site the takeover site

If a disaster occurs at the primary site, the tracker site must become the takeover site.

Save your complete tracker site prior to testing a takeover site.

To make the tracker site the takeover site:

1. Restart the takeover site.
2. Apply log data or image copies that were enroute when the disaster occurred.
3. Follow the appropriate procedure for making the tracker site a takeover site, depending on whether you use RESTORE SYSTEM LOGONLY or the RECOVER utility in your tracker site recovery cycles.

Related tasks

“Maintaining a tracker site”

Recovering at a tracker site that uses RESTORE SYSTEM LOGONLY:

One way that you can make the tracker site the takeover site is by using the RESTORE SYSTEM utility with the LOGONLY option in the recovery cycles at the tracker site.

To use RESTORE SYSTEM LOGONLY to make the tracker site the takeover site, complete the following steps:

1. If log data for a recovery cycle is enroute or is available but has not yet been used in a recovery cycle, perform the procedure in “Establishing a recovery cycle by using RESTORE SYSTEM LOGONLY” on page 695.

2. Ensure that the TRKSITE NO subsystem parameter is specified.
3. For scenarios other than data sharing, continue with step 4.

Data sharing

If this is a data sharing system, delete the coupling facility structures.

4. Start DB2 at the same RBA or ENDLSRN that you used in the most recent tracker site recovery cycle. Specify FORWARD=YES and BACKOUT=YES in the CRESTART statement; this takes care of uncommitted work.
5. Restart the objects that are in GRECP or LPL status by issuing the START DATABASE(*) SPACENAM(*) command.
6. If you used the DSN1COPY utility to create a copy of SYSUTILX in the last recovery cycle, use DSN1COPY to restore that copy.
7. Terminate any in-progress utilities by using the following procedure:
 - a. Enter the DISPLAY UTILITY(*) command .
 - b. Run the DIAGNOSE utility with DISPLAY SYSUTIL to get the names of objects on which utilities are being run.
 - c. Terminate in-progress utilities in the correct order by using the TERM UTILITY(*) command.
8. Rebuild indexes, including IBM and user-defined indexes on the DB2 catalog and user-defined indexes on table spaces.

Related tasks

“Recovering at a tracker site that uses the RECOVER utility”

“Restoring data from image copies and archive logs” on page 679

Recovering at a tracker site that uses the RECOVER utility:

One way that you can make the tracker site the takeover site is by using the RECOVER utility in the recovery cycles at your tracker site.

To make the tracker site the takeover site by using the RECOVER utility:

1. Restore the BSDS, and register the archive log from the last archive log that you received from the primary site.
2. For environments that do not use data sharing, continue with step 3.

Data sharing

If this is a data sharing system, delete the coupling facility structures.

3. Ensure that the DEFER ALL and TRKSITE NO subsystem parameters are specified.
4. Take the appropriate action, which depends on whether you received more logs from the primary site. If this is a non-data-sharing DB2 subsystem, the log truncation point varies depending on whether you have received more logs from the primary site since the last recovery cycle:
 - If you did not receive more logs from the primary site:

Start DB2 using the same ENDRBA that you used on the last tracker cycle. Specify FORWARD=YES and BACKOUT=YES; this takes care of uncommitted work. If you have fully recovered the objects during the previous cycle, they are current except for any objects that had outstanding units of recovery during restart. Because the previous cycle specified NO for both FORWARD and BACKOUT and you have now specified YES, affected data sets are placed in the LPL. Restart the objects that are in LPL status by using the following command:

```
START DATABASE(*) SPACENAM(*)
```

After you issue the command, all table spaces and indexes that were previously recovered are now current. Remember to rebuild any indexes that were not recovered during the previous tracker cycle, including user-defined indexes on the DB2 catalog.

- If you received more logs from the primary site:

Start DB2 using the truncated RBA *nnnnnnnnnn000*, which equals the value of the ENDRBA of the latest archive log plus 1. Specify FORWARD=YES and BACKOUT=YES. Run your recoveries as you did during recovery cycles.

Data sharing

You must restart every member of the data sharing group; use the following CRESTART statement:

```
CRESTART CREATE, ENDLRSN=nnnnnnnnnnnn, FORWARD=YES, BACKOUT=YES
```

In this statement, *nnnnnnnnnnnn* is the LRSN of the last log record that is to be used during restart. Specify one of the following values for the ENDLRSN:

- If you receive the ENDLRSN from the output of the print log map utility (DSNJU004) or from message DSNJ003I at the console logs use ENDLRSN -1 as the input to the conditional restart.
- If you receive the ENDLRSN from the output of the DSN1LOGP utility (DSN1213I message), use the displayed value.

The ENDLRSN or ENDRBA value indicates the end log point for data recovery and for truncating the archive log. With ENDLRSN, the missing log records between the lowest and highest ENDLRSN values for all the members are applied during the next recovery cycle.

The takeover DB2 sites must specify conditional restart with a common ENDLRSN value to allow all remote members to logically truncate the logs at a consistent point.

5. As described for a tracker recovery cycle, recover SYSUTILX from an image copy from the primary site, or from a previous DSN1COPY copy that was taken at the tracker site.
6. Terminate any in-progress utilities by using the following procedure:
 - a. Enter the command DISPLAY UTILITY(*).
 - b. Run the DIAGNOSE utility with DISPLAY SYSUTIL to get the names of objects on which utilities are being run.
 - c. Terminate in-progress utilities by using the command TERM UTILITY(*)
7. Continue with your recoveries either with the LOGONLY option or with image copies. Remember to rebuild indexes, including IBM and user-defined indexes on the DB2 catalog and user-defined indexes on table spaces.

Related tasks

“Recovering at a tracker site that uses RESTORE SYSTEM LOGONLY” on page 700

“Restoring data from image copies and archive logs” on page 679

Using data mirroring for disaster recovery

Data mirroring is the automatic replication of current data from your primary site to a secondary site. You can use this secondary site for your recovery site to recover after a disaster without the need to restore DB2 image copies or apply DB2 logs to bring DB2 data to the current point in time.

The procedures for data mirroring are intended for environments that mirror an entire DB2 subsystem or data sharing group, which includes the catalog, directory, user data, BSDS, and active logs. You must mirror all volumes in such a way that they terminate at exactly the same point. You can achieve this final condition by using consistency groups.

Follow the appropriate procedure for recovering from a disaster by using data mirroring.

Role of data mirroring in recovery from a rolling disaster

In a real disaster, your local site gradually and intermittently fails over a number of seconds. This kind of DB2 failure is known as a *rolling disaster*. You can recover from a rolling disaster by using data mirroring.

To use data mirroring for disaster recovery, you must mirror data from your local site with a method that does not reproduce a rolling disaster at your recovery site. To recover a DB2 subsystem and data with data integrity, you must use volumes that end at a consistent point in time for each DB2 subsystem or data sharing group. Mirroring a rolling disaster causes volumes at your recovery site to end over a span of time rather than at one single point.

The following figure shows how a rolling disaster can cause data to become inconsistent between two subsystems.

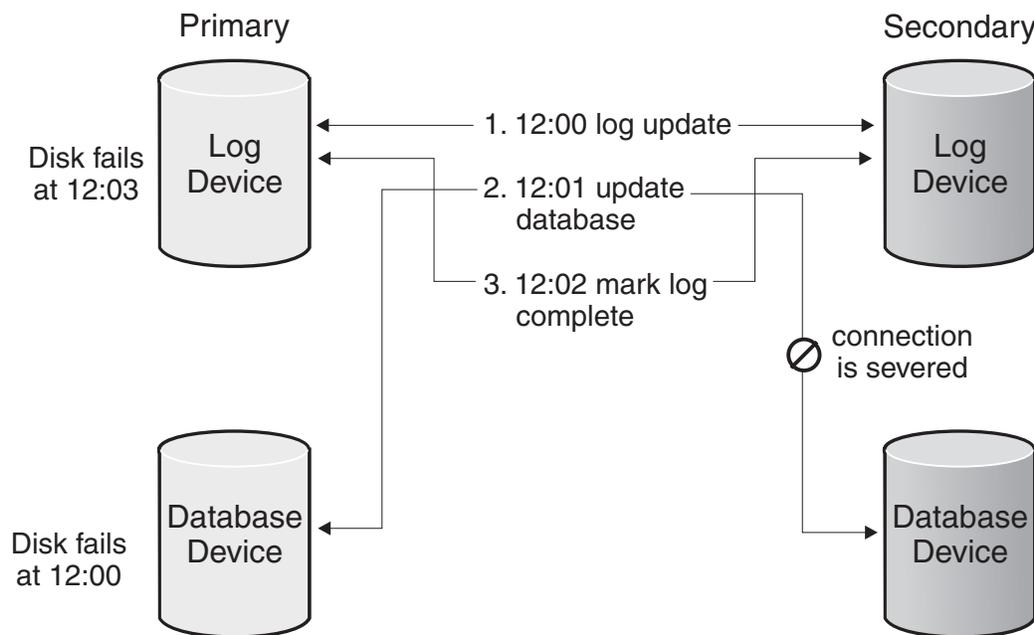


Figure 68. Data inconsistency caused by a rolling disaster

Example: In a rolling disaster, the following events at the primary site cause data inconsistency at your recovery site. This data inconsistency example follows the same scenario that the preceding figure depicts.

1. Some time prior to 12:00: A table space is updated in the buffer pool.
2. 12:00: The log record is written to disk on logical storage subsystem 1.
3. 12:01: Logical storage subsystem 2 fails.
4. 12:02: The update to the table space is externalized to logical storage subsystem 2 but is not written because subsystem 2 failed.

5. 12:03: The log record that indicates that the table space update was made is written to disk on logical storage subsystem 1.
6. 12:03: Logical storage subsystem 1 fails.

Because the logical storage subsystems do not fail at the same point in time, they contain inconsistent data. In this scenario, the log indicates that the update is applied to the table space, but the update is not applied to the data volume that holds this table space.

Important: Any disaster recovery solution that uses data mirroring must guarantee that all volumes at the recovery site contain data for the same point in time.

Role of consistency groups in recovery

Generally a *consistency group* is a collection of volumes that contain consistent, related data. Consistency groups play an important role in DB2 recovery.

A consistency group, which is a collection of related data, can span logical storage subsystems and disk subsystems. For DB2 specifically, a consistency group contains an entire DB2 subsystem or an entire DB2 data sharing group.

The following DB2 elements comprise a consistency group:

- Catalog tables
- Directory tables
- BSDS
- Logs
- All user data
- ICF catalogs

Additionally, all objects within a consistency group must represent the same point in time in at least one of the following situations:

- At the time of a backup
- After a normal DB2 restart

You can use the following methods to create consistency groups:

- XRC I/O timestamping and system data mover
- FlashCopy consistency groups
- GDPSfreeze policies
- The DB2 SET LOG SUSPEND command
- The DB2 BACKUP SYSTEM utility

When a rolling disaster strikes your primary site, consistency groups guarantee that all volumes at the recovery site contain data for the same point in time. In a data mirroring environment, you must perform both of the following actions for each consistency group that you maintain:

- Mirror data to the secondary volumes in the same sequence that DB2 writes data to the primary volumes.

In many processing situations, DB2 must complete one write operation before it begins another write operation on a different disk group or a different storage server. A write operation that depends on a previous write operation is called a *dependent write*. Do not mirror a dependent write if you have not mirrored the

write operation on which the dependent write depends. If you mirror data out of sequence, your recovery site will contain inconsistent data that you cannot use for disaster recovery.

- Temporarily suspend and queue write operations to create a group point of consistency when an error occurs between any pair of primary and secondary volumes.

When an error occurs that prevents the update of a secondary volume in a single-volume pair, this error might mark the beginning of a rolling disaster. To prevent your secondary site from mirroring a rolling disaster, you must suspend and queue data mirroring by taking the following steps after a write error between any pairs:

1. Suspend and queue all write operations in the volume pair that experiences a write error.
2. Invoke automation that temporarily suspends and queues data mirroring to all your secondary volumes.
3. Save data at the secondary site at a point of consistency.
4. If a rolling disaster does not strike your primary site, resume normal data mirroring after some amount of time that you define. If a rolling disaster does strike your primary site, follow the recovery procedure in “Recovering in a data mirroring environment.”

Recovering in a data mirroring environment

In a data mirroring environment, you can use a general procedure to recover at your secondary site from a disaster at your primary site.

This procedure applies to all DB2 data mirroring scenarios except those that use Extended Remote Copy (XRC). This general procedure is valid only if you have established and maintained consistency groups before the disaster struck the primary site. If you use data mirroring to recover, you must recover your entire DB2 subsystem or data sharing group with data mirroring.

You do not need to restore DB2 image copies or apply DB2 logs to bring DB2 data to the current point in time when you use data mirroring. However, you might need image copies at the recovery site if the LOAD or RECOVER utility was active at the time of the disaster.

To recover at the secondary site after a disaster:

1. At your recovery site, IPL all z/OS images that correspond to the z/OS images that you lost at your primary site.
2. For environments that do not use data sharing, continue with step 3 on page 706.

Data sharing

For data sharing groups, you must remove old information from the coupling facility.

- a. Enter the following z/OS command to display the structures for this data sharing group:
`D XCF,STRUCTURE,STRNAME=grpname*`
- b. For group buffer pools and the lock structure, enter the following command to force off the connections in those structures:
`SETXCF FORCE,CONNECTION,STRNAME=strname,CONNAME=ALL`
- c. Delete all the DB2 coupling facility structures by using the following command for each structure:

SETXCF FORCE,STRUCTURE,STRNAME=*strname*

3. If you are using the distributed data facility, set LOCATION and LUNAME in the BSDS to values that are specific to your new primary site. To set LOCATION and LUNAME, run the stand-alone change log inventory utility (DSNJU003) with the following control statement:

```
DDF LOCATION=locname, LUNAME=luname
```

4. Start all DB2 members by using local DSNZPARM data sets and perform a normal restart.

Data sharing

For data sharing groups, DB2 performs group restart. Shared data sets are set to GRECP (group buffer pool RECOVER-pending) status, and pages are added to the LPL (logical page list).

5. For scenarios other than data sharing, continue to step 6.

Data sharing

For data sharing groups, perform the following procedure:

- a. Display all data sets with GRECP or LPL status with the following DB2 command:

```
-DISPLAY DATABASE(*) SPACENAM(*) RESTRICT(GRECP, LPL) LIMIT(*)
```

Record the output that this command generates.

- b. Start the DB2 directory with the following DB2 command:

```
-START DATABASE(DSNDB01) SPACENAM(*)
```

- c. Start the DB2 catalog with the following DB2 command:

```
-START DATABASE(DSNDB06) SPACENAM(*)
```

6. Use the following DB2 command to display all utilities that the failure interrupted:

```
-DISPLAY UTILITY(*)
```

If utilities are pending, record the output from this command, and continue to the next step. You cannot restart utilities at a recovery site. You will terminate these utilities in step 8. If no utilities are pending, continue to step number 9.

7. Use the DIAGNOSE utility to access the SYSUTILX directory table. You cannot access this directory table by using normal SQL statements (as you can with most other directory tables). You can access SYSUTILX only by using the DIAGNOSE utility, which is normally intended to be used under the direction of IBM Software Support.

Use the following control statement to run the DIAGNOSE utility job:

```
DIAGNOSE DISPLAY SYSUTILX
```

To stop the utility, issue this control statement:

```
END DIAGNOSE
```

Examine the output. Record the phase in which each pending utility was interrupted, and record the object on which each utility was operating.

8. Terminate all pending utilities with the following command:

```
-TERM UTILITY(*)
```

9. For environments that do not use data sharing, continue to step 10 on page 707.

Data sharing

For data sharing groups, use the following START DATABASE command on each database that contains objects that are in GRECP or LPL status:

-START DATABASE(*database*) SPACENAM(*)

When you use the START DATABASE command to recover objects, you do not need to provide DB2 with image copies.

Tip: Use up to 10 START DATABASE commands for each DB2 subsystem to increase the speed at which DB2 completes this operation. Multiple commands that run in parallel complete faster than a single command that specifies the same databases.

10. Start all remaining database objects with the following START DATABASE command:

```
START DATABASE(*) SPACENAM(*)
```

11. For each object that the LOAD utility places in a restrictive status, take one of the following actions:
 - If the object was a target of a LOAD utility control statement that specified SHRLEVEL CHANGE, restart the LOAD utility on this object at your convenience. This object contains valid data.
 - If the object was a target of a LOAD utility control statement that specified SHRLEVEL REFERENCE and the LOAD job was interrupted before the RELOAD phase, rebuild the indexes on this object.
 - If the object was a target of a LOAD utility control statement that specified SHRLEVEL REFERENCE and the LOAD job was interrupted during or after the RELOAD phase, recover this object to a point in time that is before this utility ran.
 - Otherwise, recover the object to a point in time that is before the LOAD job ran.
12. For each object that the REORG utility places in a restrictive status, take one of the following actions:
 - When the object was a target of a REORG utility control statement that specified SHRLEVEL NONE:
 - If the REORG job was interrupted before the RELOAD phase, no further action is required. This object contains valid data, and the indexes on this object are valid.
 - If the REORG job was interrupted during the RELOAD phase, recover this object to a point in time that is before this utility ran.
 - If the REORG job was interrupted after the RELOAD phase, rebuild the indexes on the object.
 - When the object was a target of a REORG utility control statement that does not specify SHRLEVEL NONE:
 - If the REORG job was interrupted before the SWITCH phase, no further action is required. This object contains valid data, and the indexes on this object are valid.
 - If the REORG job was interrupted during the SWITCH phase, no further action is required. This object contains valid data, and the indexes on this object are valid.
 - If the REORG job was interrupted after the SWITCH phase, you might need to rebuild non-partitioned secondary indexes.

Recovering with Extended Remote Copy

One method that ensures that data volumes remain consistent at your recovery site involves Extended Remote Copy (XRC). In XRC remote mirroring, the DFSMS

Advanced Copy services function automatically replicates current data from your primary site to a secondary site and establishes consistency groups.

This procedure assumes that you are familiar with basic use of XRC.

To recover at an XRC secondary site after a disaster:

1. Issue the TSO command XEND XRC to end the XRC session.
2. Issue the TSO command XRECOVER XRC. This command changes your secondary site to your primary site and applies the XRC journals to recover data that was in transit when your primary site failed.
3. Complete the procedure in “Recovering in a data mirroring environment” on page 705.

Related information

 z/OS Internet Library

Scenarios for resolving problems with indoubt threads

Indoubt threads can cause a variety of problem. These scenarios provide concrete examples of how you can recover from problems with indoubt threads.

The recovery scenarios for indoubt threads are based on a sample environment, which this topic describes. System programmer, operator, and database administrator actions are indicated for the examples as appropriate. In these descriptions, the term “administrator” refers to the database administrator (DBA) if not otherwise specified.

Configuration

The configuration includes four systems at three geographic locations: Seattle (SEA), San Jose (SJ) and Los Angeles (LA). The system descriptions are as follows.

- DB2 subsystem at Seattle, Location name = IBMSEADB20001, Network name = IBM.SEADB21
- DB2 subsystem at San Jose, Location name = IBMSJ0DB20001, Network name = IBM.SJDB21
- DB2 subsystem at Los Angeles, Location name = IBMLA0DB20001, Network name = IBM.LADB21
- IMS subsystem at Seattle, Connection name = SEAIMS01

Applications

The following IMS and TSO applications run at Seattle and access both local and remote data.

- IMS application, IMSAPP01, at Seattle, accesses local data and remote data by DRDA access at San Jose, which accesses remote data on behalf of Seattle by DB2 private protocol access at Los Angeles.
- TSO application, TSOAPP01, at Seattle, accesses data by DRDA access at San Jose and at Los Angeles.

Threads

The following threads are described and keyed to Figure 69 on page 709. Database access threads (DBAT) access data on behalf of a thread (either allied or DBAT) at a remote requester.

- Allied IMS thread A at Seattle accesses data at San Jose by DRDA access.
 - DBAT at San Jose accesses data for Seattle by DRDA access 1 and requests data at Los Angeles by DB2 private protocol access 2.

- DBAT at Los Angeles accesses data for San Jose by DB2 private protocol access 2.
- Allied TSO thread **B** at Seattle accesses local data and remote data at San Jose and Los Angeles, by DRDA access.
 - DBAT at San Jose accesses data for Seattle by DRDA access 3.
 - DBAT at Los Angeles accesses data for Seattle by DRDA access 4.

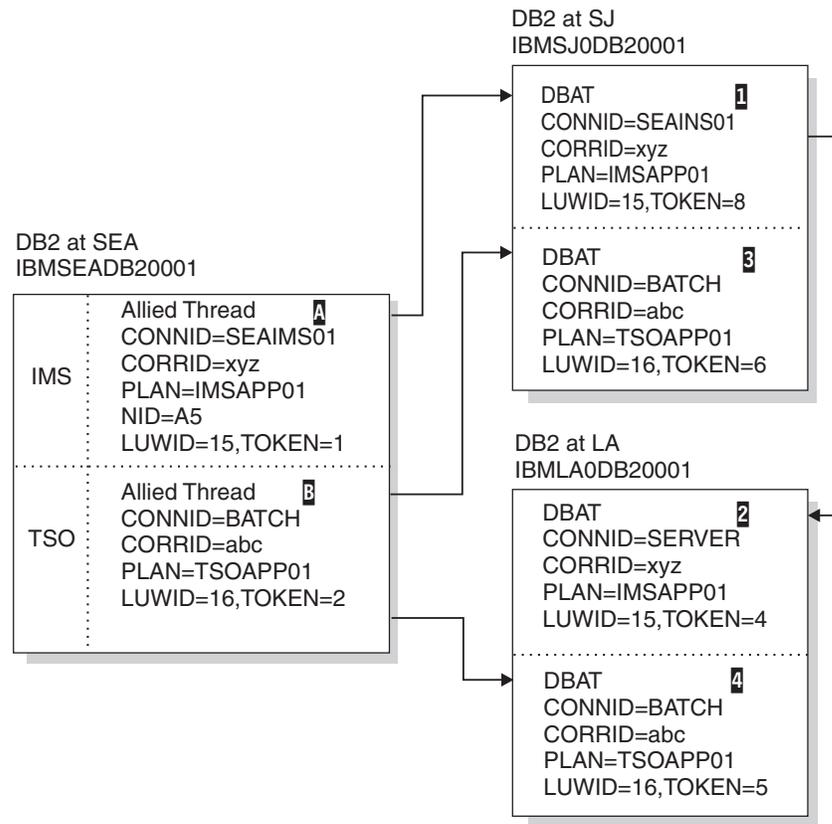


Figure 69. Resolution of indoubt threads. Results of issuing DISPLAY THREAD TYPE(ACTIVE) at each DB2 subsystem.

The results of issuing the DISPLAY THREAD TYPE(ACTIVE) command to display the status of threads at all DB2 locations are summarized in the boxes of the preceding figure. The logical unit of work IDs (LUWIDs) have been shortened for readability, as follows:

- LUWID=15 is IBM.SEADB21.15A86A876789.0010.
- LUWID=16 is IBM.SEADB21.16B57B954427.0003.

For the purposes of procedures that are based on this configuration, assume that both applications have updated data at all DB2 locations. In the following problem scenarios, the error occurs after the coordinator has recorded the commit decision, but before the affected participants have recorded the commit decision. These participants are therefore indoubt.

Read one or more of the scenarios to learn how best to handle problems with indoubt threads in your own environment.

Scenario: Recovering from communication failure

This scenario revolves around a communication failure that occurred between Seattle (SEA) and Los Angeles (LA) after the database access thread (DBAT) at LA completed phase 1 of commit processing. At SEA, the TSO thread, LUWID=16 and TOKEN=2 B, cannot complete the commit with the DBAT at LA4.

Symptoms

At SEA, NetView alert A006 is generated, and message DSNL406 is displayed, indicating that an indoubt thread at LA because of a communication failure. At LA, alert A006 is generated, and message DSNL405 is displayed, to indicate that a thread is in an indoubt state because of a communication failure with SEA.

Causes

A communication failure caused the indoubt thread.

Environment

The following figure illustrates the environment for this scenario.

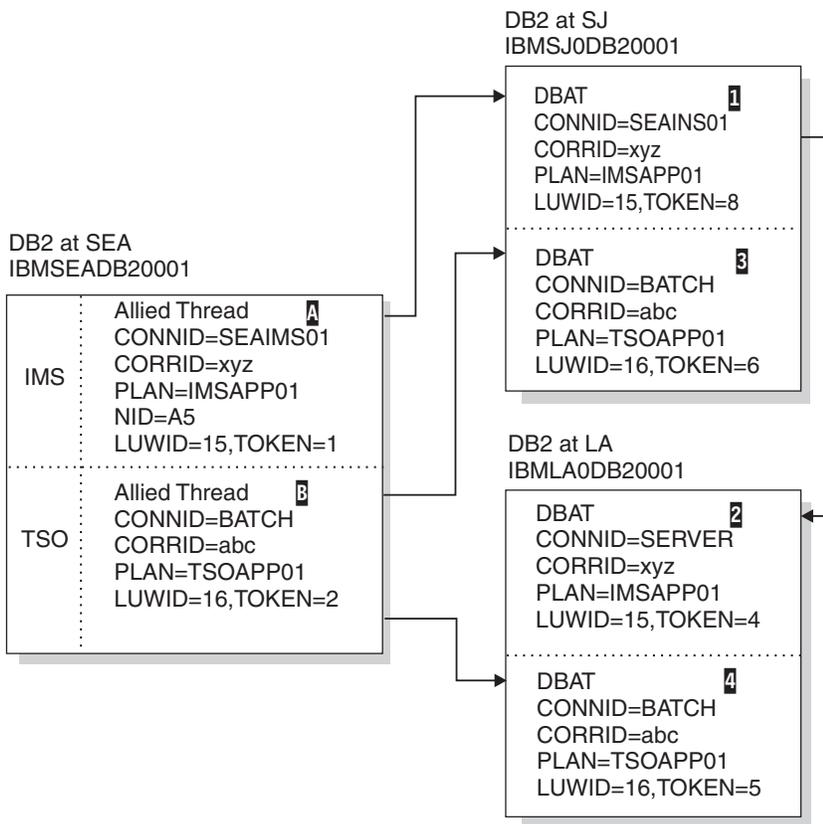


Figure 70. Resolution of indoubt threads. “Scenarios for resolving problems with indoubt threads” on page 708 contains a detailed description of the scenario depicted in this figure.

At SEA, an IFCID 209 trace record is written. After the alert is generated and the message is displayed, the thread completes the commit, which includes the DBAT at SJ 3. Concurrently, the thread is added to the list of threads for which the SEA DB2 subsystem has an indoubt resolution responsibility. The thread shows up in a

DISPLAY THREAD report for indoubt threads. The thread also shows up in a DISPLAY THREAD report for active threads until the application terminates.

The TSO application is informed that the commit succeeded. If the application continues and processes another SQL request, it is rejected with an SQL code to indicate that it must roll back before any more SQL requests can be processed. This is to ensure that the application does not proceed with an assumption based on data that is retrieved from LA, or with the expectation that cursor positioning at LA is still intact.

At LA, an IFCID 209 trace record is written. After the alert is generated and the message displayed, the DBAT 4 is placed in the indoubt state. All locks remain held until resolution occurs. The thread shows up in a DISPLAY THREAD report for indoubt threads.

The DB2 subsystems, at both SEA and LA, periodically attempt to reconnect and automatically resolve the indoubt thread. If the communication failure affects only the session that is being used by the TSO application, and other sessions are available, automatic resolution occurs in a relatively short time. At this time, message DSNL407 is displayed by both DB2 subsystems.

Resolving the problem

Operator response: If message DSNL407 or DSNL415 for the thread that is identified in message DSNL405 is not issued in a reasonable period of time, contact the system programmer. A communication failure is making database resources unavailable.

System programmer response: Determine and correct the cause of the communication failure. When the problem is corrected, automatic resolution of the indoubt thread occurs within a short time. If the failure cannot be corrected for a long time, call the database administrator. The database administrator might want to make a heuristic decision to release the database resources that are held for the indoubt thread.

Related troubleshooting information

“Scenario: Making a heuristic decision about whether to commit or abort an indoubt thread”

Scenario: Making a heuristic decision about whether to commit or abort an indoubt thread

In this scenario, an indoubt thread at Los Angeles (LA) holds database resources that are needed by other applications. The organization makes a heuristic decision about whether to commit or abort an indoubt thread.

Symptoms

Many symptoms are possible, including:

- Message DSNL405 to indicate a thread in the indoubt state
- A DISPLAY THREAD report of active threads showing a larger-than-normal number of threads
- A DISPLAY THREAD report of indoubt threads continuing to show the same thread
- A DISPLAY DATABASE LOCKS report that shows a large number of threads that are waiting for the locks that are held by the indoubt thread

- Some threads terminating due to timeout
- IMS and CICS transactions not completing

Environment

The following figure illustrates the environment for this scenario.

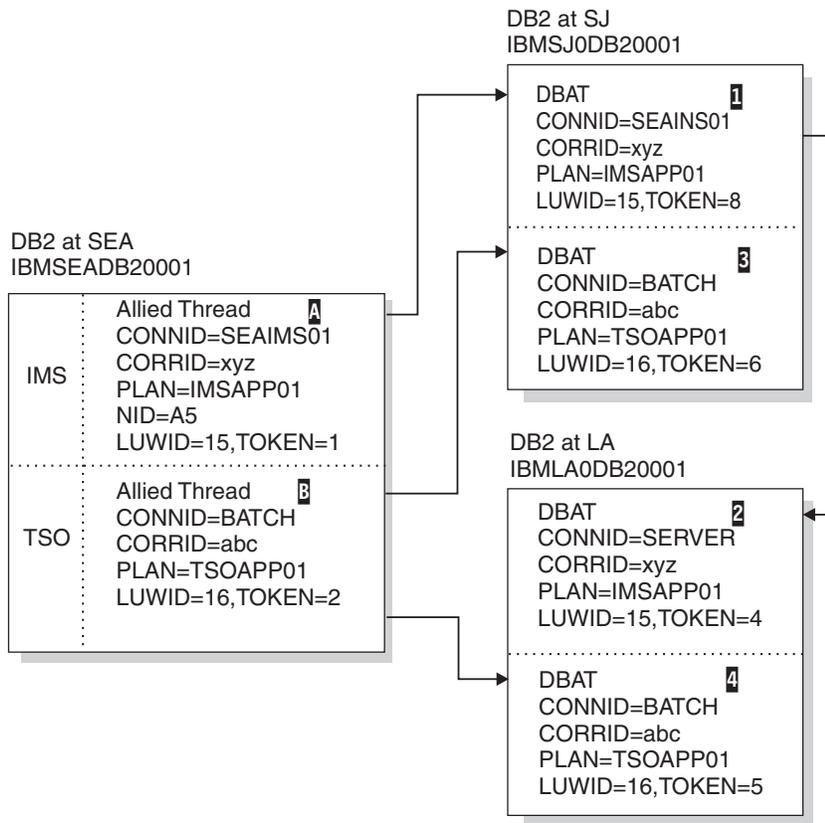


Figure 71. Resolution of indoubt threads. “Scenarios for resolving problems with indoubt threads” on page 708 contains a detailed description of the scenario depicted in this figure.

Resolving the problem

Database administrator response: Determine whether to commit or abort the indoubt thread. First, determine the name of the commit coordinator for the indoubt thread. This name matches the location name of the DB2 subsystem at SEA, and it is included in the DB2 indoubt thread DISPLAY THREAD report at LA. Then, have an authorized person at SEA perform one of the following actions:

- If the coordinator DB2 subsystem is active, or if it can be started, request a DISPLAY THREAD report for indoubt threads, specifying the LUWID of the thread. (Remember that the token that is used at LA is different than the token that is used at SEA). If no report entry exists for the LUWID, the proper action is to abort. If a report entry for the LUWID exists, it shows the proper action to take.
- If the coordinator DB2 subsystem is not active and cannot be started, and if statistics class 4 was active when DB2 was active, search the SEA SMF data for an IFCID 209 event entry that contains the indoubt LUWID. This entry indicates whether the commit decision was commit or abort.

- If statistics class 4 is not available, run the DSN1LOGP utility, and request a summary report. The volume of log data that is to be searched can be restricted if you can determine the approximate SEA log RBA value that was in effect at the time of the communication failure. A DSN1LOGP entry in the summary report for the indoubt LUWID indicates whether the decision was commit or abort.

After determining the correct action to take, issue the RECOVER INDOUBT command at the LA DB2 subsystem, specifying the LUWID and the correct action.

System action: Issuing the RECOVER INDOUBT command at LA results in committing or aborting the indoubt thread. Locks are released. The thread does not disappear from the indoubt thread display until resolution with SEA is completed. The RECOVER INDOUBT report shows that the thread is either committed or aborted by heuristic decision. An IFCID 203 trace record is written, recording the heuristic action.

Scenario: Recovering from an IMS outage that results in an IMS cold start

In this scenario, an IMS outage results in an IMS cold start. The organization recovers from this situation.

Symptoms

When IMS is cold started and later reconnects with the SEA DB2 subsystem, IMS is not able to resolve the indoubt thread with DB2. Message DSNM004I is displayed at the IMS master terminal.

Environment

The following figure illustrates the environment for this scenario.

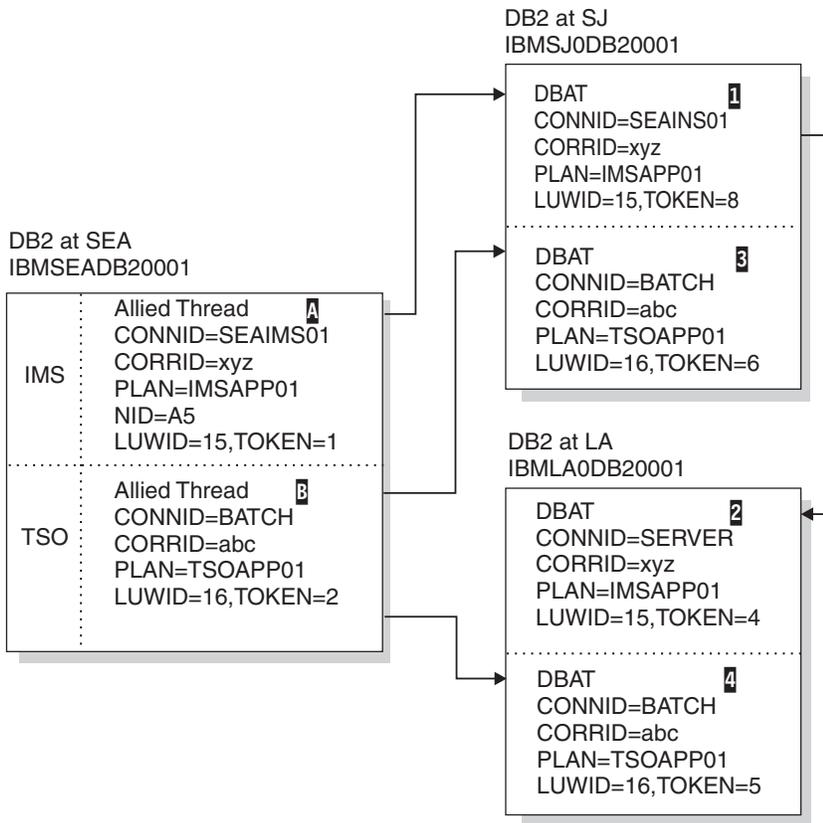


Figure 72. Resolution of indoubt threads. “Scenarios for resolving problems with indoubt threads” on page 708 contains a detailed description of the scenario depicted in this figure.

The abnormal termination of IMS has left one allied thread A at the SEA DB2 subsystem indoubt. This is the thread whose LUWID=15. Because the SEA DB2 subsystem still has effective communication with the DB2 subsystem at SJ, the LUWID=15 DBAT 1 at this subsystem is waiting for the SEA DB2 to communicate the final decision and is not aware that IMS has failed. Also, the LUWID=15 DBAT at LA 2, which is connected to SJ, is also waiting for SJ to communicate the final decision. This cannot be done until SEA communicates the decision to SJ.

- The connection remains active.
- IMS applications can still access DB2 databases.
- Some DB2 resources remain locked out.

If the indoubt thread is not resolved, the IMS message queues can start to back up. If the IMS queues fill to capacity, IMS terminates. Therefore, users must be aware of this potential difficulty and must monitor IMS until the indoubt units of work are fully resolved.

Resolving the problem

System programmer response: Issue the RECOVER INDOUBT command to resolve the indoubt thread at the SEA DB2 subsystem. This completes the two-phase commit process with the DB2 subsystems at SJ and LA, and the unit of work either commits or aborts.

1. Force the IMS log closed by using the /DBR FEOV command, and then archive the IMS log. Use the command DFSERA10 to print the records from the

previous IMS log tape for the last transaction that was processed in each dependent region. Record the PSB and the commit status from the X'37' log that contains the recovery ID.

2. Run the DL/I batch job to back out each PSB that is involved and that has not reached a commit point. The process might take some time because transactions are still being processed. The process might also lock up a number of records, which could affect the rest of the processing and the rest of the message queues.
3. Enter the DB2 command DISPLAY THREAD (*imsid*) TYPE (INDOUBT).
4. Compare the NIDs (IMSID + OASN in hexadecimal) that is displayed in the DISPLAY THREAD messages with the OASNs (4 bytes decimal) as shown in the DFSERA10 output. Decide whether to commit or roll back.
5. Use DFSERA10 to print the X'5501FE' records from the current IMS log tape. Every unit of recovery that undergoes indoubt resolution processing is recorded; each record with an 'IDBT' code is still indoubt. Note the correlation ID and the recovery ID, for use during the next step.
6. Enter the following DB2 command, choosing to commit or roll back, and specify the correlation ID:

```
-RECOVER INDOUBT (imsid) ACTION(COMMIT|ABORT) NID (nid)
```

If the command is rejected because network IDs are associated, use the same command again, substituting the recovery ID for the network ID.

Related concepts

“Duplicate IMS correlation IDs” on page 448

Scenario: Recovering from a DB2 outage at a requester that results in a DB2 cold start

In this scenario, an outage at a DB2 requester results in a cold start. The organization recovers from the situation.

Symptoms

The DB2 subsystem at SEA is started with a conditional restart record in the BSDS to indicate a cold start:

- When the IMS subsystem reconnects, it attempts to resolve the indoubt thread that is identified in IMS as NID=A5. IMS has a resource recovery element (RRE) for this thread. The SEA DB2 subsystem informs IMS that it has no knowledge of this thread. IMS does not delete the RRE, and the RRE can be displayed by using the IMS DISPLAY OASN command. The SEA DB2 subsystem also:
 - Generates message DSN3005 for each IMS RRE for which DB2 has no knowledge
 - Generates an IFCID 234 trace event
- When the DB2 subsystems at SJ and LA reconnect with SEA, each detects that the SEA DB2 subsystem has cold started. Both the SJ DB2 and the LA DB2 subsystem:
 - Display message DSNL411
 - Generate alert A001
 - Generate an IFCID 204 trace event
- A DISPLAY THREAD report of indoubt threads at both the SJ and LA DB2 subsystems shows the indoubt threads and indicates that the coordinator has cold started.

Causes

An abnormal termination of the SEA DB2 subsystem caused the outage.

Environment

The following figure illustrates the environment for this scenario.

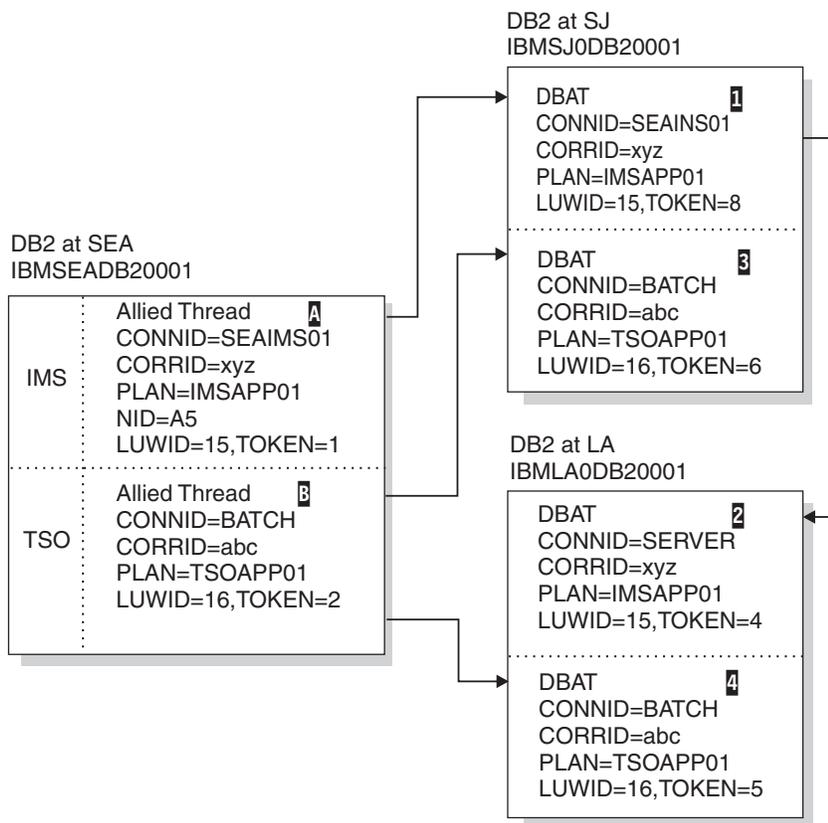


Figure 73. Resolution of indoubt threads. “Scenarios for resolving problems with indoubt threads” on page 708 contains a detailed description of the scenario depicted in this figure.

The abnormal termination of the SEA DB2 subsystem has left the two DBATs at SJ 1, 3, and the LUWID=16 DBAT at LA 4 indoubt. The LUWID=15 DBAT at LA 2, connected to SJ, is waiting for the SJ DB2 subsystem to communicate the final decision.

The IMS subsystem at SEA is operational and has the responsibility of resolving indoubt units with the SEA DB2 subsystem.

The DB2 subsystems at both SJ and LA accept the cold start connection from SEA. Processing continues, waiting for the heuristic decision to resolve the indoubt threads.

Resolving the problem

Database administrator response: At this point:

- Neither the SJ nor the LA administrator know if the SEA coordinator was a participant of another coordinator. In this scenario, the SEA DB2 subsystem originated LUWID=16. However, the SEA DB2 subsystem was a participant for LUWID=15, which was being coordinated by IMS.

- The administrator at LA also does not know is the fact that SEA distributed the LUWID=16 thread to SJ, where it is also indoubt. Likewise, the administrator at SJ does not know that LA has an indoubt thread for the LUWID=16 thread. Both SJ and LA need to make the same heuristic decision. The administrators at SJ and LA also need to determine the originator of the two-phase commit.
- The recovery log of the originator indicates whether the decision was commit or abort. The originator might have more accessible functions to determine the decision. Even though the SEA DB2 subsystem cold started, you might be able to determine the decision from its recovery log. Alternatively, if the failure occurred before the decision was recorded, you might be able to determine the name of the coordinator, if the SEA DB2 subsystem was a participant. You can obtain a summary report of the SEA DB2 recovery log by running the DSN1LOGP utility.
- The LUWID contains the name of the logical unit (LU) where the distributed logical unit of work originated. This logical unit is most likely in the system that originated the two-phase commit.
- If an application is distributed, any distributed piece of the application can initiate the two-phase commit. In this type of application, the originator of two-phase commit can be at a different system than the site that is identified by the LUWID. With DB2 private protocol access, the two-phase commit can flow only from the system that contains the application that initiates distributed SQL processing. In most cases, this is where the application originates.
- The administrator must determine if the LU name that is contained in the LUWID is the same as the LU name of the SEA DB2 subsystem. If this is not the case (it is the case in this example), the SEA DB2 subsystem is a participant in the logical unit of work, and it is being coordinated by a remote system. The DBA must communicate with that system and request that facilities of that system be used to determine if the logical unit of work is to be committed or aborted.
- If the LUWID contains the LU name of the SEA DB2 subsystem, the logical unit of work originated at SEA and can be an IMS, CICS, TSO, or batch allied thread of the SEA DB2 subsystem. The DISPLAY THREAD report for indoubt threads at a DB2 participant includes message DSNV458 if the coordinator is remote. This line provides external information that is provided by the coordinator to assist in identifying the thread. A DB2 coordinator provides the following identifier:

connection-name.correlation-id

where *connection-name* is:

- SERVER: the thread represents a remote application to the DB2 coordinator and uses DRDA access.
- BATCH: the thread represents a local batch application to the DB2 coordinator.
- Anything else represents an IMS or CICS connection name. The thread represents a local application, and the commit coordinator is the IMS or CICS system by using this connection name.
- In this example, the administrator at SJ sees that both indoubt threads have an LUWID with the LU name that match the SEA DB2 subsystem LU name, and furthermore, that one thread (LUWID=15) is an IMS thread and the other thread (LUWID=16) is a batch thread. The LA administrator sees that the LA indoubt thread (LUWID=16) originates at the SEA DB2 subsystem and is a batch thread.
- The originator of a DB2 batch thread is DB2. To determine the commit or abort decision for the LUWID=16 indoubt threads, the SEA DB2 recovery log must be

analyzed, if possible. Run the DSN1LOGP utility against the SEA DB2 recovery log, and look for the LUWID=16 entry. Three possibilities exist:

1. No entry is found. That portion of the DB2 recovery log is not available.
 2. An entry is found but incomplete.
 3. An entry is found, and the status is committed or aborted.
- In the third case, the heuristic decision at SJ and LA for indoubt thread LUWID=16 is indicated by the status that is indicated in the SEA DB2 recovery log. In the other two cases, the recovery procedure that is used when cold starting DB2 is important. If recovery was to a previous point in time, the correct action is to abort. If recovery included repairing the SEA DB2 database, the SEA administrator might know what decision to make.
 - The recovery logs at SJ and LA can help determine what activity took place. If the administrator determines that updates were performed at SJ, LA, or both (but not SEA), and if both SJ and LA make the same heuristic action, data inconsistency probably exists. If updates were also performed at SEA, the administrator can look at the SEA data to determine what action to take. In any case, both SJ and LA should make the same decision.
 - For the indoubt thread with LUWID=15 (the IMS coordinator), several alternative paths to recovery are available. The SEA DB2 subsystem has been restarted. When it reconnects with IMS, message DSN3005 is issued for each thread that IMS is trying to resolve with DB2. The message indicates that DB2 has no knowledge of the thread that is identified by the IMS-assigned NID. The outcome for the thread, either commit or abort, is included in the message. Trace event IFCID=234 is also written to statistics class 4, which contains the same information.
 - If only one such message exists, or if one such entry is in statistics class 4, the decision for indoubt thread LUWID=15 is known and can be communicated to the administrator at SJ. If multiple such messages exist, or if multiple such trace events exist, the administrator must match the IMS NID with the network LUWID. Again, the administrator should use DSN1LOGP to analyze the SEA DB2 recovery log if possible. Now four possibilities exist:
 1. No entry is found. That portion of the DB2 recovery log was not available.
 2. An entry is found but is incomplete because of lost recovery log data.
 3. An entry is found, and the status is indoubt.
 4. An entry is found, and the status is committed or aborted.
 - In the fourth case, the heuristic decision at SJ for the indoubt thread LUWID=15 is determined by the status that is indicated in the SEA DB2 recovery log. If an entry is found and its status is indoubt, DSN1LOGP also reports the IMS NID value. The NID is the unique identifier for the logical unit of work in IMS and CICS. Knowing the NID enables correlation to the DSN3005 message, or to the 234 trace event, either of which provides the correct decision.
 - If an incomplete entry is found, the NID might have been reported by DSN1LOGP. If it was reported, use it as previously discussed.
 - Determine if any of the following conditions exists:
 - No NID is found.
 - The SEA DB2 subsystem has not been started.
 - Reconnecting to IMS has not occurred.

If any of these conditions exists, the administrator must use the *correlation-id* that is used by IMS to correlate the IMS logical unit of work to the DB2 thread in a search of the IMS recovery log. The SEA DB2 site provided this value to the SJ DB2 subsystem when distributing the thread to SJ. The SJ DB2 site displays this value in the report that is generated by the DISPLAY THREAD TYPE(INDOUBT) command.

- For IMS, the *correlation-id* is:
pst#.psbname
- In CICS, the *correlation-id* consists of four parts:
Byte 1 - Connection type - G=Group, P=Pool
Byte 2 - Thread type - T=transaction, G=Group, C=Command
Bytes 3-4 - Thread number
Bytes 5-8 - Transaction-id

Related concepts

“Scenario: What happens when the wrong DB2 subsystem is cold started”

Scenario: What happens when the wrong DB2 subsystem is cold started

This scenario explains what happens when one DB2 subsystem, instead of another DB2 subsystem, is cold started. Threads are left indoubt, and the organization recovers from the situation.

The following figure illustrates the environment for this scenario.

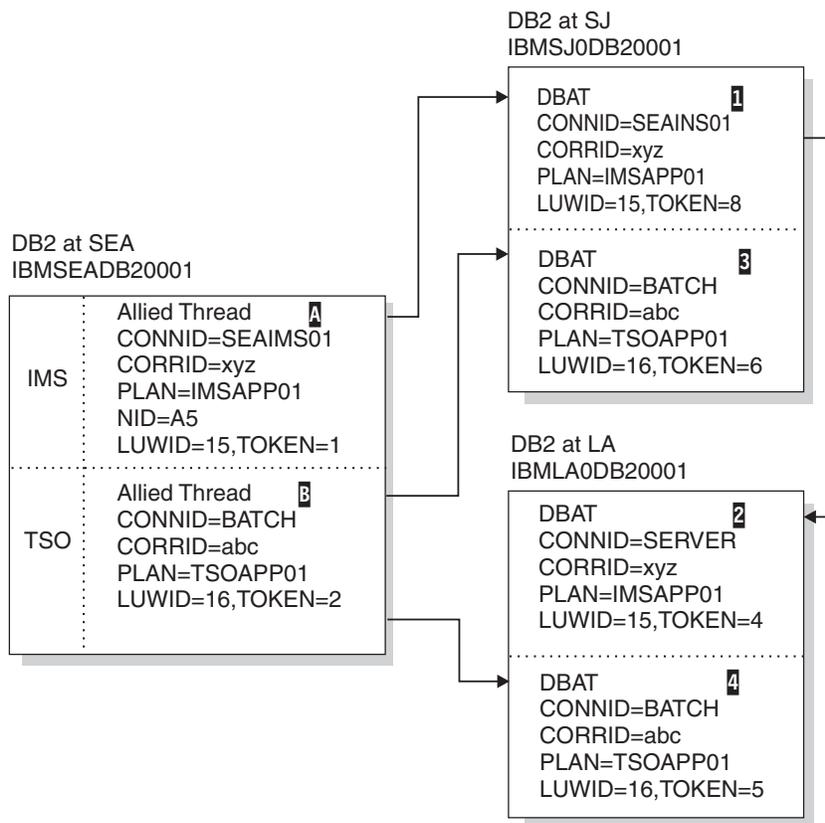


Figure 74. Resolution of indoubt threads. “Scenarios for resolving problems with indoubt threads” on page 708 contains a detailed description of the scenario depicted in this figure.

If the DB2 subsystem at SJ is cold started instead of the DB2 at SEA, the LA DB2 subsystem has the LUWID=15 2 thread indoubt. The administrator can see that this thread did not originate at SJ, but that it did originate at SEA. To determine the commit or abort action, the LA administrator requests that DISPLAY THREAD TYPE(INDOUBT) be issued at the SEA DB2 subsystem, specifying LUWID=15. IMS

does not have any indoubt status for this thread because it completes the two-phase commit process with the SEA DB2 subsystem.

The DB2 subsystem at SEA tells the application that the commit succeeded.

When a participant cold starts, a DB2 coordinator continues to include in the display of information about indoubt threads all committed threads where the cold starting participant was believed to be indoubt. These entries must be explicitly purged by issuing the RESET INDOUBT command. If a participant has an indoubt thread that cannot be resolved because of coordinator cold start, the administrator can request a display of indoubt threads at the DB2 coordinator to determine the correct action.

Related troubleshooting information

“Scenario: Recovering from a DB2 outage at a requester that results in a DB2 cold start” on page 715

“Scenario: Recovering from communication failure” on page 710

Scenario: Correcting damage from an incorrect heuristic decision about an indoubt thread

This scenario assumes that an incorrect heuristic decision was made regarding an indoubt thread. The organization recovers from this incorrect decision.

Symptoms

When the DB2 subsystem at SEA reconnects with the DB2 at LA, indoubt resolution occurs for LUWID=16. Both systems detect heuristic damage, and both generate alert A004; each writes an IFCID 207 trace record. Message DSNL400 is displayed at LA, and message DSNL403 is displayed at SEA.

Causes

This scenario is based on the conditions described in “Scenario: Recovering from communication failure” on page 710.

The LA administrator is called to make an heuristic decision and decides to abort the indoubt thread with LUWID=16. The decision is made without communicating with SEA to determine the proper action. The thread at LA is aborted, whereas the threads at SEA and SJ are committed. Processing continues at all systems. The DB2 subsystem at SEA has indoubt resolution responsibility with LA for LUWID=16.

Environment

The following figure illustrates the environment for this scenario.

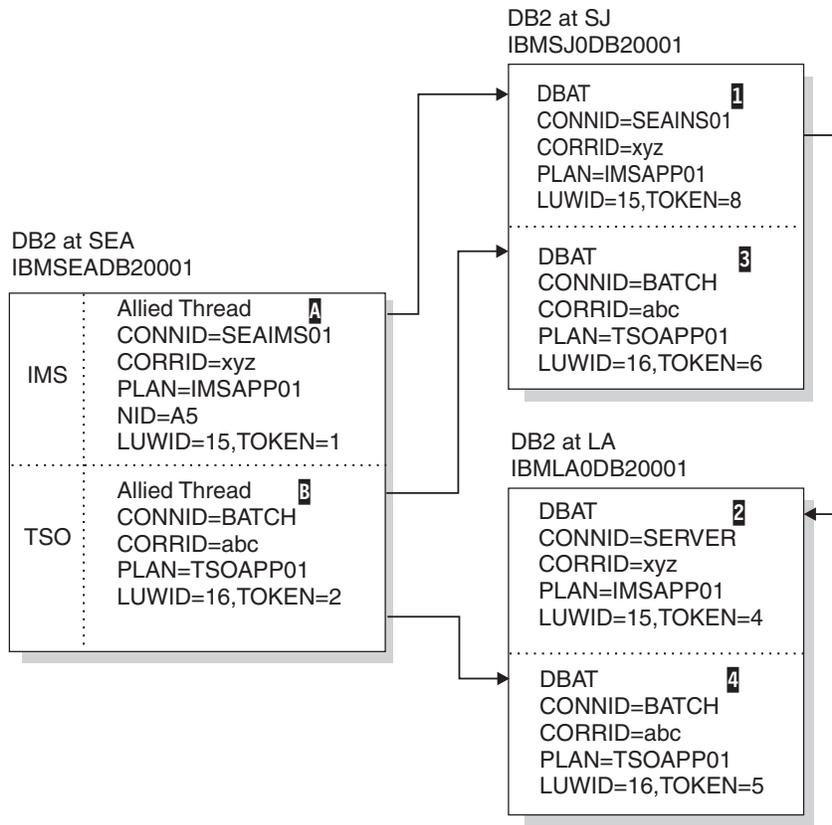


Figure 75. Resolution of indoubt threads. “Scenarios for resolving problems with indoubt threads” on page 708 contains a detailed description of the scenario depicted in this figure.

In this scenario, processing continues. Indoubt thread resolution responsibilities have been fulfilled, and the thread completes at both SJ and LA.

Resolving the problem

Database administrator response: Correct the damage. This is not an easy task. Since the time of the heuristic action, the data at LA might have been read or written by many applications. Correcting the damage can involve reversing the effects of these applications, also. The available tools are:

- DSN1LOGP utility, which generates a summary report that identifies the table spaces that were modified by the LUWID=16 thread.
- The statistics trace class 4, which contains an IFCID 207 entry. This entry identifies the recovery log RBA for the LUWID=16 thread.

Notify IBM Software Support about the problem.

Chapter 21. Reading log records

For diagnostic or recovery purposes, it can be useful to read DB2 log records.

This section provides information about the log and discusses three approaches to writing programs that read log records.

Contents of the log

The log contains the information needed to recover the results of program execution, the contents of the database, and the DB2 subsystem. The log does not contain information for accounting, statistics, traces, or performance evaluation.

PSPI There are three main types of log records: unit of recovery, checkpoint, and database page set control records.

Each log record has a header that indicates its type, the DB2 subcomponent that made the record, and, for unit-of-recovery records, the unit-of-recovery identifier. The log records can be extracted and printed by the DSN1LOGP program.

The log relative byte address and log record sequence number

The DB2 log can contain up to 2^{48} bytes, where 2^{48} is 2 to the 48th power. Each byte is addressable by its offset from the beginning of the log. That offset is known as its *relative byte address* (RBA).

A log record is identifiable by the RBA of the first byte of its header; that RBA is called the *relative byte address of the record*. The record RBA is like a timestamp because it uniquely identifies a record that starts at a particular point in the continuing log.

In the data sharing environment, each member has its own log. The *log record sequence number* (LRSN) uniquely identifies the log records of a data sharing member. The LRSN is a 6-byte hexadecimal value derived from a store clock timestamp. DB2 uses the LRSN for recovery in the data sharing environment.

Effects of ESA data compression

Log records can contain compressed data if a table contains compressed data. For example, if the data in a DB2 row are compressed, all data logged because of changes to that row (resulting from inserts, updates and deletes) are compressed. If logged, the record prefix is not compressed, but all of the data in the record are in compressed format. Reading compressed data requires access to the dictionary that was in use when the data was compressed. **PSPI**

Related reference

"DSN1LOGP" (DB2 Utility Guide and Reference)

Unit of recovery log records

Most of the log records describe changes to the DB2 database. All such changes are made within units of recovery.

This section describes changes to the DB2 database, the effects of these changes, and the log records that correspond to the changes.

Undo and redo records

When a change is made to the database, DB2 logs an *undo/redo* record that describes the change.

PSPI The *redo* information is required if the work is committed and later must be recovered. The *undo* information is used to back out work that is not committed.

If the work is rolled back, the undo/redo record is used to remove the change. At the same time that the change is removed, a new redo/undo record is created that contains information, called *compensation information*, that is used if necessary to reverse the change. For example, if a value of 3 is changed to 5, redo compensation information changes it back to 3.

If the work must be recovered, DB2 scans the log forward and applies the redo portions of log records and the redo portions of compensation records, without keeping track of whether the unit of recovery was committed or rolled back. If the unit of recovery had been rolled back, DB2 would have written compensation redo log records to record the original undo action as a redo action. Using this technique, the data can be completely restored by applying only redo log records on a single forward pass of the log.

DB2 also logs the creation and deletion of data sets. If the work is rolled back, the operations are reversed. For example, if a table space is created using DB2-managed data sets, DB2 creates a data set; if rollback is necessary, the data set is deleted. If a table space using DB2-managed data sets is dropped, DB2 deletes the data set when the work is committed, not immediately. If the work is rolled back, DB2 does nothing. **PSPI**

Database exception table records

Database exception table (DBET) log records register several types of information: exception states and image copies of special table spaces.

PSPI DBET log records also register exception information that is not related to units of recovery.

Exception states

DBET log records register whether any database, table space, index space, or partition is in an exception state. To list all objects in a database that are in an exception state, use the command `DISPLAY DATABASE (database name) RESTRICT`.

Image copies of special table spaces

Image copies of DSNDB01.SYSUTILX, DSNDB01.DBD01, and DSNDB06.SYSCOPY are registered in the DBET log record rather than in SYSCOPY. During recovery, they are recovered from the log, and then image copies of other table spaces are located from the recovered SYSCOPY. **PSPI**

Related reference

“Other exception information” on page 728

"DSNT392I" (DB2 Messages)

Typical unit of recovery log records

This example shows a sequence of log records that might be written for an insert of one row via TSO.

PSPI The following record types are included:

Begin_UR

The first request to change a database begins a unit of recovery. The log record of that event is identified by its log RBA. That same RBA serves as an ID for the entire unit of recovery (the URID). All records related to that unit have that RBA in their log record headers (LRH). For rapid backout, the records are also linked by a backward chain in the LRH.

Undo/Redo

Log records are written for each insertion, deletion, or update of a row. They register the changes to the stored data, but not the SQL statement that caused the change. Each record identifies one data or index page and its changes.

End Phase 2 records

The end of a UR is marked by log records that tell whether the UR was committed or rolled back, and whether DB2 has completed the work associated with it. If DB2 terminates before a UR has completed, it completes the work at the next restart.

Table 111. Example of a log record sequence for an INSERT of one row using TSO

Type of record	Information recorded
1. Begin_UR	Beginning of the unit of recovery. Includes the connection name, correlation name, authorization ID, plan name, and LUWID.
2. Undo/Redo for data	Insertion of data. Includes the database ID (DBID), page set ID, page number, internal record identifier (RID), and the data inserted.
3. Undo/Redo for Index	Insertion of index entry. Includes the DBID, index space object ID, page number, and index entry to be added.
4. Begin Commit 1	The beginning of the commit process. The application has requested a commit either explicitly (EXEC SQL COMMIT) or implicitly (for example, by ending the program).
5. Phase 1-2 Transition	The agreement to commit in TSO. In CICS and IMS, an End Phase 1 record notes that DB2 agrees to commit. If both parties agree, a Begin Phase 2 record is written; otherwise, a Begin Abort record is written, noting that the unit of recovery is to be rolled back.
6. End Phase 2	Completion of all work required for commit.

Table 112 shows the log records for processing and rolling back an insertion.

Table 112. Log records written for rolling back an insertion

Type of record	Information recorded
1. Begin_UR	Beginning of the unit of recovery.

Table 112. Log records written for rolling back an insertion (continued)

Type of record	Information recorded
2. Undo/Redo for data	Insertion of data. Includes the database ID (DBID), page set ID, page number, internal record identifier, and the data inserted.
3. Begin_Abort	Beginning of the rollback process.
4. Compensation Redo/Undo	Backing-out of data. Includes the database ID (DBID), page set ID, page number, internal record ID (RID), and data to undo the previous change.
5. End_Abort	End of the unit of recovery, with rollback complete.

PSPI

Classes of changes to data

There are three basic classes of changes to a data page: changes to control information, to database pointers, or to the data.

PSPI

Changes to control information

Those changes include pages that map available space and indicators that show that a page has been modified. The COPY utility uses that information when making incremental image copies.

Changes to database pointers.

Pointers are used in two situations:

- The DB2 catalog and directory, but not user databases, contain pointers that connect related rows. Insertion or deletion of a row changes pointers in related data rows.
- When a row in a user database becomes too long to fit in the available space, it is moved to a new page. An address, called an *overflow pointer*, that points to the new location is left in the original page. With this technique, index entries and other pointers do not have to be changed. Accessing the row in its original position gives a pointer to the new location.

Changes to data

In DB2, a row is confined to a single page. Each row is uniquely identified by a RID containing:

- The number of the page
- A 1-byte ID that identifies the row within the page. A single page can contain up to 255 rows. (A page in a catalog table space that has links can contain up to 127 rows.) IDs are reused when rows are deleted.

The log record identifies the RID, the operation (insert, delete, or update), and the data. Depending on the data size and other variables, DB2 can write a single log record with both undo and redo information, or it can write separate log records for undo and redo.

The following table summarizes the information logged for data and index changes.

Table 113. Information logged for database changes

Operation	Information logged
Insert data	The new row <ul style="list-style-type: none"> • On redo, the row is inserted with its original RID. • On undo, the row is deleted and the RID is made available for another row.
Delete data	The deleted row <ul style="list-style-type: none"> • On redo, the RID is made available for another row. • On undo, the row is inserted again with its former RID.
Update data ¹	The old and new values of the changed data. <ul style="list-style-type: none"> • On redo, the new data is replaced. • On undo, the old data is replaced.
Insert index entry	The new key value and the data RID.
Delete index entry	The deleted key value and the data RID.
Add column	The information about the column being added, if the table was defined with DATA CAPTURE(CHANGES).
Alter column	The information about the column being altered, if the table was defined with DATA CAPTURE(CHANGES).
Roll back to a savepoint	Information about the savepoint.
Modify table space	Information about the table space version.

Note:

1. If an update occurs to a table defined with DATA CAPTURE(CHANGES), the entire before-image and after-image of the data row is logged.



Checkpoint log records

DB2 takes periodic checkpoints during normal operation in order to reduce restart time.



DB2 takes checkpoints in the following circumstances:

- When a predefined number of log records have been written or a predetermined amount of time in minutes has elapsed.
This number is defined by field CHECKPOINT_FREQ on installation panel DSNTIPL.
- When switching from one active log data set to another
- At the end of a successful restart
- At normal termination

At a checkpoint, DB2 logs its current status and registers the log RBA of the checkpoint in the bootstrap data set (BSDS). At restart, DB2 uses the information in the checkpoint records to reconstruct its state when it terminated.

Many log records can be written for a single checkpoint. DB2 can write one to begin the checkpoint; others can then be written, followed by a record to end the checkpoint. The following table summarizes the information logged.

Table 114. Contents of checkpoint log records

Type of log record	Information logged
Begin_Checkpoint	Marks the start of the summary information. All later records in the checkpoint have type X'0100' (in the LRH).
Unit of Recovery Summary	Identifies an incomplete unit of recovery (by the log RBA of the Begin_UR log record). Includes the date and time of its creation, its connection ID, correlation ID, authorization ID, the plan name it used, and its current state (inflight, indoubt, in-commit, or in-abort).
Page Set Summary	Contains information for allocating and opening objects at restart, and identifies (by the log RBA) the earliest checkpoint interval containing log records about data changes that have not been applied to the DASD version of the data or index. There is one record for each open page set (table space or index space).
Page Set Exception Summary	Identifies the type of exception state. For descriptions of the states, see "Database page set control records." There is one record for each database and page set with an exception state.
Page Set UR Summary Record	Identifies page sets modified by any active UR (inflight, in-abort, or in-commit) at the time of the checkpoint.
End_Checkpoint	Marks the end of the summary information about a checkpoint.

PSPI

Related reference

"Active log data set parameters: DSNTIPL" (DB2 Installation Guide)

Database page set control records

Page set control records primarily register the allocation, opening, and closing of every page set (table space or index space).

PSPI The same information is in the DB2 directory (SYSIBM.SYSLGRNX). It is also registered in the log so that it is available at restart. **PSPI**

Other exception information

Entries for data pages that are logically in error (logical page list or LPL entries) or physically in error (write error page range or WEPR entries) are registered in the DBET log record.

The physical structure of the log

The active log consists of VSAM data sets with certain required characteristics.

PSPI The physical output unit written to the active log data set is a control interval (CI) of 4096 bytes (4 KB). Each CI contains one VSAM record. **PSPI**

Physical and logical log records

The VSAM CI provides 4089 bytes to hold DB2 information. That space is called a *physical* record. The information to be logged at a particular time forms a *logical* record, whose length varies independently of the space available in the CI.



One physical record can contain several logical records, one or more logical records and part of another, or only part of one logical record. The physical record must also contain 21 bytes of DB2 control information, called the *log control interval definition (LCID)*.

Figure 76 shows a VSAM CI containing four log records or segments, namely:

- The last segment of a log record of 768 bytes (X'0300'). The length of the segment is 100 bytes (X'0064').
- A complete log record of 40 bytes (X'0028').
- A complete log record of 1024 bytes (X'0400').
- The first segment of a log record of 4108 bytes (X'100C'). The length of the segment is 2911 bytes (X'0B5F').

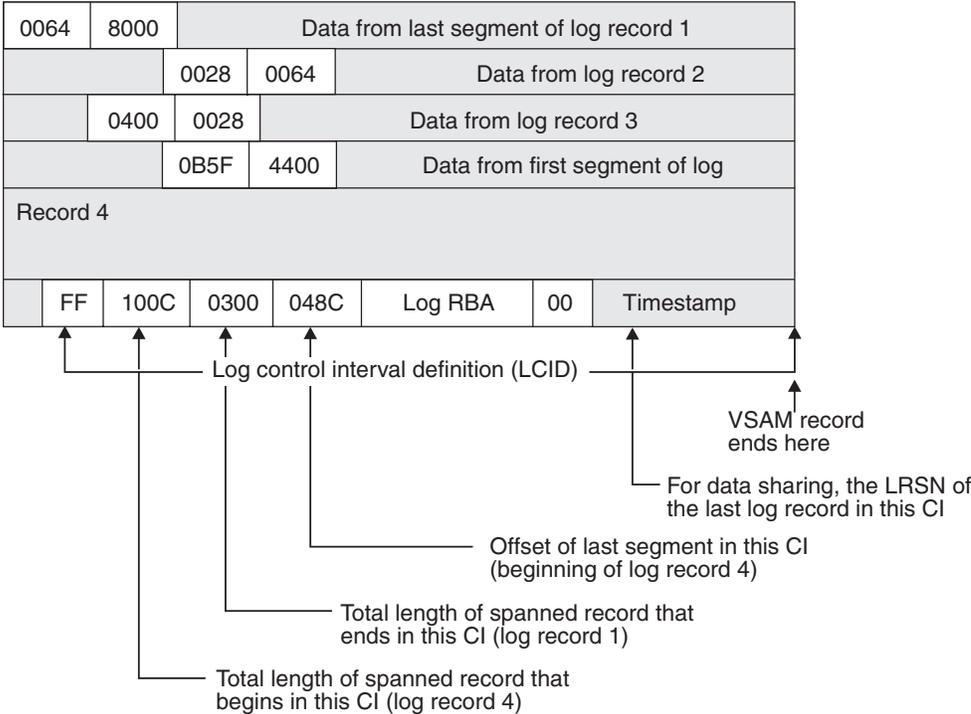


Figure 76. A VSAM CI and its contents

The term *log record* refers to a logical record, unless the term *physical log record* is used. A part of a logical record that falls within one physical record is called a *segment*.



Related reference

“The log control interval definition (LCID)” on page 730

The log record header

Each logical record includes a prefix, called a *log record header (LRH)*, which contains control information.



The first segment of a log record must contain the header and some bytes of data. If the current physical record has too little room for the minimum segment

of a new record, the remainder of the physical record is unused, and a new log record is written in a new physical record.

The log record can span many VSAM CIs. For example, a minimum of nine CIs are required to hold the maximum size log record of 32815 bytes. Only the first segment of the record contains the entire LRH; later segments include only the first two fields. When a specific log record is needed for recovery, all segments are retrieved and presented together as if the record were stored continuously.

Table 115. Contents of the log record header

Hex offset	Length	Information
00	2	Length of this record or segment
02	2	Length of any previous record or segment in this CI; 0 if this is the first entry in the CI. The two high-order bits tell the segment type: B'00' A complete log record B'01' The first segment B'11' A middle segment B'10' The last segment
04	2	Type of log record
06	2	Subtype of the log record
08	1	Resource manager ID (RMID) of the DB2 component that created the log record
09	1	Flags
0A	6	Unit of recovery ID, if this record relates to a unit of recovery; otherwise, 0
10	6	Log RBA of the previous log record, if this record relates to a unit of recovery; otherwise, 0
16	1	Release identifier
17	1	Length of header
18	6	Undo next LSN
1E	8	LRHTIME

PSPI

Related concepts

“Unit of recovery log records” on page 723

Related reference

“Log record type codes” on page 732

“Log record subtype codes” on page 733

The log control interval definition (LCID)

Each physical log record includes a suffix called the *log control interval definition* (LCID), which tells how record segments are placed in the physical control interval.

PSPI The following table describes the contents of the LCID.

Table 116. Contents of the log control interval definition

Hex offset	Length	Information
00	1	An indication of whether the CI contains free space: X'00' = Yes, X'FF' = No
01	2	Total length of a segmented record that begins in this CI; 0 if no segmented record begins in this CI
03	2	Total length of a segmented record that ends in this CI; 0 if no segmented record ends in this CI
05	2	Offset of the last record or segment in the CI
07	6	Log RBA of the start of the CI
0D	6	Timestamp, reflecting the date and time that the log buffer was written to the active log data set. The timestamp is the high-order 7 bytes of the store clock value (STCK).
13	2	Reserved

Each recovery log record consists of two parts: a *header*, which describes the record, and data. The following illustration shows the format schematically; the following list describes each field.

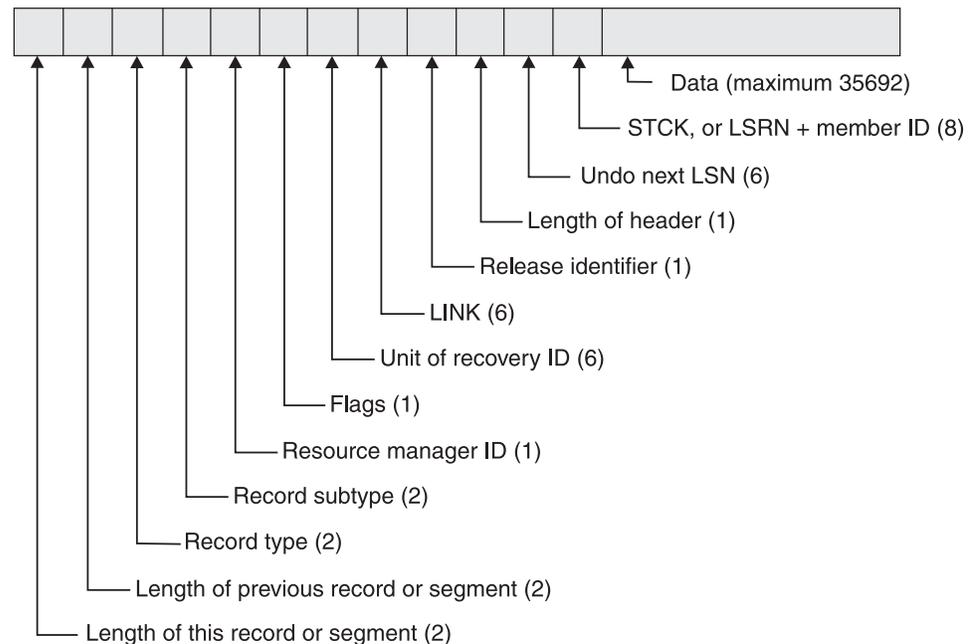


Figure 77. Format of a DB2 recovery log record

The fields are:

Length of this record

The total length of the record in bytes.

Length of previous record

The total length of the previous record in bytes.

Type The code for the type of recovery log record.

Subtype

Some types of recovery log records are further divided into subtypes.

Resource manager ID

Identifier of the resource manager that wrote the record into the log. When the log is read, the record can be given for processing to the resource manager that created it.

Unit of recovery ID

A unit of recovery to which the record is related. Other log records can be related to the same unit of recovery; all of them must be examined to recover the data. The URID is the RBA (relative byte address) of the Begin-UR log record, and indicates the start of that unit of recovery in the log.

LINK Chains all records written using their RBAs. For example, the link in an end checkpoint record links the chains back to the begin checkpoint record.

Release identifier

Identifies in which release the log was written.

Log record header length

The total length of the header of the log record.

Undo next LSN

Identifies the log RBA of the next log record to be undone during backwards (UNDO processing) recovery.

STCK, or LRSN+member ID

In a non-data-sharing environment, this is a 6-byte store clock value (STCK) reflecting the date and time the record was placed in the output buffer. The last 2 bytes contain zeros.

In a data sharing environment, this contains a 6-byte log record sequence number (LRSN) followed by a 2-byte member ID.

Data Data associated with the log record. The contents of the data field depend on the type and subtype of the recovery log record.

**Related reference**

“Log record type codes”

“Log record subtype codes” on page 733

Log record type codes

The type code of a log record tells what kind of DB2 event the record describes.



Code	Type of event
0002	Page set control
0004	SYSCOPY utility
0010	System event
0020	Unit of recovery control
0100	Checkpoint
0200	Unit of recovery undo

0400 Unit of recovery redo
 0800 Archive log command
 1000 to 8000 Assigned by DB2
 2200 Savepoint
 4200 End of rollback to savepoint
 4400 Alter or modify recovery log record

A single record can contain multiple type codes that are combined. For example, 0600 is a combined UNDO/REDO record; F400 is a combination of four DB2-assigned types plus a REDO. A diagnostic log record for the TRUNCATE IMMEDIATE statement is type code 4200, which is a combination of a diagnostic log record (4000) and an UNDO record (0200). 

Log record subtype codes

The log record subtype code provides a more granular definition of the event that occurred to produce the log record. Log record subtype codes are unique only within the scope of the corresponding log record type code.

 Log record type 0004 (SYSCOPY utility) has log subtype codes that correspond to the page set ID values of the table spaces that have their SYSCOPY records in the log (SYSIBM.SYSUTILX, SYSIBM.SYSCOPY and DSNDB01.DBD01).

Log record type 0800 (quiesce) does not have subtype codes.

Some log record types (1000 to 8000 assigned by DB2) can have proprietary log record subtype codes assigned.

Subtypes for type 0002 (page set control)

Code	Type of event
0001	Page set open
0002	Data set open
0003	Page set close
0004	Data set close
0005	Page set control checkpoint
0006	Page set write
0007	Page set write I/O
0008	Page set reset write
0009	Page set status

Subtypes for type 0010 (system event)

Code	Type of event
0001	Begin checkpoint
0002	End checkpoint
0003	Begin current status rebuild

- 0004 Begin historic status rebuild
- 0005 Begin active unit of recovery backout
- 0006 Pacing record

Subtypes for type 0020 (unit of recovery control)

- | Code | Type of event |
|------|---|
| 0001 | Begin unit of recovery |
| 0002 | Begin commit phase 1 (Prepare) |
| 0004 | End commit phase 1 (Prepare) |
| 0008 | Begin commit phase 2 |
| 000C | Commit phase 1 to commit phase 2 transition |
| 0010 | End commit phase 2 |
| 0020 | Begin abort |
| 0040 | End abort |
| 0081 | End undo |
| 0084 | End todo |
| 0088 | End redo |

Subtypes for type 0100 (checkpoint)

- | Code | Type of event |
|------|--------------------------------|
| 0001 | Unit of recovery entry |
| 0002 | Restart unit of recovery entry |

Subtypes for type 2200 (savepoint)

- | Code | Type of event |
|------|-----------------------|
| 0014 | Rollback to savepoint |
| 000E | Release to savepoint |

Subtypes for type 4200 (end of rollback to savepoint)

- | Code | Type of event |
|------|------------------|
| 0084 | End of savepoint |
| 0085 | End of savepoint |

Subtypes for type 4200 (diagnostic log record for TRUNCATE IMMEDIATE)

- | Code | Type of event |
|------|---------------------------------------|
| 0085 | Special begin for TRUNCATE IMMEDIATE |
| 0086 | Special commit for TRUNCATE IMMEDIATE |

Subtypes for type 4400 (alter or modify log record)

- | Code | Type of event |
|------|---|
| 0083 | Alter or modify log record used for DB2 replication |

PSPI

Related reference

"DSN1LOGP" (DB2 Utility Guide and Reference)

Interpreting data change log records

DB2 provides the mapping and description of specific log record types that allow you to interpret data changes made to DB2 tables from the log.

PSPI

The macros are contained in the data set library prefix. SDSNMACS and are documented by comments in the macros themselves.

Log record formats for the record types and subtypes are detailed in the mapping macro DSNDQJ00. DSNDQJ00 provides the mapping of specific data change log records, UR control log records, and page set control log records that you need to interpret data changes by the UR. DSNDQJ00 also explains the content and usage of the log records. PSPI

Related reference

"Log record subtype codes" on page 733

Reading log records with IFI

This is an online method using the instrumentation facility interface (IFI) when DB2 is running. You use the READA (read asynchronously) command of IFI to read log records into a buffer and the READS (read synchronously) command to pick up specific log control intervals from a buffer.

PSPI

You can write a program that uses IFI to capture log records while DB2 is running. You can read the records asynchronously, by starting a trace that reads the log records into a buffer and then issuing an IFI call to read those records out of the buffer. Alternatively, you can read those log records synchronously, by using an IFI call that returns those log records directly to your IFI program.

Restriction: Either the primary or one of the secondary authorization IDs must have the MONITOR2 privilege. For details on how to code an IFI program, see

DB2 Performance Monitoring and Tuning Guide. PSPI

Reading log records into a buffer

To begin gathering active log records into a buffer, you use the START TRACE command

PSPI

Issuing the following START TRACE command in an IFI program causes DB2 to begin gathering active log records into a buffer:

```
-START TRACE(P) CLASS(30) IFCID(126) DEST(OPX)
```

where:

- P signifies to start a DB2 performance trace. Any of the DB2 trace types can be used.
- CLASS(30) is a user-defined trace class (31 and 32 are also user-defined classes).
- IFCID(126) activates DB2 log buffer recording.

- DEST(OPX) starts the trace to the next available DB2 online performance (OP) buffer. The size of this OP buffer can be explicitly controlled by the BUFSIZE keyword of the START TRACE command. Valid sizes range from 256 KB to 16 MB. The number must be evenly divisible by 4.

When the START TRACE command takes effect, from that point forward until DB2 terminates, DB2 begins writing 4-KB log buffer VSAM control intervals (CIs) to the OP buffer as well as to the active log. As part of the IFI COMMAND invocation, the application specifies an ECB to be posted and a threshold to which the OP buffer is filled when the application is posted to obtain the contents of the buffer.

The IFI READA request is issued to obtain OP buffer contents. 

Reading specific log records (IFCID 0129)

IFCID 129 can be used with an IFI READS request to return a specific range of log records from the active log into the return area your program has initialized.

 Enter the following command into your IFI program:

```
CALL DSNWLI(READS,ifca,return_area,ifcid_area,qual_area)
```

IFCID 129 must appear in the IFCID area.

To retrieve the log control interval, your program must initialize certain fields in the qualification area:

WQALLTYP

This is a 3-byte field in which you must specify CI (with a trailing blank), which stands for “control interval”.

WQALLMOD

In this 1-byte field, you specify whether you want the first log CI of the restarted DB2 subsystem, or whether you want a specific control interval as specified by the value in the RBA field.

- F** The “first” option is used to retrieve the first log CI of this DB2 instance. This option ignores any value in WQALLRBA and WQALLNUM.
- P** The “partial” option is used to retrieve partial log CIs for the log capture exit routine. DB2 places a value in field IFCAHLRS of the IFI communication area, as follows:
 - The RBA of the log CI given to the log capture exit routine, if the last CI written to the log was not full.
 - 0, if the last CI written to the log was full.

When you specify option P, DB2 ignores values in WQALLRBA and WQALLNUM.

- R** The “read” option is used to retrieve a set of up to 7 continuous log CIs. If you choose this option, you must also specify the WQALLRBA and WQALLNUM options, which the following text details.

WQALLRBA

In this 8-byte field, you specify the starting log RBA of the control intervals to be returned. This value must end in X'000' to put the address on a valid boundary. This field is ignored when using the WQALLMOD=F option.

If you specify an RBA that is not in the active log, reason code 00E60854 is returned in the field IFCARC2, and the RBA of the first CI of the active log is returned in field IFCAFCI of the IFCA. These 6 bytes contain the IFCAFCI field.

WQALLNUM

In this 2-byte field, specify the number of control intervals you want returned. The valid range is from X'0001' through X'0007', which means that you can request and receive up to seven 4-KB log control intervals. This field is ignored when using the WQALLMOD=F option. For a complete description of the qualification area, see *DB2 Performance Monitoring and Tuning Guide*.

If you specify a range of log CIs, but some of those records have not yet been written to the active log, DB2 returns as many log records as possible. You can find the number of CIs returned in field QWT02R1N of the self-defining section of the record. For information about interpreting trace output, see *DB2 Performance*

Monitoring and Tuning Guide. 

Related concepts

“Log capture routines” on page 807

Reading complete log data (IFCID 0306)

There are several benefits in using IFCID 0306 to read log data.

PSPI

- IFCID 0306 can request DB2 to decompress log records if compressed, before passing them to the return area of your IFI program.
- In a data sharing environment, DB2 merges log records if the value of the IFI READS qualification WQALFLTR is X'00'. If WQALFLTR is X'01', log records are not merged.
- IFCID can retrieve log records from the archive data sets.
- Complete log records are always returned.

To use this IFCID, use the same call as IFCID 0129. IFCID 0306 must appear in the IFCID area. IFCID 0306 returns complete log records and the spanned record indicators in bytes 2 will have no meaning, if present. Multi-segmented control

interval log records are combined for a complete log record. 

Related tasks

“Reading specific log records (IFCID 0129)” on page 736

Specifying the return area

You must specify the return area for IFCID 0306 requests

 For IFCID 0306 requests, your program's return area must reside in ECSA key 7 storage. The IFI application program must set the eye-catcher to 'I306' at offset 4 in the Return Area before making the IFCID 0306 call. There is an additional 60 byte area that must be included after the 4-byte length indicator and the 'I306' eye-catcher. This area is used by DB2 between successive application calls and must not be modified by the application. The return area mapping is documented later in this section.

The IFI application program needs to run in supervisor state to request the ECSA key 7 return area. The return area storage size need be a minimum of the largest

DB2 log record returned plus the additional area defined in DSNDQW04. Minimize the number of IFI calls required to get all log data but do not over use ECSA by the IFI program. The other IFI storage areas can remain in user storage key 8. The IFI application must be in supervisor state and key 0 when making IFCID 0306 calls.

IFCID 0306 has a unique return area format. The first section is mapped by QW0306OF instead of the write header DSNDQWIN. See *DB2 Performance Monitoring and Tuning Guide* for details. 

Qualifying log records

To retrieve IFCID 0306 log records, your program must initialize certain fields in the qualification area mapped by DSNDWQAL.

 These qualification fields are described here:

WQALLMOD

In this 1-byte field, specify one of the following modes:

- D** Retrieves the single log record whose RBA value and member id is specified in WQALLRBA. Issuing a D request while holding a position in the log, causes the request to fail and terminates the log position held.
- F** Used as a first call to request log records beyond the LRSN or RBA specified in WQALLRBA that meet the criteria specified in WQALLCRI.
- H** Retrieves the highest LRSN or log RBA in the active log. The value is returned in field IFCAHLRS of the IFI communications area (IFCA). There is no data returned in the return area and the return code for this call will indicate that no data was returned.
- N** Used following mode F or N calls to request any remaining log records that meet the criteria specified in WQALLCRI. * and any option specified in WQALLOPT. As many log records as fit in the program's return area are returned.
- T** Terminates the log position that was held by any previous F or N request. This allows held resources to be released.

Mode R is not used for IFCID 0306.

For both F or N requests, each log record returned contains a record-level feedback area recorded in QW0306L. The number of log records retrieved is in QW0306CT. The ending log RBA or LRSN of the log records to be returned is in QW0306ES.

WQALLRBA

In this 8-byte field, specify the starting log RBA or LRSN of the control records to be returned. For IFCID 0306, this is used on the "first" option (F) request to request log records beyond the LRSN or RBA specified in this field. Determine the RBA or LRSN value from the H request. For RBAs, the value plus one should be used. For IFCID 0306 with D request of WQALLMOD, the high-order 2 bytes must specify member id and the low order 6 bytes contain the RBA.

WQALLCRI

In this 1-byte field, indicate what types of log records you want:

X'00'

Tells DB2 to retrieve only log records for changed data capture and unit of recovery control.

X'FF'

Tells DB2 to retrieve all types of log records. Use of this option can retrieve large data volumes and degrade DB2 performance.

WQALLOPT

In this 1-byte field, indicate whether you want the returned log records to be decompressed.

X'01'

Tells DB2 to decompress the log records before they are returned.

X'00'

Tells DB2 to leave the log records in the compressed format.

A typical sequence of IFCID 0306 calls is:

WQALLMOD=H

This is only necessary if you want to find the current position in the log. The LRSN or RBA is returned in IFC AHLRS. The return area is not used.

WQALLMOD=F

The WQALLRBA, WQALLCRI and WQALLOPT should be set. If 00E60812 is returned, you have all the data for this scope. You should wait a while before issuing another WQALLMOD=F call. In data sharing, log buffers are flushed when the F request is issued.

WQALLMOD=N

If the 00E60812 has not been returned, you issue this call until it is. You should wait a while before issuing another WQALLMOD=F call.

WQALLMOD=T

This should only be used if you do not want to continue with the WQALLMOD=N before the end is reached. It has no use if a position is not held in the log.

PSPI

Reading log records with OPEN, GET, and CLOSE

This is a stand-alone method that can be used when DB2 is down. You use the assembler language macro DSNJSLR to submit OPEN, GET, and CLOSE functions. This method can be used to capture log records that you cannot pick up with IFI after DB2 goes down.

PSPI

DB2 provides three stand-alone log services that user-written application programs can use to read DB2 recovery log records and control intervals even when DB2 is not running.

- OPEN initializes stand-alone log services.
- GET returns a pointer to the next log record or log record control interval.
- CLOSE deallocates data sets and frees storage.

To invoke these services, use the assembler language macro, DSNJSLR, specifying one of the preceding functions.

These log services use a *request block*, which contains a feedback area in which information for all stand-alone log GET calls is returned. The request block is created when a stand-alone log OPEN call is made. The request block must be passed as input to all subsequent stand-alone log calls (GET and CLOSE). The request block is mapped by the DSNDSLRB macro and the feedback area is mapped by the DSNDSLRF macro.

When you issue an OPEN request, you can indicate whether you want to get log records or log record control intervals. Each GET request returns a single logical record or control interval depending on which you selected with the OPEN request. If neither is specified, the default, RECORD, is used. DB2 reads the log in the forward direction of ascending relative byte addresses or log record sequence numbers (LRSNs).

If a bootstrap data set (BSDS) is allocated before stand-alone services are invoked, appropriate log data sets are allocated dynamically by z/OS. If the bootstrap data set is not allocated before stand-alone services are invoked, the JCL for your user-written application to read a log must specify and allocate the log data sets to be read.

Important: Use one of the following methods to read active logs while the DB2 subsystem that owns the logs is active:

- IFCID 0129
- IFCID 0306
- Log capture exit

There are no restrictions on reading archive logs. 

JCL DD statements for DB2 stand-alone log services

Stand-alone services, such as OPEN, GET, and CLOSE, use a variety of JCL DD statements as they operate.

 The following tables list and describe the JCL DD statements that are used by stand-alone services.

Table 117. JCL DD statements for DB2 stand-alone log services

JCL DD statement	Explanation
BSDS	Specifies the bootstrap data set (BSDS). Optional. Another ddname can be used for allocating the BSDS, in which case the ddname must be specified as a parameter on the FUNC=OPEN. Using the ddname in this way causes the BSDS to be used. If the ddname is omitted on the FUNC=OPEN request, the processing uses DDNAME=BSDS when attempting to open the BSDS.
ARCHIVE	Specifies the archive log data sets to be read. Required if an archive data set is to be read and the BSDS is not available (the BSDS DD statement is omitted). Should not be present if the BSDS DD statement is present. If multiple data sets are to be read, specify them as concatenated data sets in ascending log RBA order.

Table 117. JCL DD statements for DB2 stand-alone log services (continued)

JCL DD statement	Explanation
ACTIVE n	(Where n is a number from 1 to 7). Specifies an active log data set that is to be read. Should not be present if the BSDS DD statement is present. If only one data set is to be read, use ACTIVE1 as the ddname. If multiple active data sets are to be read, use DDNAMEs ACTIVE1, ACTIVE2, ... ACTIVE n to specify the data sets. Specify the data sets in ascending log RBA order with ACTIVE1 being the lowest RBA and ACTIVE n being the highest.

Table 118. JCL DD statements for DB2 stand-alone log services in a data-sharing environment

JCL DD statement	Explanation
GROUP	<p>If you are reading logs from every member of a data sharing group in LRSN sequence, you can use this statement to locate the BSDSs and log data sets needed. You must include the data set name of one BSDS in the statement. DB2 can find the rest of the information from that one BSDS.</p> <p>All members' logs and BSDS data sets must be available. If you use this DD statement, you must also use the LRSN and RANGE parameters on the OPEN request. The GROUP DD statement overrides any MxxBSDS statements that are used.</p> <p>(DB2 searches for the BSDS DD statement first, then the GROUP statement, and then the MxxBSDS statements. If you want to use a particular member's BSDS for your own processing, you must call that DD statement something other than BSDS.)</p>
MxxBSDS	<p>Names the BSDS data set of a member whose log must participate in the read operation and whose BSDS is to be used to locate its log data sets. Use a separate MxxBSDS DD statement for each DB2 member. xx can be any two valid characters.</p> <p>Use these statements if logs from selected members of the data sharing group are required and the BSDSs of those members are available. These statements are ignored if you use the GROUP DD statement.</p> <p>For one MxxBSDS statement, you can use either RBA or LRSN values to specify a range. If you use more than one MxxBSDS statement, you must use the LRSN to specify the range.</p>
MyyARCHV	<p>Names the archive log data sets of a member to be used as input. yy can be any two valid characters that do not duplicate any xx used in an MxxBSDS DD statement.</p> <p>Concatenate all required archived log data sets of a given member in time sequence under one DD statement. Use a separate MyyARCHV DD statement for each member. You must use this statement if the BSDS data set is unavailable or if you want only some of the log data sets from selected members of the group.</p> <p>If you name the BSDS of a member by a MxxBSDS DD statement, do not name the log of the same member by an MyyARCHV statement. If both MyyARCHV and MxxBSDS identify the same log data sets, the service request fails. MyyARCHV statements are ignored if you use the GROUP DD statement.</p>

Table 118. JCL DD statements for DB2 stand-alone log services in a data-sharing environment (continued)

JCL DD statement	Explanation
<i>MyyACTn</i>	<p>Names the active log data set of a member to be used as input. <i>yy</i> can be any two valid characters that do not duplicate any <i>xx</i> used in an <i>MxxBSDS</i> DD statement. Use the same characters that identify the <i>MyyARCHV</i> statement for the same member; do <i>not</i> use characters that identify the <i>MyyARCHV</i> statement for any other member. <i>n</i> is a number from 1 to 16. Assign values of <i>n</i> in the same way as for <i>ACTIVEn</i> DD statements.</p> <p>You can use this statement if the BSDS data sets are unavailable or if you want only some of the log data sets from selected members of the group.</p> <p>If you name the BSDS of a member by a <i>MxxBSDS</i> DD statement, do not name the log of the same member by an <i>MyyACTn</i> statement. <i>MyyACTn</i> statements are ignored if you use the <i>GROUP</i> DD statement.</p>

The DD statements must specify the log data sets in ascending order of log RBA (or LRSN) range. If both *ARCHIVE* and *ACTIVEn* DD statements are included, the first archive data set must contain the lowest log RBA or LRSN value. If the JCL specifies the data sets in a different order, the job terminates with an error return code with a GET request that tries to access the first record breaking the sequence. If the log ranges of the two data sets overlap, this is not considered an error; instead, the GET function skips over the duplicate data in the second data set and returns the next record. The distinction between out-of-order and overlap is as follows:

- An out-of-order condition occurs when the log RBA or LRSN of the first record in a data set is greater than that of the first record in the following data set.
- An overlap condition occurs when the out-of-order condition is not met but the log RBA or LRSN of the last record in a data set is greater than that of the first record in the following data set.

Gaps within the log range are permitted. A gap is created when one or more log data sets containing part of the range to be processed are not available. This can happen if the data set was not specified in the JCL or is not reflected in the BSDS. When the gap is encountered, an exception return code value is set, and the next complete record after the gap is returned.

Normally, the BSDS DD name is supplied in the JCL, rather than a series of *ACTIVE* DD names or a concatenated set of data sets for the *ARCHIVE* ddname.

This is commonly referred to as “running in BSDS mode”. 

Related reference

“Stand-alone log OPEN request” on page 744

“Stand-alone log GET request” on page 745

“Stand-alone log CLOSE request” on page 747

Data sharing members that participate in a read

The number of members whose logs participate in a particular read request varies based on what statements are used.

PSPI If you use the GROUP DD statement, then the determinant is the number of members in the group. Otherwise, the number of different *xxs* and *yys* used in the *Mxx* and *Myy* type DD statements.

For example, assume you need to read log records from members S1, S2, S3, S4, S5 and S6.

- S1 and S2 locate their log data sets by their BSDSs.
- S3 and S4 need both archive and active logs.
- S4 has two active log data sets.
- S5 needs only its archive log.
- S6 needs only one of its active logs.

You then need the following DD statements to specify the required log data sets:

MS1BSDS	MS2BSDS	MS3ARCHV MS3ACT1	MS4ARCHV MS4ACT1 MS4ACT2	MS5ARCHV	MS6ACT1
---------	---------	---------------------	--------------------------------	----------	---------

The order of the DD statements in the JCL stream is not important. **PSPI**

Registers and return codes

DB2 uses registers and return codes to store important information that can help you determine the status of stand-alone log activity.

PSPI The request macro invoking these services can be used by reentrant programs. The macro requires that register 13 point to an 18-word save area at invocation. In addition, registers 0, 1, 14, and 15 are used as work and linkage registers. A return code is passed back in register 15 at the completion of each request. When the return code is nonzero, a reason code is placed in register 0. Return codes identify a class of errors, while the reason code identifies a specific error condition of that class. The stand-alone log return codes are shown in the following table.

Table 119. Stand-alone log return codes

Return code	Explanation
0	Successful completion.
4	Exception condition (for example, end of file), not an error. This return code is not applicable for OPEN and CLOSE requests.
8	Unsuccessful completion due to improper user protocol.
12	Unsuccessful completion. Error encountered during processing of a valid request.

The stand-alone log services invoke executable macros that can execute only in 24-bit addressing mode and reference data below the 16-MB line. User-written applications should be link-edited as AMODE(24), RMODE(24). **PSPI**

Stand-alone log OPEN request

A stand-alone log OPEN request initializes the stand-alone log services.

PSPI The syntax for the stand-alone log OPEN request is:

```
{label} DSNJSLR  FUNC=OPEN
                ,LRSN=YES3NO
                ,DDNAME= address or (Reg. 2-12) optional
                ,RANGE= address or (Reg. 2-12) optional
                ,PMO=CI or RECORD
```

Keyword

Explanation

FUNC=OPEN

Requests the stand-alone log OPEN function.

LRSN

Tells DB2 how to interpret the log range:

NO: the log range is specified as RBA values. This is the default.

YES: the log range is specified as LRSN values.

DDNAME

Specifies the address of an 8-byte area which contains the ddname to be used as an alternate to a ddname of the BSDS when the BSDS is opened, or a register that contains that address.

RANGE

Specifies the address of a 12-byte area containing the log range to be processed by subsequent GET requests against the request block generated by this request, or a register that contains that address.

If LRSN=NO, then the range is specified as RBA values. If LRSN=YES, then the range is specified as LRSN values.

The first 6 bytes contain the low RBA or LRSN value. The first complete log record with an RBA or LRSN value equal to or greater than this value is the record accessed by the first log GET request against the request block. The last 6 bytes contain the end of the range or high RBA or LRSN value. An end-of-data condition is returned when a GET request tries to access a record with a starting RBA or LRSN value greater than this value. A value of 6 bytes of X'FF' indicates that the log is to be read until either the end of the log (as specified by the BSDS) or the end of the data in the last JCL-specified log data set is encountered.

If BSDS, GROUP, or MxxBSDS DD statements are used for locating the log data sets to be read, the RANGE parameter is required. If the JCL determines the log data sets to be read, the RANGE parameter is optional.

PMO Specifies the processing mode. You can use OPEN to retrieve either log records or control intervals in the same manner. Specify PMO=CI or RECORD, then use GET to return the data you have selected. The default is RECORD.

The rules remain the same regarding control intervals and the range specified for the OPEN function. Control intervals must fall within the range specified on the RANGE parameter.

Output

Explanation

GPR 1 General-purpose register 1 contains the address of a request block on

return from this request. This address must be used for subsequent stand-alone log requests. When no more log GET operations are required by the program, this request block should be used by a FUNC=CLOSE request.

GPR 15

General-purpose register 15 contains a return code upon completion of a request. For nonzero return codes, a corresponding reason code is contained in register 0.

GPR 0 General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

Log control interval retrieval

You can use the PMO option to retrieve log control intervals from archive log data sets. DSNJSLR also retrieves log control intervals from the active log if the DB2 system is not active. During OPEN, if DSNJSLR detects that the control interval range is not within the archive log range available (for example, the range purged from BSDS), an error condition is returned.

Specify CI and use GET to retrieve the control interval you have chosen. The rules remain the same regarding control intervals and the range specified for the OPEN function. Control intervals must fall within the range specified on the RANGE parameter.

Log control interval format

A field in the last 7 bytes of the control interval, offset 4090, contains a 7-byte timestamp. This field reflects the time at which the control interval was written to the active log data set. The timestamp is in store clock (STCK) format and is the high-order 7 bytes of the 8-byte store clock value. 

Related reference

“JCL DD statements for DB2 stand-alone log services” on page 740

“Registers and return codes” on page 743

Stand-alone log GET request

A stand-alone log GET request returns a pointer to a buffer containing the next log record based on position information in the request block.

 A log record is available in the area pointed to by the request block until the next GET request is issued. At that time, the record is no longer available to the requesting program. If the program requires reference to a log record's content after requesting a GET of the next record, the program must move the record into a storage area that is allocated by the program.

The first GET request, after a FUNC=OPEN request that specified a RANGE parameter, returns a pointer in the request feedback area. This points to the first record with a log RBA value greater than or equal to the low log RBA value specified by the RANGE parameter. If the RANGE parameter was not specified on the FUNC=OPEN request, then the data to be read is determined by the JCL specification of the data sets. In this case, a pointer to the first complete log record in the data set that is specified by the ARCHIVE, or by ACTIVE1 if ARCHIVE is omitted, is returned. The next GET request returns a pointer to the next record in ascending log RBA order. Subsequent GET requests continue to move forward in

log RBA sequence until the function encounters the end of RANGE RBA value, the end of the last data set specified by the JCL, or the end of the log as determined by the bootstrap data set.

The syntax for the stand-alone log GET request is:

```
{label} DSNJSLR  FUNC=GET
                ,RBR=(Reg. 1-12)
```

Keyword

Explanation

FUNC=GET

Requests the stand-alone log GET function.

RBR Specifies a register that contains the address of the request block this request is to use. Although you can specify any register between 1 and 12, using register 1 (RBR=(1)) avoids the generation of an unnecessary load register and is therefore more efficient. The pointer to the request block (that is passed in register *n* of the RBR=(*n*) keyword) must be used by subsequent GET and CLOSE function requests.

Output

Explanation

GPR 15

General-purpose register 15 contains a return code upon completion of a request. For nonzero return codes, a corresponding reason code is contained in register 0.

GPR 0 General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

Reason codes 00D10261 - 00D10268 reflect a damaged log. In each case, the RBA of the record or segment in error is returned in the stand-alone feedback block field (SLRFRBA). A damaged log can impair DB2 restart; special recovery procedures are required for these circumstances.

Information about the GET request and its results is returned in the request feedback area, starting at offset X'00'. If there is an error in the length of some record, the control interval length is returned at offset X'0C' and the address of the beginning of the control interval is returned at offset X'08'.

On return from this request, the first part of the request block contains the feedback information that this function returns. Mapping macro DSNDSLRF defines the feedback fields which are shown in the following table. The information returned is status information, a pointer to the log record, the length of the log record, and the 6-byte log RBA value of the record.

Table 120. Stand-alone log get feedback area contents

Field name	Hex offset	Length (bytes)	Field contents
SLRFRFC	00	2	Log request return code
SLRFINFO	02	2	Information code returned by dynamic allocation. Refer to the z/OS SPF job management publication for information code descriptions
SLRFERCD	04	2	VSAM or dynamic allocation error code, if register 15 contains a nonzero value.
SLRFRG15	06	2	VSAM register 15 return code value.

Table 120. Stand-alone log get feedback area contents (continued)

Field name	Hex offset	Length (bytes)	Field contents
SLRFFRAD	08	4	Address of area containing the log record or CI
SLRFRCLL	0C	2	Length of the log record or RBA
SLRFRBA	0E	6	Log RBA of the log record
SLRFDNM	14	8	ddname of data set on which activity occurred

PSPI

Related tasks

Chapter 20, “Recovering from different DB2 for z/OS problems,” on page 597

Related reference

“JCL DD statements for DB2 stand-alone log services” on page 740

“Registers and return codes” on page 743

“X'D1.....’ codes” (DB2 Codes)

Stand-alone log CLOSE request

A stand-alone log CLOSE request deallocates any log data sets that were dynamically allocated by previous processing. In addition, all storage that was obtained by previous functions, including the request block specified on the request, is freed.

PSPI

The syntax for the stand-alone log CLOSE request is:

```
{label} DSNJSLR  FUNC=CLOSE
                ,RBR=(Reg. 1-12)
```

Keyword

Explanation

FUNC=CLOSE

Requests the CLOSE function.

RBR Specifies a register that contains the address of the request block that this function uses. Although you can specify any register between 1 and 12, using register 1 (RBR=(1)) avoids the generation of an unnecessary load register and is therefore more efficient.

Output

Explanation

GPR 15

Register 15 contains a return code upon completion of a request. For nonzero return codes, a corresponding reason code is contained in register 0.

GPR 0

Register 0 contains a reason code that is associated with a nonzero return code that is contained in register 15. The only reason code used by the CLOSE function is 00D10030.

PSPI

Related reference

“JCL DD statements for DB2 stand-alone log services” on page 740

"Registers and return codes" on page 743

"00D10030" (DB2 Codes)

Sample application that uses stand-alone log services

This sample application includes various stand-alone log calls.

PSPI The following excerpt shows sample segments of an assembler program that uses the three stand-alone log services (OPEN, GET, and CLOSE) to process one log record.

```

TSTJSLR5 CSECT
:
:
OPENCALL EQU *
          LA R2,NAME          GET BSDS DDNAME ADDRESS
          LA R3,RANGER        GET ADDRESS OF RBA RANGE
          DSNJSLR FUNC=OPEN,DDNAME=(R2),RANGE=(R3)
          LTR R15,R15         CHECK RETURN CODE FROM OPEN
          BZ   GETCALL        OPEN OK, DO GET CALLS
:
:
*****
*      HANDLE ERROR FROM OPEN FUNCTION AT THIS POINT      *
*****
:
GETCALL  EQU *
        DSNJSLR FUNC=GET,RBR=(R1)
        C   R0,=X'00D10020'   END OF RBA RANGE ?
        BE  CLOSE             YES, DO CLEANUP
        C   R0,=X'00D10021'   RBA GAP DETECTED ?
        BE  GAPRTN            HANDLE RBA GAP
        LTR R15,R15           TEST RETURN CODE FROM GET
        BNZ ERROR
:
:
:
*****
*      PROCESS RETURNED LOG RECORD AT THIS POINT. IF LOG RECORD *
*      DATA MUST BE KEPT ACROSS CALLS, IT MUST BE MOVED TO A *
*      USER-PROVIDED AREA.                                     *
*****
          USING SLRF,1        BASE SLRF DSECT
          L   R8,SLRFFRAD     GET LOG RECORD START ADDR
          LR  R9,R8
          AH  R9,SLRFRCLL     GET LOG RECORD END ADDRESS
          BCTR R9,R0
:
:
CLOSE   EQU *
        DSNJSLR FUNC=CLOSE,RBR=(1)
:
:
NAME    DC   C'DDBSDS'
RANGER  DC   X'000000000000000000000005FFFF'
:
:
        DSNDSLRLB
        DSNDSLRLF
        EJECT
R0      EQU  0
R1      EQU  1
R2      EQU  2
:
:
R15     EQU  15
        END

```

Figure 78. Excerpts from a sample program using stand-alone log services



Reading log records with the log capture exit routine

This is an online method using the log capture exit when DB2 is running. You write an exit routine to use this exit to capture and transfer log records in real time.

PSPI You can use the log capture exit routine to capture DB2 log data in real time. This installation exit routine presents log data to a log capture exit routine when the data is written to the DB2 active log. This exit routine is not intended to be used for general purpose log auditing or tracking. The IFI interface is designed (and is more appropriate) for this purpose.

The log capture exit routine executes in an area of DB2 that is critical for performance. As such, it is primarily intended as a mechanism to capture log data for recovery purposes. In addition, the log capture exit routine operates in a very restrictive z/OS environment, which severely limits its capabilities as a stand-alone routine.

To capture log records with this exit routine, you must first write an exit routine (or use the one provided by the preceding program offering) that can be loaded and called under the various processing conditions and restrictions required of this exit routine. **PSPI**

Related concepts

“Contents of the log” on page 723

“Log capture routines” on page 807

Related tasks

“Reading log records with IFI” on page 735

Related reference

“The physical structure of the log” on page 728

Part 4. Appendixes

Appendix A. Writing exit routines

DB2 provides installation-wide exit points to the routines that you provide. These exit routines are described in the following sections:

- “Connection routines and sign-on routines”
- “Access control authorization exit routine” on page 764
- “Edit routines” on page 781
- “Validation routines” on page 785
- “Date and time routines” on page 788
- “Conversion procedures” on page 792
- “Field procedures” on page 795
- “Log capture routines” on page 807
- “Routines for dynamic plan selection in CICS” on page 809
- “Routine for CICS transaction invocation stored procedure” on page 810.

Connection routines and sign-on routines

Your DB2 subsystem has two exit points for authorization routines, one in connection processing and one in sign-on processing. Both exit points perform crucial steps in the assignment of values to primary IDs, secondary IDs, and SQL IDs.

PSPI You must have a routine for each exit. Default routines are provided for both. DSN3@ATH is the default exit routine for connections, and DSN3@SGN is the default exit routine for sign-ons.

If your installation has a connection exit routine and you plan to use CONNECT with the USER/USING clause, you should examine your exit routine and take the following into consideration. DB2 does not update the following to reflect the user ID and password that are specified in the USER/USING clause of the CONNECT statement:

- The security-related control blocks that are normally associated with the thread
- The address space that your exit routine can access

This topic describes the interfaces for those routines and the functions that they provide. If you want to use secondary authorization IDs, you must replace the default routines with the sample routines, or with routines of your own.

“General guidelines for writing exit routines” on page 810 applies to these routines. One exception to the description of execution environments is that the routines execute in non-cross-memory mode. **PSPI**

Related tasks

“Using sample connection and sign-on exit routines for CICS transactions” on page 273

“Specifying connection and sign-on routines” on page 754

“Debugging connection and sign-on routines” on page 762

Related reference

“Processing of connection requests” on page 268

“Processing of sign-on requests” on page 270

“Sample connection and sign-on routines”

“Exit parameter list for connection and sign-on routines” on page 756

Specifying connection and sign-on routines

Your connection routine must have a CSECT name and entry point of DSN3@ATH. The connection routine’s load module name can be the same, or it can be a different name. Your sign-on routine must have a CSECT name and entry point of DSN3@SGN. The sign-on routine’s load module name can be the same, or it can be a different name.

PSPI You can use an ALIAS statement of the linkage editor to provide the entry-point name.

Default routines exist in library *prefix*.SDSNLOAD. To use your routines instead, place your routines in library *prefix*.SDSNEXIT. You can use the install job DSNTIJEX to assemble and link-edit the routines and place them in the new library. If you use any other library, you might need to change the STEPLIB or JOBLIB concatenations in the DB2 start-up procedures.

You can combine both routines into one CSECT and load module if you wish, but the module must include both entry points, DSN3@ATH and DSN3@SGN. Use standard assembler and linkage editor control statements to define the entry points. DB2 loads the module twice at startup, by issuing the z/OS LOAD macro first for entry point DSN3@ATH and then for entry point DSN3@SGN. However, because the routines are reentrant, only one copy of each remains in virtual storage. **PSPI**

Related concepts

“Connection routines and sign-on routines” on page 753

Related reference

“Processing of connection requests” on page 268

“Processing of sign-on requests” on page 270

“Sample connection and sign-on routines”

“Exit parameter list for connection and sign-on routines” on page 756

Sample connection and sign-on routines

The sample exit routines provide examples of the functions and interfaces that are described in this topic. These exit routines are provided in source code as members of *prefix*.SDSNSAMP.

PSPI To examine the sample connection routine, list or assemble member DSN3SATH. To examine the sample sign-on routine, list or assemble member DSN3SSGN. Because both routines use features that are not available in Assembler XF, you must use Assembler H to assemble them.

Change required for some CICS users: You must change the sample sign-on exit routine (DSN3SSGN) before assembling and using it, if the following conditions are true:

- You attach to DB2 with an AUTH parameter other than AUTH=GROUP.
- You have the RACF list-of-groups option active.

- You have transactions whose initial primary authorization ID is not defined to RACF

To change the sample sign-on exit routine (DSN3SSGN), perform the following steps:

1. Locate the following statement in DSN3SSGN as a reference point:

```
SSGN035 DS OH BLANK BACKSCAN LOOP REENTRY
```

2. Locate the following statement, which comes after the reference point:

```
B SSGN037 ENTIRE NAME IS BLANK, LEAVE
```

3. Replace the statement with the following statement:

```
B SSGN090 NO GROUP NAME... BYPASS RACF CHECK
```

By changing the statement, you avoid an abend with SQLCODE -922. The routine with the new statement provides no secondary IDs unless you use AUTH=GROUP.

PSPI

Related concepts

“Connection routines and sign-on routines” on page 753

Related tasks

“Using sample connection and sign-on exit routines for CICS transactions” on page 273

“Specifying connection and sign-on routines” on page 754

“Debugging connection and sign-on routines” on page 762

When connection and sign-on routines are taken

Different local processes enter the access control procedure at different points, depending on the environment from which they originate. Different criteria apply to remote requests.

PSPI

The following processes go through connection processing only:

- Requests that originate in TSO foreground and background (including online utilities and requests through the call attachment facility)
- JES-initiated batch jobs
- Requests through started task control address spaces (from the MVS START command)

The following processes go through connection processing, and can later go through the sign-on exit:

- The IMS control region
- The CICS recovery coordination task
- DL/I batch
- Requests through the Resource Recovery Services attachment facility (RRSAF)

The following processes go through sign-on processing:

- Requests from IMS-dependent regions (including MPP, BMP, and Fast Path)
- CICS transaction subtasks. **PSPI**

EXPL for connection and sign-on routines

PSPI The following diagram shows how the parameter list points to other information.

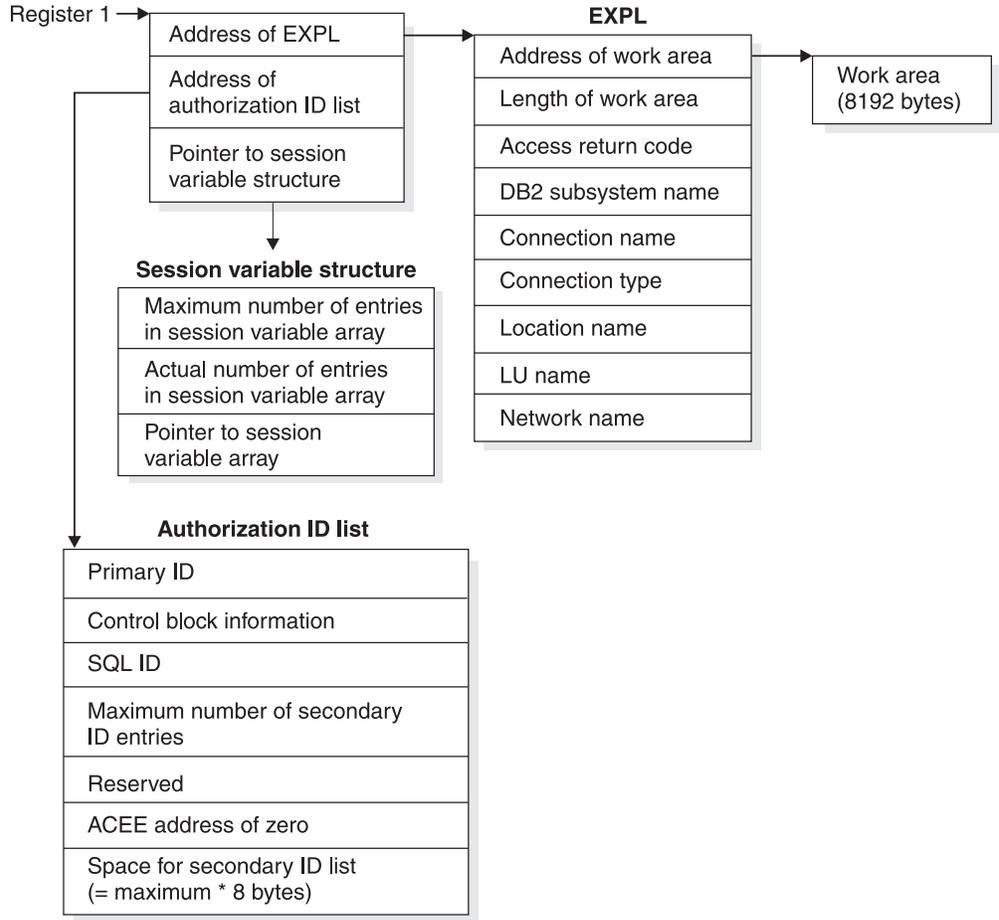


Figure 79. How a connection or sign-on parameter list points to other information

PSPI

Exit parameter list for connection and sign-on routines

Connection routines and sign-on routines use 28 more bytes of the exit parameter list EXPL than other routines. The following table shows the entire list of connection routines and sign-on routines. The exit parameter list is described by macro DSNDEXPL.

PSPI

Table 121. Exit parameter list for connection routines and sign-on routines

Name	Hex offset	Data type	Description
EXPLWA	0	Address	Address of a 8192-byte work area to be used by the routine.
EXPLWL	4	Signed 4-byte integer	Length of the work area, in bytes; value is 8192.

Table 121. Exit parameter list for connection routines and sign-on routines (continued)

Name	Hex offset	Data type	Description
EXPLRSV1	8	Signed 2-byte integer	Reserved.
EXPLRC1	A	Signed 2-byte integer	Not used.
EXPLRC2	C	Signed 4-byte integer	Not used.
EXPLARC	10	Signed 4-byte integer	Access return code. Values can be: 0 Access allowed; DB2 continues processing. 12 Access denied; DB2 terminates processing with an error.
EXPLSSNM	14	Character, 8 bytes	DB2 subsystem name, left justified; for example, 'DSN '.
EXPLCONN	1C	Character, 8 bytes	Connection name for requesting location.
EXPLTYPE	24	Character, 8 bytes	Connection type for requesting location. For DDF threads, the connection type is 'DIST '.
EXPLSITE	2C	Character, 16 bytes	For SNA protocols, this is the location name of the requesting location or <luname>. For TCP/IP protocols, this is the dotted decimal IP address of the requester.
EXPLLUNM	3C	Character, 8 bytes	For SNA protocols, this is the locally known LU name of the requesting location. For TCP/IP protocols, this is the character string 'TCPIP'.
EXPLNTID	44	Character, 17 bytes	For SNA protocols, the fully qualified network name of the requesting location. For TCP/IP protocols, this field is reserved.
EXPLVIDS			DB2 version identifier

PSPI

Related concepts

“Connection routines and sign-on routines” on page 753

Related tasks

“Using sample connection and sign-on exit routines for CICS transactions” on page 273

“Specifying connection and sign-on routines” on page 754

“Debugging connection and sign-on routines” on page 762

Authorization ID parameter list for connection and sign-on routines

The second parameter list, which is specific to connection routines and sign-on routines, is called an authorization ID list. Its contents are shown in the following table. The description is given by macro DSNDAIDL.

PSPI

Table 122. Authorization ID list for a connection or sign-on exit routine

Name	Hex offset	Data type	Description
AIDLPRIM	0	Character, 8 bytes	Primary authorization ID for input and output; see descriptions in the text.
AIDLCODE	8	Character, 2 bytes	Control block identifier.
AIDLTLEN	A	Signed 2-byte integer	Total length of control block.
AIDLEYE	C	Character, 4 bytes	Eyecatcher for block, "AIDL".
AIDLSQL	10	Character, 8 bytes	On output, the current SQL ID.
AIDLSCNT	18	Signed 4-byte integer	Number of entries allocated to secondary authorization ID list. Always equal to 1012.
AIDLSAPM	1C	Address	For a sign-on routine only, the address of an 8-character additional authorization ID. If RACF is active, the ID is the user ID's connected group name. If the address was not provided, the field contains zero.
AIDLCKEY	20	Character, 1 byte	Storage key of the ID pointed to by AIDLSAPM. To move that ID, use the "move with key" (MVCK) instruction, specifying this key.
AIDLRV1	21	Character, 3 bytes	Reserved
AIDLRV2	24	Signed 4-byte integer	Reserved
AIDLACEE	28	Signed 4-byte integer	The address of the ACEE structure, if known; otherwise, zero
AIDLRACL	2C	Signed 4-byte integer	Length of data area returned by RACF, plus 4 bytes
AIDLRACR	30	26 bytes	Reserved
AIDLSEC	4A	Character, maximum x 8 bytes	List of the secondary authorization IDs, 8 bytes each

PSPI

Input values for connection routines

The input values for a connection routine are as follows:

- **PSPI** The initial primary authorization ID for a local request can be obtained from the z/OS address space extension block (ASXB).

The ASXB contains at most only a seven-character value. That is always sufficient for a TSO user ID or a user ID from an z/OS JOB statement, and the ASXB is always used for those cases.

For CICS, IMS, or other started tasks, z/OS can also pass an eight-character ID. If an eight-character ID is available, and if its first seven characters agree with the ASXB value, then DB2 uses the eight-character ID. Otherwise it uses the ASXB value.

You can alter the sample exit routine to use the ASXB value always. For instructions, see “Processing in the sample connection and sign-on routines” on page 760.

If RACF is active, the field used contains a verified RACF user ID; otherwise, it contains blanks.

- The primary ID for a remote request is the ID passed in the conversation attach request header (SNA FMH5) or in the DRDA SECCHK command.
- The SQL ID contains blanks.
- The list of secondary IDs contains blanks. 

Input values for sign-on routines

The input values for a sign-on routine are as follows:

-  The initial primary ID depends on the sign-on method.
- The SQL ID and all secondary IDs contain blanks.
- Field AIDLSAPM in the authorization ID list can contain the address of an 8-character additional authorization ID, obtained by the CICS attachment facility using the RACROUTE REQUEST=EXTRACT service with the requester’s user ID. If RACF is active, this ID is the RACF-connected group name from the ACEE corresponding to the requester’s user ID. Otherwise, this field contains blanks. IMS does not pass this parameter.
- Field AIDLCKEY contains the storage key of the identifier pointed to by AIDLSAPM. To move that ID, use the “move with key” (MVCK) instruction, specifying this key.
- Field AIDLACEE contains the ACEE address only for a sign-on through the CICS attachment facility and only when the CICS RCT uses AUTH=GROUP.



Expected output for connection and sign-on routines

DB2 uses the output values of the primary ID, the SQL ID, and the secondary IDs. Your routines can set these IDs to any value that is an SQL short identifier.

 **Important:** If your identifier does not meet the 8-character criteria, the request fails. Therefore, when necessary, add blanks to the end of short identifiers to ensure that they meet the criteria.

If the values that are returned **are not blank**, DB2 interprets them in the following ways:

- The primary ID becomes the primary authorization ID.
- The list of secondary IDs, down to the first blank entry or to a maximum of 1012 entries, becomes the list of secondary authorization IDs. The space allocated for the secondary ID list is only large enough to contain the maximum number of authorization IDs. This number is in field AIDLSCNT.

Attention: If you allow more than 1012 secondary authorization IDs, abends and storage overlays can occur.

- The SQL ID is checked to see if it is the same as the primary or one of the secondary IDs. If it is not, the connection or sign-on process fails. Otherwise, the validated ID becomes the current SQL ID.

If the returned value of the primary ID is **blank**, DB2 takes the following steps:

- In connection processing, the default ID that is defined when DB2 is installed (UNKNOWN AUTHID on panel DSNTIPP) is substituted as the primary authorization ID and the current SQL ID. The list of secondary IDs is set to blanks.
- Sign-on processing abends. No default value exists for the primary ID.

If the returned value of the SQL ID is blank, DB2 makes it equal to the value of the primary ID. If the list of secondary IDs is blank, it remains blank. No default secondary IDs exist.

Your routine must also set a return code in word 5 of the exit parameter list to allow or deny access (field EXPLARC). By those means you can deny the connection altogether. The code must have one of the values that are shown in Table 123.

Table 123. Required return code in EXPLARC

Value	Meaning
0	Access allowed; continue processing.
12	Access denied; terminate.

Any other value will cause an abend. 

Processing in the sample connection and sign-on routines

The sample routines provided by IBM can serve as models for the processing that is required in connection routines and sign-on routines.

 **Recommendation:** Consider using the sample routines as a starting point when you write your own routines.

Both the sample connection routine (DSN3SATH) and the sample sign-on routine have similar sections for setup, constants, and storage areas. Both routines set values of the primary ID, the SQL ID, and the secondary IDs in three numbered sections.

In the sample connection routine (DSN3SATH): The three sections of the sample connection routine perform the following functions:

Section 1

Section 1 provides the same function as in the default connection routine. It determines whether the first character of the input primary ID has a value that is greater than blank (hex 40), and performs the following operations:

- If the first character is greater than hex 40, the value is not changed.
- If the first character is not greater than hex 40, the value is set according to the following rules:
 - If the request is from a TSO foreground address space, the primary ID is set to the logon ID.
 - If the request is not from a TSO foreground address space, the primary ID is set to the job user ID from the JES job control table.

- If no primary ID is located, Section 2 is bypassed.

Section 2

At the beginning of Section 2, you can restore one commented-out instruction, which then truncates the primary authorization ID to 7 characters. (The instruction is identified by comments in the code.)

Section 2 next tests RACF options and makes the following changes in the list of secondary IDs, which is initially blank:

- If RACF is not active, the list remains blank.
- If the list of groups option is not active, but an ACEE exists, the connected group name is copied as the only secondary ID.
- If the list of groups option is active, the list of group names from the ICHPCGRP block is copied into AIDLSEC in the authorization ID list.

Section 3

Section 3 performs the following steps:

1. The SQL ID is set equal to the primary ID.
2. If the TSO data set name prefix is a valid primary or secondary ID, the SQL ID is replaced with the TSO data set name prefix. Otherwise, the SQL ID remains set to the primary ID.

In the sample sign-on routine (DSN3SSGN): The three sections of the sample sign-on routine perform the following functions:

Section 1

Section 1 does not change the primary ID.

Section 2

Section 2 sets the SQL ID to the value of the primary ID.

Section 3

Section 3 tests RACF options and makes the following changes in the list of secondary IDs, which is initially blank:

- If RACF is not active, the list remains blank.
- If the list of groups option is active, section 3 attempts to find an existing ACEE from which to copy the authorization ID list.
 - If AIDLACEE contains a valid ACEE, it is used.
Otherwise, look for a valid ACEE chained from the TCB or from the ASXB or, if no usable ACEE exists, issue RACROUTE to have RACF build an ACEE structure for the primary ID.
Copy the list of group names from the ACEE structure into the secondary authorization list.
 - If the exit issued RACROUTE to build an ACEE, another RACROUTE macro is issued and the structure is deleted.
- If a list of secondary authorization IDs has not been built, and AIDLAPM is not zero, the data that is pointed to by AIDLAPM is copied into AIDLSEC. 

Performance considerations for connection and sign-on routines

Your sign-on exit routine is part of the critical path for transaction processing in IMS and CICS, so you want it to execute as quickly as possible.

PSPI Avoid writing SVC calls like GETMAIN, FREEMAIN, and ATTACH. Also avoid I/O operations to any data set or database. To improve performance, you might be able to delete the list of groups that process in Section 3 of the sample sign-on exit routine.

The sample sign-on exit routine can issue the RACF RACROUTE macro with the default option SMC=YES. If another product issues RACROUTE with SMC=NO, a deadlock might occur.

Your routine can also enhance the performance of later authorization checking. Authorization for dynamic SQL statements is checked first for the CURRENT SQLID, then for the primary authorization ID, and then for the secondary authorization IDs. If you know that a user's privilege most often comes from a secondary authorization ID, then set the CURRENT SQLID to this secondary ID within your exit routine. **PSPI**

Related concepts

“General guidelines for writing exit routines” on page 810

Debugging connection and sign-on routines

The diagnostic aids can assist you in debugging connection exit routines and sign-on exit routines.

PSPI *Subsystem support identify recovery:* The identify ESTAE recovery routine, DSN3IDES, generates the VRADATA entries that are shown in Table 124. The last entry, key VRAIMO, is generated only if the abend occurred within the connection exit routine.

Table 124. VRADATA entries that are generated by DSN3IDES

VRA keyname	Key hex value	Data length	Content
VRAFPI	22	8	Constant 'IDESTRAK'
VRAFP	23	24	<ul style="list-style-type: none"> 32-bit recovery tracking flags 32-bit integer AGNT block unique identifier AGNT block address AIDL block address Initial primary authorization ID as copied from ASXBUSER
VRAIMO	7C	10	<ul style="list-style-type: none"> Connection exit load module load point address Connection exit entry point address Offset of failing address in the PSW from the connection exit entry point address

Subsystem support sign-on recovery: The sign-on ESTAE recovery routine DSN3SIES generates the VRADATA entries that are shown in Table 125 on page 763. The last entry, key VRAIMO, is generated only if the abend occurred within the sign-on exit routine.

Table 125. VRADATA entries that are generated by DSN3SIES

VRA keyname	Key hex value	Data length	Content
VRAFPI	22	8	Constant 'SIESTRAK'
VRAFP	23	20	<ul style="list-style-type: none"> • Primary authorization ID (CCBUSER) • AGNT block address • Identify-level CCB block address • Sign-on-level CCB block address
VRAIMO	7C	10	<ul style="list-style-type: none"> • Sign-on exit load module load point address • Sign-on exit entry point address • Offset of failing address in the PSW from the sign-on exit entry point address

Diagnostics for connection exit routines and sign-on exit routines: The connection (identify) recovery routine and the sign-on recovery routine provide diagnostics for the corresponding exit routines. The diagnostics are produced only when the abend occurs in the exit routine. The following diagnostics are available:

Dump title

The component failing module name is "DSN3@ATH" for a connection exit or "DSN3@SGN" for a sign-on exit.

z/OS and RETAIN symptom data

SDWA symptom data fields SDWAC SCT (CSECT/) and SDWAMODN (MOD/) are set to "DSN3@ATH" or "DSN3@SGN", as appropriate.

Summary dump additions

The AIDL, if addressable, and the SADL, if present, are included in the summary dump for the failing allied agent. If the failure occurred in connection or sign-on processing, the exit parameter list (EXPL) is also included. If the failure occurred in the system services address space, the entire SADL storage pool is included in the summary dump. 

Related concepts

"Connection routines and sign-on routines" on page 753

Related reference

"Processing of connection requests" on page 268

"Processing of sign-on requests" on page 270

"Sample connection and sign-on routines" on page 754

"Exit parameter list for connection and sign-on routines" on page 756

Session variables in connection and sign-on routines

DB2 supplies default session variables. In addition, the connection exit routine and the sign-on exit routine support up to 10 more session variables. You can define these additional session variables and use them to provide information to applications by using the GETVARIABLE function.



The session variable structure: The connection exit routine and the sign-on exit routine point to the session variable structure (DSNDSVS). DSNDSVS specifies the maximum number of entries in the session array, the actual number of entries in the session array, and a pointer to the session variable array. The default value for the actual number of session variables is zero.

Defining session variables: To define session variables, use the session variable array (DSNDSVA) to list up to 10 session variables as name and value pairs. The session variables that you establish in the connection exit routine and the sign-on exit routine are defined in the SESSION schema. The values that the exit routine supplies in the session variable array replace the previous values.

Example: The session variable array that is shown in Table 126 lists six session variables.

Table 126. Sample session variable array

Name	Value
default_database	DATAXM
default_driver	PZN4Y7
location	Kyoto
member_of	GROUP_42
filename	report.txt
account_number	A1-X142783

The unqualified names are defined as VARCHAR(128), and the values are defined as VARCHAR(255). The exit routines must provide these values in Unicode CCSID 1208. 

For more information about the default session variables that DB2 supplies, see *DB2 SQL Reference*

Access control authorization exit routine

You can provide your own access control authorization exit routine by using an exit point that DB2 provides. Alternatively, after you carefully consider several important factors, you might choose to let RACF perform DB2 authorization checking for you.

 PSPI

Is the access control authorization exit routine right for you?

Using the RACF (Security Server for z/OS) to perform access control is not the best choice for every customer. Consider the following points before choosing RACF to perform access control:

- If you want the database administrators to manage security, integration with DB2 is very important. Using RACF access control provides less integration with DB2. In most of these cases, DB2 authorization provides advantages.
- If you want security administrators to manage security, integration with the security server is more important. In most of these cases, using RACF for access control provides advantages. Furthermore, if you want a security group to define authorization and a centralized security control point, RACF access control is an excellent match.

If you change from DB2 authorization to RACF access control, you must change to RACF methods for some authorization techniques, and you must understand how DB2 and RACF work together. Expect to make the following changes when you implement RACF access control:

- Plan to use RACF facilities (such as groups and patterns) more.
- Plan to use patterns instead of individual item access profiles and permissions.
- Plan to use RACF groups instead of secondary authorization IDs, which are not implemented in RACF. OWNER(secondaryID) generally must be a valid group.
- Find an alternative to BINDAGENT. BINDAGENT is based on secondary authorization IDs, which are not implemented in RACF. BINDAGENT provides a relatively weak security separation. Customers have found alternatives.
- Understand how SET CURRENT SQLID works with RACF. SET CURRENT SQLID can set a qualifier, but does not change authorization.
- Know that authorizations are not dropped when objects are dropped or renamed.
- Be aware of the relationship between objects and revoked privileges. Plans and packages are not invalidated when authorizations are revoked. Views are not dropped when authorizations are revoked.

How the access control authorization routine works

Your routine specifies whether the authorization checking should all be done by RACF only, or by both RACF and DB2. (Also, the routine can be called and still let all checking be performed by DB2.)

When DB2 invokes the routine, it passes three possible functions to the routine:

- Initialization (DB2 startup)
- Authorization check
- Termination (DB2 shutdown)

The bulk of the work in the routine is for authorization checking. When DB2 must determine the authorization for a privilege, it invokes your routine. The routine determines the authorization for the privilege and then indicates to DB2 whether the privilege is authorized or not authorized, or whether DB2 should do its own authorization check, instead.

When you write an access control authorization routine, use the general guidelines for writing exit routines, with the following exceptions to the environment description:

- The routine executes in non-cross-memory mode during initialization and termination (XAPLFUNC of 1 or 3).
- During authorization checking, the routine can execute under a TCB or SRB in cross-memory or non-cross-memory mode.

Bypass of the access control authorization routine

In the following situations, the access control authorization routine is not called to check authorization:

- The authorization ID that DB2 uses to determine access has installation SYSADM or installation SYSOPR authority (where installation SYSOPR authority is sufficient to authorize the request). This authorization check is made strictly within DB2. For example, if the execute privilege is being checked on a package, DB2 performs the check on the plan owner that this package is in. If the plan owner has installation SYSADM, the routine is not called.
- DB2 security has been disabled. (You can disable DB2 security by specifying NO on the USE PROTECTION field of installation panel DSNTIPP).
- Authorization has been cached from a prior check.

- In a prior invocation of the exit routine, the routine indicated that it should not be called again.
- GRANT statements.

The routine executes in the *ssnm*DBM1 address space of DB2.



Related reference

“Parameter list for the access control authorization routine” on page 770

Related information

[DB2 RACF Access Control Module Guide](#)

Specifying the access control authorization routine

Your access control authorization routine must have a CSECT name and an entry point of DSNX@XAC. The load module name or alias name must also be DSNX@XAC. A default routine with this name and entry point exists in library *prefix*.SDSNLOAD.



To use your routine instead, place it in the *prefix*.SDSNEXIT library. Use installation job DSNTIJEX to assemble and link-edit the routine and to place it in the *prefix*.SDSNEXIT library. If you use any other library, you might need to change the STEPLIB or JOBLIB concatenations in the DB2 start-up procedures.

The source code for the default routine is in *prefix*.SDSNSAMP as DSNXSXAC. You can use it to write your own exit routine. To assemble it, you must use Assembler H.

RACF provides a sample exit routine DSNRXAC, which is shipped with DB2. It can be found in *prefix*.SDSNSAMP.

The default access control authorization routine

The default exit routine returns a code to the DB2 authorization module. The code indicates that a user-defined access control authorization exit routine is not available. DB2 then performs normal authorization checking and does not attempt to invoke this exit routine again.

When the access control authorization routine is taken

The access control authorization routine is taken in the following three instances:



At DB2 startup

This exit routine is taken when DB2 starts to allow the external authorization checking application to perform any required setup prior to authorization checking. For example, loading authorization profiles into storage is a required setup task. DB2 uses the reason code that the exit routine sets during startup to determine how to handle exception situations.

When an authorization check is to be performed on a privilege

This exit routine is taken when DB2 accesses security tables in the catalog

to check authorization on a privilege. The exit routine is taken only if none of the prior invocations have indicated that the exit routine must not be called again.

At DB2 shutdown

This exit routine is taken when DB2 is stopping, to let the external authorization checking application perform its cleanup before DB2 stops.



Considerations for the access control authorization routine

This topic discusses the following special considerations for using the access control authorization exit routine:

Related concepts

“General guidelines for writing exit routines” on page 810

When DB2 cannot provide an ACEE

Sometimes DB2 cannot provide an ACEE. For example, if you do not use external security in CICS (that is, SEC=NO is specified in the DFHSIT), CICS does not pass an ACEE to the CICS attachment facility. The ACEE address is passed for CICS transactions, if available.



When DB2 does not have an ACEE, it passes zeros in the XAPLACEE field. If this happens, your routine can return a 4 in the EXPLRC1 field, and let DB2 handle the authorization check.

DB2 does not pass the ACEE address for IMS transactions. The ACEE address is passed for CICS transactions, if available.

DB2 does pass the ACEE address when it is available for DB2 commands that are issued from a logged on z/OS console. DB2 does not pass the ACEE address for DB2 commands that are issued from a console that is not logged on, or for the START DB2 command, or commands issued automatically during DB2 startup.



Authorization IDs and ACEEs

XAPL has two authorization ID fields, XAPLUPRM (the primary authorization ID) and XAPLUCHK (the authorization ID that DB2 uses to perform the authorization). These two fields might have different values.



The ACEE passed in XAPLACEE is that of the primary authorization ID, XAPLUPRM.

The implications of the XAPLUPRM and XAPLUCHK relationship need to be clearly understood. XAPLUCHK, the authorization ID that DB2 uses to perform authorization may be the primary authorization ID (XAPLUPRM), a secondary authorization ID, or another authorization ID such as a package owner.

If the RACF access control module is used, the following rules apply:

- RACF uses the ACEE of the primary authorization ID (XAPLUPRM) to perform authorization.
- Secondary authorization IDs are not implemented in RACF. RACF groups should be used instead.

Examples: The following examples show how the rules apply:

- A plan or package may be bound successfully by using the privileges of the binder (XAPLUPRM). Then only the EXECUTE privilege on the plan or package is needed to execute it. If at some point this plan or package is marked invalid (for instance, if a table it depends upon is dropped and recreated), the next execution of it will cause an AUTOBIND, which will usually fail. It fails because the AUTOBIND checks the privilege on the runner. The plan or package must be rebound by using an authorization ID that has the necessary privileges.
- If the OWNER on the BIND command is based on secondary authorization IDs, which are not supported by RACF. RACF groups should be used instead.
- SET CURRENT SQLID can set a qualifier, but it cannot change authorization.
- DYNAMIC RULES settings have a limited effect on which authorization ID is checked. Only the primary authorization ID and secondary IDs that are valid RACF groups for this user are considered.
- User-defined function and stored procedure authorizations involve several authorization IDs, such as implementer, definer, invoker, and so forth. Only the primary authorization ID and secondary IDs that are RACF groups are considered. 

Stored procedure authorization

DB2 checks the EXECUTE privilege on stored procedure packages at run time. The only authorization ID that is available to check at run time is the primary authorization ID that runs the stored procedure package.

 If your security plan requires that you do not grant the EXECUTE privilege to all authorization IDs, the ID that runs the stored procedure package might not have it.

However, you can ensure that the ID runs the package without explicitly granting the EXECUTE privilege on the stored procedure package. Instead, you can specify GRANT EXECUTE ON PACKAGE to the ID in the OWNER column of SYSIBM.SYSROUTINES for each package that is invoked during the stored procedure execution:

- The stored procedure definition must not specify COLLID.
- The program that calls the stored procedure must be bound directly to the plan that calls it by using the MEMBER bind option and by specifying the name of the calling program. The stored procedure package must be referenced in the PKLIST as *collection-id.** or *collection-id.stored-procedure-package-id* with the VALIDATE(BIND) option.
- The SET CURRENT PACKAGESET statement must not be used before the stored procedure is called in the program. 

Invalid and inoperative plans and packages

In DB2, when a privilege that is required by a plan or package is revoked, the plan or package is invalidated. DB2 can automatically rebound an invalidated plan or package if proper privileges are granted.

 However, if you use an authorization access control routine, it cannot tell DB2 that a privilege is revoked. Therefore, DB2 cannot know to invalidate the plan or package.

If the revoked privilege is the EXECUTE privilege on a user-defined function, DB2 marks the plan or package inoperative, instead of invalid; you will need to manually rebind the inoperative plan or package.

If a privilege that the plan or package depends on is revoked, and if you want to invalidate the plan or package or make it inoperative, you must use the SQL GRANT statement to grant the revoked privilege and then use the SQL REVOKE statement to revoke it. 

Dropping views

When a privilege that is required to create a view is revoked, the view is dropped. Similar to the revocation of plan privileges, such an event is not communicated to DB2 by the authorization checking routine. If you want DB2 to drop the view when a privilege is revoked, you must use the SQL statements GRANT and REVOKE.

Caching of EXECUTE on plans, packages, and routines

The results of authorization checks on the EXECUTE privilege for plans are not cached when those checks are performed by the exit routine. The results of authorization checks on the EXECUTE privilege for packages and routines are cached if the package and routine authorization caching is enabled on your system.

 If this privilege is revoked in the exit routine, the cached information is not updated to reflect the revoke. You must use the GRANT statement and the REVOKE statement to update the cached information. 

Caching of dynamic SQL statements

Dynamic statements can be cached when they have passed the authorization checks if the dynamic statement caching is enabled on your system.

 If the privileges that this statement requires are revoked from the authorization ID that is cached with the statement, this cached statement must be invalidated. If the privilege is revoked in the exit routine this does not happen, and you must use the SQL statements GRANT and REVOKE to refresh the cache.



Resolution of user-defined functions

The create timestamp for the user-defined function must be older than the bind timestamp for the package or plan in which the user-defined function is invoked. If DB2 authorization checking is in effect, and DB2 performs an automatic rebind on a plan or package that invokes a user-defined function, any user-defined functions that were created after the original BIND or REBIND of the invoking plan or package are not candidates for execution.

 If you use an access control authorization exit routine, some user-defined functions that were not candidates for execution before the original BIND or REBIND of the invoking plan or package might become candidates for execution during the automatic rebind of the invoking plan or package. If a user-defined function is invoked during an automatic rebind, and that user-defined function is invoked from a trigger body and receives a transition table, the form of the invoked function that DB2 uses for function selection includes only the columns of the transition table that existed at the time of the original BIND or REBIND of the package or plan for the invoking program. 

Creating materialized query tables

When a materialized query table is created, a CRTVUAUTT authorization check is performed. The CRTVUAUTT check is used to determine whether the creator of a materialized query table can provide the required SELECT privileges on base tables to the owner of the materialized query table.

PSPI If the owner of the materialized query table has the required privileges, the CRTVUAUTT authorization check proves redundant. However, the check is performed before the owner of the materialized query table's privileges are determined. Therefore, if the materialized query table owner holds the necessary privileges and the creator of the materialized query table does not, the CRTVUAUTT check can produce unwanted error messages.

For an ACA exit routine to suppress unwanted error messages during the creation of materialized query tables, XAPLFSUP is turned on. **PSPI**

Parameter list for the access control authorization routine

PSPI The following diagram shows how the parameter list points to other information.

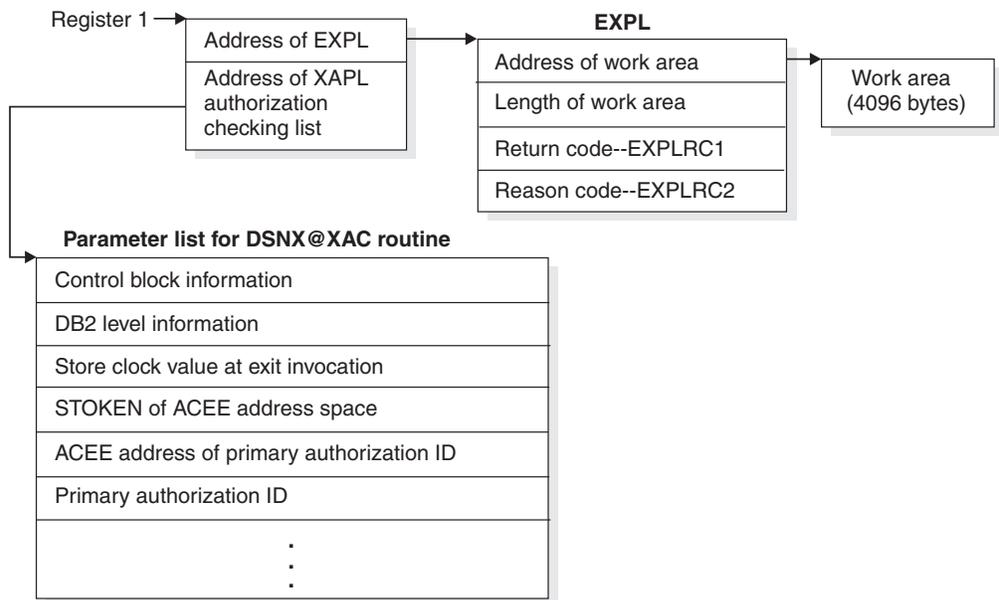


Figure 80. How an authorization routine's parameter list points to other information

The work area (4096 bytes) is obtained once during the startup of DB2 and only released when DB2 is shut down. The work area is shared by all invocations to the exit routine.

At invocation, registers are set as described in "Registers at invocation for exit routines" on page 811, and the authorization checking routine uses the standard exit parameter list (EXPL) described there. Table 127 on page 771 shows the exit-specific parameter list, described by macro DSNDXAPL. Field names indicated by an asterisk (*) apply to initialization, termination, and authorization checking. Other fields apply to authorization checking only.

Table 127. Parameter list for the access control authorization routine

Name	Hex offset	Data type	Input or output	Description
XAPLCBID*	0	Character, 2-bytes	Input	Control block identifier; value X'216A'.
XAPLLEN*	2	Signed, 2-byte integer	Input	Length of XAPL; value X'100' (decimal 256).
XAPLEYE*	4	Character, 4 bytes	Input	Control block eye catcher; value "XAPL".
XAPLLVL*	8	Character, 8 bytes	Input	DB2 version and level; for example, "VxRxMx ".
XAPLSTCK*	10	Character, 8 bytes	Input	The store clock value when the exit is invoked. Use this to correlate information to this specific invocation.
XAPLSTKN*	18	Character, 8 bytes	Input	STOKEN of the address space in which XAPLACEE resides. Binary zeroes indicate that XAPLACEE is in the home address space.
XAPLACEE*	20	Address, 4 bytes	Input	ACEE address: <ul style="list-style-type: none"> • Of the DB2 address space (<i>ssnmDBM1</i>) when XAPLFUNC is 1 or 3. • Of the primary authorization ID associated with this agent when XAPLFUNC is 2. There may be cases were an ACEE address is not available for an agent. In such cases this field contains binary zeroes.
XAPLUPRM*	24	Character, 8 bytes	Input	One of the following IDs: <ul style="list-style-type: none"> • When XAPLFUNC is 1 or 3, it contains the User ID of the DB2 address space (<i>ssnmDBM1</i>) • When XAPLFUNC is 2, it contains the primary authorization ID associated with the agent
XAPLFUNC*	2C	Signed, 2-byte integer	Input	Function to be performed by exit routine <ul style="list-style-type: none"> 1 Initialization 2 Authorization Check 3 Termination
XAPLGPAT*	2E	Character, 4 bytes	Input	DB2 group attachment name for data sharing. The DB2 subsystem name if not data sharing.
XAPLUCKT	32	Character, 1 byte	Input	Type of the authorization ID on which DB2 performs the check. <ul style="list-style-type: none"> '' An authorization ID L A role
XAPLONRT	33	Character, 1 byte	Input	Type of the authorization ID that owns the object in XAPLOWNR. <ul style="list-style-type: none"> '' An authorization ID L A role
XAPLRSV1	34	Character, 4 bytes		Reserved
XAPLPRIV	38	Signed, 2-byte integer	Input	DB2 privilege being checked.

Table 127. Parameter list for the access control authorization routine (continued)

Name	Hex offset	Data type	Input or output	Description
XAPLTYPE	3A	Character, 1	Input	DB2 object type: B Buffer pool C Collection D Database E Distinct typeDistinct type F User-defined functionUser-defined function J JAR K Package L Role M Schema N Trusted context O Stored procedure P Application plan Q Sequence R Table space S Storage group T Table U System privilege V View
XAPLFLG1	3B	Character, 1	Input	The highest-order bit, bit 8, (XAPLCHKS) is on if the secondary IDs associated with this authorization ID (XAPLUCHK) are included in DB2's authorization check. If it is off, only this authorization ID is checked. Bit 7 (XAPLUTB) is on if this is a table or view privilege (SELECT, INSERT, and so on) and if SYSCTRL is not sufficient authority to perform the specified operation on a table or view. SYSCTRL does not have the privilege of accessing user data unless the privilege is specifically granted to it. Bit 6 (XAPLAUTO) is on if this is an AUTOBIND. Bit 5 (XAPLCRVW) is on if the installation parameter DBADM CREATE AUTH is set to YES. Bit 4 (XAPLRDWR) is on if the privilege is a write privilege. If the privilege is a read-only privilege, bit 4 is off. Bit 3 (XAPLFSUP) is on to suppress error messages from the CRTVUAUTT authorization check during the creation of a materialized query table. These error messages are caused by intermediate checks that do not affect the final result. Bit 2 (XAPLRAOO) is on if this operation is in a trusted context that is defined with the ROLE AS OBJECT OWNER clause. Bit 1 (XAPLIMPD) is on if authorization checking involves an implicitly created database.

Table 127. Parameter list for the access control authorization routine (continued)

Name	Hex offset	Data type	Input or output	Description
XAPLUCHK	3C	Address, 4 bytes	Input	Address to the authorization ID on which DB2 performs the check. It could be the primary, secondary, or some other ID. This is a VARCHAR(128) field.
XAPLOBJN	40	Address, 4 bytes	Input	<p>Address to the unqualified name of the object with which the privilege is associated. This is a VARCHAR(128) field. It is one of the following names:</p> <p>Name Length</p> <p>Application plan 8</p> <p>Buffer pool 8</p> <p>Collection VARCHAR(128)</p> <p>Database 8</p> <p>Distinct type VARCHAR(128)</p> <p>JAR VARCHAR(128)</p> <p>Package VARCHAR(128)</p> <p>Role VARCHAR(128)</p> <p>Schema VARCHAR(128)</p> <p>Sequence VARCHAR(128)</p> <p>Storage group VARCHAR(128)</p> <p>Table VARCHAR(128)</p> <p>Table space 8</p> <p>Trusted context VARCHAR(128)</p> <p>User-defined function VARCHAR(128)</p> <p>View VARCHAR(128)</p> <p>For special system privileges (SYSADM, SYSCTRL, and so on) this field might contain binary zeroes.</p>
XAPLOWNQ	44	Address, 4 bytes	Input	<p>Address of the object owner (creator) or object qualifier. The contents of this parameter depends on either the privilege being checked or the object. This is a VARCHAR(128) field.</p> <p>If this field is not applicable, it contains binary zeros.</p>

Table 127. Parameter list for the access control authorization routine (continued)

Name	Hex offset	Data type	Input or output	Description
XAPLREL1	48	Address, 4 bytes	Input	Address of other related information 1. The contents of this parameter depend on either the privilege being checked or the object. This is a VARCHAR(128) field. If this field is not applicable, it contains binary zeros.
XAPLREL2	4C	Address, 4 bytes	Input	Address of other related information 2. The contents of this parameter depends on the privilege being checked. This is a VARCHAR(128) field. If this field is not applicable, it contains binary zeros.
XAPLDBSP	50	Address, 4 bytes	Input	Address of database information. This information is passed for CREATE VIEW and CREATE ALIAS. If this field is not applicable, it contains binary zeros.
XAPLOWNR	54	Address, 4 bytes	Input	Address of the object owner. This is a VARCHAR(128) field. If this field is not applicable, it contains binary zeros.
XAPLROLE	58	Address, 4 bytes	Input	Address of the user's role when operating in a trusted context. If this field is not applicable, it contains binary zeros.
XAPLOONM	5C	Character, 4 byte	Input	Address of other object name
XAPLOOON	60	Character, 4 byte	Input	Address of other object owner
XAPLRV2	64	Character, 61 bytes		Reserved.
XAPLOOTP	A1	Character, 1 byte	Input	Address of other object type
XAPLOOOT	A2	Character, 1 byte	Input	Address of other object owner type
XAPLFROM	A3	Character, 1 byte	Input	Source of the request: S Remote request that uses DB2 private protocol. ' ' Not a remote request that uses DB2 private protocol. DB2 authorization restricts remote requests that use DB2 private protocol to the SELECT, UPDATE, INSERT and DELETE privileges.
XAPLXBTS	A4	Timestamp, 10 bytes	Input	The function resolution timestamp. Authorizations received prior to this timestamp are valid. Applicable to functions and procedures.
XAPLONWT	AE	Character, 1 byte	Output	Information required by DB2 from the exit routine for the UPDATE and REFERENCES table privileges: Value Explanation ' ' Requester has privilege on the entire table * Requester has privilege on just this column
XAPLRV3	AF	Character, 1 byte		Reserved.
XAPLDIAG	B0	Character, 80 bytes	Output	Information returned by the exit routine to help diagnose problems.

Table 128 has database information for determining authorization for creating a view. The address to this parameter list is in XAPLREL2.

Table 128. Parameter list for the access control authorization routine—database information

Name	Hex offset	Data type	Input or output	Description
XAPLDBNP	0	Address	Input	Address of information for the next database. X'00000000' indicates no next database exists.
XAPLDBNM	4	Character, 8 bytes	Input	Database name.
XAPLDBDA	C	Character, 1 byte	Output	<p>Required by DB2 from the exit routine for CREATE VIEW.</p> <p>A value of Y and EXPLRC1=0 indicate that the user ID in field XAPLUCHK has database administrator authority on the database in field XAPLDBNM.</p> <p>When the exit checks if XAPLUCHK can create a view for another authorization ID, it first checks for SYSADM or SYSCTRL authority. If the check is successful, no more checking is necessary because SYSCTRL authority (for non-user tables) or SYSADM authority satisfies the requirement that the view owner has the SELECT privilege for all tables and views that the view might be based on. This is indicated by a blank value and EXPLRC1=0.</p> <p>If the authorization ID does not have SYSADM or SYSCTRL authority, the exit checks if the view creator has DBADM on each database of the tables that the view is based on because the DBADM authority on the database of the base table satisfies the requirement that the view owner has the SELECT privilege for all base tables in that database.</p>
XAPLRSV5	D	Character, 3 bytes	none	Reserved.

XAPLOWNQ, XAPLREL1 and XAPLREL2 might further qualify the object or may provide additional information that can be used in determining authorization for certain privileges. These privileges and the contents of XAPLOWNQ, XAPLREL1 and XAPLREL2 are shown in Table 129.

Table 129. Related information for certain privileges

Privilege	Object type (XAPLTYPE)	XAPLOWNQ	XAPLREL1	XAPLREL2	XAPLOWNR
0263 (USAGE)	E	Address of schema name	Address of distinct type owner	Contains binary zeroes	Address of distinct type owner

Table 129. Related information for certain privileges (continued)

Privilege	Object type (XAPLTYPE)	XAPLOWNQ	XAPLREL1	XAPLREL2	XAPLOWNR
0064 (EXECUTE) 0265 (START) 0266 (STOP) 0267 (DISPLAY)	F	Address of schema name	Address of user-defined function owner	Contains binary zeroes	Address of user-defined function owner
0263 (USAGE)	J	Address of schema name	Address of JAR owner	Contains binary zeroes	Address of JAR owner
0064 (EXECUTE)	K	Address of collection ID	Contains binary zeroes	Contains binary zeroes	Contains binary zeroes
0065 (BIND)	K	Address of collection ID	Address of package owner	Contains binary zeroes	Address of package owner
0073 (DROP)	K	Address of collection ID	Contains binary zeroes	Address of version ID	Contains binary zeroes
0097 (COMMENT)	K	Address of collection ID	Address of package owner	Contains binary zeroes	Address of package owner
0225 (COPY ON PKG)	K	Address of collection ID	Address of package owner	Contains binary zeroes	Address of package owner
0228 (ALLPKAUT)	K	Address of collection ID	Contains binary zeroes	Contains binary zeroes	Contains binary zeroes
0229 (SUBPKAUT)	K	Address of collection ID	Contains binary zeroes	Contains binary zeroes	Contains binary zeroes
0252 (ALTERIN) 0097 (COMMENT) 0252 (DROPIN)	M	Address of schema name	Address of object owner	Contains binary zeroes	Address of object owner
0064 (EXECUTE) 0265 (START) 0266 (STOP) 0267 (DISPLAY)	O	Address of schema name	Address of procedure owner	Contains binary zeroes	Address of procedure owner
0065 (BIND)	P	Address of plan owner	Contains binary zeroes	Contains binary zeroes	Address of plan owner
0097 (COMMENT)	P	Address of plan owner	Contains binary zeroes	Contains binary zeroes	Address of plan owner
0061 (ALTER) 0263 (USAGE)	Q	Address of schema name	Address of sequence name	Contains binary zeroes	Contains binary zeroes
0061 (ALTER)	R	Address of database name	Contains binary zeroes	Contains binary zeroes	Contains binary zeroes
0073 (DROP)	R	Address of database name	Contains binary zeroes	Contains binary zeroes	Contains binary zeroes
0087 (USE)	R	Address of database name	Contains binary zeroes	Contains binary zeroes	Contains binary zeroes
0053 (UPDATE) 0054 (REFERENCES)	T	Address of table owner qualifier	Address of column name, if applicable	Address of database name	Address of table owner

Table 129. Related information for certain privileges (continued)

Privilege	Object type (XAPLTYPE)	XAPLOWNQ	XAPLREL1	XAPLREL2	XAPLOWNR
0022 (CATMAINT CONVERT)	T	Address of table owner qualifier	Contains binary zeroes	Address of database name	Address of table owner
0050 (SELECT)					
0051 (INSERT)					
0052 (DELETE)					
0055 (TRIGGER)					
0056 (CREATE INDEX)					
0061 (ALTER)					
0073 (DROP)					
0075 (LOAD)					
0076 (CHANGE NAME QUALIFIER)					
0097 (COMMENT)					
0098 (LOCK)					
0233 (ANY TABLE PRIVILEGE)					
0251 (RENAME)					
0275 (REFRESH)					
0020 (DROP ALIAS)	T	Address of table owner qualifier	Contains binary zeroes	Contains binary zeroes	Contains binary zeroes
0104 (DROP SYNONYM)					
0103 (ALTER INDEX)	T	Address of table owner qualifier	Contains binary zeroes	Address of database name	Address of index owner
0105 (DROP INDEX)					
0274 (COMMENT ON INDEX)					
0283 (RENAME INDEX)					
0227 (BIND AGENT)	U	Address of package owner	Contains binary zeroes	Contains binary zeroes	Address of package owner
0015 (CREATE ALIAS)	U	Contains binary zeroes	Contains binary zeroes	Address of database name, if the alias is on a table	Contains binary zeroes
0053 (UPDATE)	V	Address of view owner qualifier	Address of column name, if applicable	Contains binary zeroes	Address of view owner
0050 (SELECT)	V	Address of view owner qualifier	Contains binary zeroes	Contains binary zeroes	Address of view owner
0051 (INSERT)					
0052 (DELETE)					
0073 (DROP)					
0097 (COMMENT)					
0233 (ANY TABLE PRIVILEGE)					
0055 (TRIGGER)	V	Address of view owner qualifier	Contains binary zeroes	Contains binary zeroes	Address of view owner
0061 (ALTER)	V	Address of view owner qualifier	Contains binary zeroes	Contains binary zeroes	Address of view owner

The data types and field lengths of the information shown in Table 129 on page 775 is shown in Table 130 on page 778.

Table 130. Data types and field lengths

Resource name or other	Type	Length
Database name	Character	8
Table name qualifier	Character	VARCHAR(128)
Object name qualifier	Character	VARCHAR(128)
Column name	Character	VARCHAR(128)
Collection ID	Character	VARCHAR(128)
Plan owner	Character	VARCHAR(128)
Package owner	Character	VARCHAR(128)
Package version ID	Character	VARCHAR(64)
Schema name	Character	VARCHAR(128)
Distinct typeowner	Character	VARCHAR(128)
JAR owner	Character	VARCHAR(128)
User-defined function owner	Character	VARCHAR(128)
Procedure owner	Character	VARCHAR(128)
View name qualifier	Character	VARCHAR(128)
Sequence owner	Character	VARCHAR(128)
Sequence name	Character	VARCHAR(128)



Expected output for the access control authorization routine

Your authorization exit routine is expected to return certain fields when it is called. If an unexpected value is returned in any of these fields an abend occurs.



The output fields are indicated in Table 127 on page 771. Register 3 points to the field in error, and abend code 00E70009 is issued.

Table 131. Output fields for the access control authorization routine

Field	Required or optional
EXPLRC1	Required
EXPLRC2	Optional
XAPLONWT	Required only for UPDATE and REFERENCES table privileges
XAPLDIAG	Optional



Handling return codes

Place return codes from the exit routine in the EXPL field named EXPLRC1.



EXPLRC1 must have one of the values that are shown in Table 132 on page 779 during initialization.

Table 132. Required values in EXPLRC1 during initialization

Value	Meaning
0	Initialization successful.
12	Unable to service request; don't call exit again.

DB2 does not check EXPLRC1 on return from the exit routine.

Make sure that EXPLRC1 has one of the values that are shown in Table 133 during the authorization check.

Table 133. Required values in EXPLRC1 during authorization check

Value	Meaning
0	Access permitted.
4	Unable to determine; perform DB2 authorization checking.
8	Access denied.
12	Unable to service request; don't call exit routine again.

See "Exception processing" for an explanation of how the EXPLRC1 value affects DB2 processing. On authorization failures, the return code is included in the IFCID 0140 trace record.



Handling reason codes

The reason code (EXPLRC2) that the exit routine returns after initialization determines how DB2 processes the return code (EXPLRC1) that the exit returns during initialization and authorization checking.



The reason codes are shown in Table 134.

Table 134. Reason codes during initialization

Value	Meaning
-1	Identifies the default exit routine shipped with DB2. If you replace or modify the default exit, you should not use this value.
16	Indicates to DB2 that it should terminate if the exit routine returns EXPLRC1=12, an invalid EXPLRC1 or abnormally terminates during initialization or authorization checking. When the exit routine sets the reason code to 16, DB2 does an immediate shutdown, without waiting for tasks to end. For long-running tasks, an immediate shutdown can mean that recovery times are long.
Other	Ignored by DB2.

Field EXPLRC2 lets you put in any code that would be of use in determining why the authorization check in the exit routine failed. On authorization failures, the reason code is included in the IFCID 0140 trace record. 

Exception processing

During initialization or authorization checking, DB2 issues diagnostic message DSNX210I to the operator's console, if one of the following conditions occur:

PSPI

- The authorization exit returns a return code of 12 or an invalid return code.
- The authorization exit abnormally terminates.

Additional actions that DB2 performs depend on the reason code that the exit returns during initialization. Table 135 summarizes these actions.

Table 135. How an error condition affects DB2 actions during initialization and authorization checking

Exit result	Reason code of 16 returned by exit routine during initialization	Reason code other than 16 or -1 returned by exit routine during initialization ¹
Return code 12	<ul style="list-style-type: none"> • The task² abnormally terminates with reason code 00E70015 • DB2 terminates 	<ul style="list-style-type: none"> • The task² abnormally terminates with reason code 00E70009 • DB2 switches to DB2 authorization checking
Invalid return code	<ul style="list-style-type: none"> • The task² abnormally terminates with reason code 00E70015 • DB2 terminates 	<ul style="list-style-type: none"> • The task² abnormally terminates with reason code 00E70009 • DB2 switches to DB2 authorization checking
Abnormal termination during initialization	DB2 terminates	DB2 switches to DB2 authorization checking
Abnormal termination during authorization checking	<p>You can use the subsystem parameter AEXITLIM³ to control how DB2 and the exit behave.</p> <p>Example: If you set AEXITLIM to 10, the exit routine continues to run after the first 10 abnormal terminations. On the eleventh abnormal termination, the exit stops and DB2 terminates.</p>	<p>You can use the subsystem parameter AEXITLIM to control how DB2 and the exit behave.</p> <p>Example: If you set AEXITLIM to 10, the exit routine continues to run after the first 10 abnormal terminations. On the eleventh abnormal termination, the exit routine stops and DB2 switches to DB2 authorization checking.</p>

Note:

1. During initialization, DB2 sets a value of -1 to identify the default exit. The user exit routine should not set the reason code to -1.
2. During initialization, the task is DB2 startup. During authorization checking, the task is the application.
3. AEXITLI (authorization exit limit) can be updated online.

PSPI

Debugging the access control authorization routine

You can use IFCID 0314 to provide a trace record of the parameter list on return from the exit routine. You can activate this trace by turning on performance trace class 22.

Determining whether the access control authorization routine is active

To determine whether the exit routine or DB2 is performing authorization checks:

PSPI

1. Start audit trace class 1.
2. Choose a DB2 table on which to issue a SELECT statement and an authorization ID to perform the SELECT. The authorization ID must not have the DB2 SELECT privilege or the external security system SELECT privilege on the table.
3. Use the authorization ID to issue a SELECT statement on the table. The SELECT statement should fail.
4. Format the trace data and examine the return code (QW0140RC) in the IFCID 0140 trace record.
 - QW0140RC = -1 indicates that DB2 performed the authorization check and denied access.
 - QW0140RC = 8 indicates that the external security system performed the authorization check and denied access.

PSPI

Edit routines

Edit routines are assigned to a table by the EDITPROC clause of CREATE TABLE. An edit routine receives the entire row of the base table in internal DB2 format; it can transform that row when it is stored by an INSERT or UPDATE SQL statement, or by the LOAD utility.

PSPI

An edit routine also receives the transformed row during retrieval operations and must change it back to its original form. Typical uses are to compress the storage representation of rows to save space on DASD and to encrypt the data.

You cannot use an edit routine on a table that contains a LOB or a ROWID column. Also, you cannot use an EDITPROC if the table has an XML column or a security label column or if the table contains a column name that is longer than 18 EBCDIC bytes.

The transformation your edit routine performs on a row (possibly encryption or compression) is called *edit-encoding*. The same routine is used to undo the transformation when rows are retrieved; that operation is called *edit-decoding*.

Important: The edit-decoding function must be the exact inverse of the edit-encoding function. For example, if a routine encodes 'ALABAMA' to '01', it must decode '01' to 'ALABAMA'. A violation of this rule can lead to an abend of the DB2 connecting thread, or other undesirable effects.

Your edit routine can encode the entire row of the table, including any index keys. However, index keys are extracted from the row before the encoding is done, therefore, index keys are stored in the index in *edit-decoded* form. Hence, for a table with an edit routine, index keys in the table are edit-coded; index keys in the index are **not** edit-coded.

The sample application contains a sample edit routine, DSN8EAE1. To print it, use ISPF facilities, IEBTPCH, or a program of your own. Or, assemble it and use the assembly listing.

There is also a sample routine that does Huffman data compression, DSN8HUFF in library *prefix*.SDSNSAMP. That routine not only exemplifies the use of the exit parameters, it also has potentially some use for data compression. If you intend to use the routine in any production application, please pay particular attention to the warnings and restrictions given as comments in the code. You might prefer to let DB2 compress your data.

“General guidelines for writing exit routines” on page 810 applies to edit routines.

PSPI

Specifying the edit routine

To name an edit routine for a table, use the EDITPROC clause of the CREATE TABLE statement, followed by the name of the routine.

PSPI

If you plan to use an edit routine, specify it when you create the table. In operation, the routine is loaded on demand.

You cannot add an edit routine to a table that already exists: you must drop the table and re-create it. Also, you cannot alter a table with an edit routine to add a column. Again, you must drop the table and re-create it, and presumably also alter the edit routine in some way to account for the new column. PSPI

When the edit routine is taken

An edit routine is invoked to edit-encode a row whenever DB2 inserts or updates one, including inserts made by the LOAD utility.

PSPI

An edit routine is invoked *after* any date routine, time routine, or field procedure. If there is also a validation routine, the edit routine is invoked **after** the validation routine. Any changes made to the row by the edit routine do not change entries made in an index.

The same edit routine is invoked to edit-decode a row whenever DB2 retrieves one. On retrieval, it is invoked *before* any date routine, time routine, or field procedure. If retrieved rows are sorted, the edit routine is invoked *before* the sort. An edit routine is not invoked for a DELETE operation without a WHERE clause that deletes an entire table in a segmented table space. PSPI

Parameter lists for the edit routine

PSPI

At invocation, registers are set as described in “Registers at invocation for exit routines” on page 811, and the edit routine uses the standard exit parameter list (EXPL) described there. Table 136 on page 783 shows the exit-specific parameter list, described by macro DSNDEDIT.

Table 136. Parameter list for an edit routine

Name	Hex offset	Data type	Description
EDITCODE	0	Signed 4-byte integer	Edit code telling the type of function to be performed, as follows:
			0 Edit-encode row for insert or update
EDITROW	4	Address	Address of a row description. Its format is shown in Table 179 on page 817.
	8	Signed 4-byte integer	Reserved
EDITILTH	C	Signed 4-byte integer	Length of the input row
EDITPTR	10	Address	Address of the input row
EDITOLTH	14	Signed 4-byte integer	Length of output row. On entry, this is the size of the area in which to place the output row. The exit must not modify storage beyond this length.
EDITOPTR	18	Address	Address of the output row



Processing requirements for edit routines

Your routine must be based on the DB2 data formats. See “Row formats for edit and validation routines” on page 813 for more information.

Incomplete rows and edit routines

If DB2 passes, to an edit routine, an input row that has fewer fields than the number of columns in the table, the routine must stop processing the row after the last input field.



Columns for which no input field is provided and that are not in reordered row format are always at the end of the row and are never defined as NOT NULL. In this case, the columns allow nulls, they are defined as NOT NULL WITH DEFAULT, or the columns are ROWID or DOCID columns.

Use macro DSNDEDIT to get the starting address and row length for edit exits. Add the row length to the starting address to get the first invalid address beyond the end of the input buffer; your routine must *not* process any address as large as that.

Figure 81 on page 784 shows how the parameter list points to other row information. The address of the *n*th column description is given by: $RFMTAFLD + (n-1) * (FFMTE - FFMTE)$.

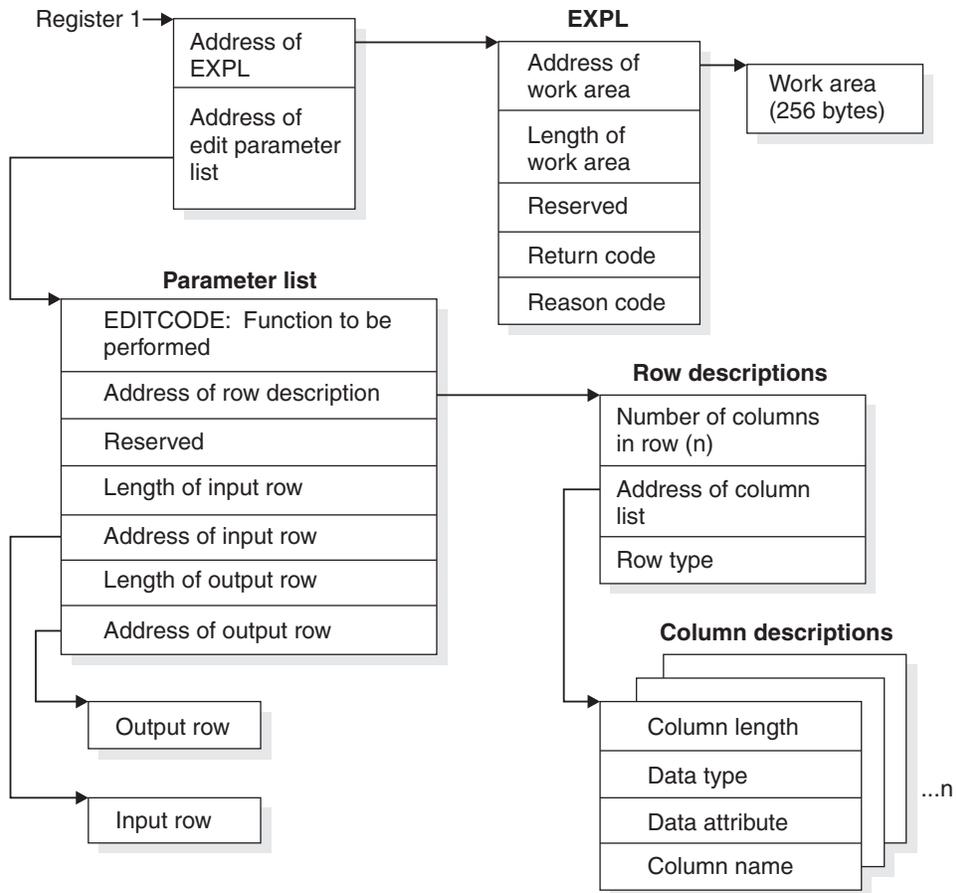


Figure 81. How the edit exit parameter list points to row information



Expected output for edit routines

If **EDITCODE** contains 0, the input row is in decoded form. Your routine must encode it.

In that case, the maximum length of the output area, in **EDITOLTH**, is 10 bytes more than the maximum length of the record. In counting the maximum length for a row in basic row format, "record" includes fields for the lengths of varying-length columns and for null indicators. In counting the maximum length for a row in reordered row format, "record" includes fields for the offsets to the varying length columns and for null indicators. The maximum length of the record does not include the 6-byte record header.

If **EDITCODE** contains 4, the input row is in coded form. Your routine must decode it.

In that case, **EDITOLTH** contains the maximum length of the record. In counting the maximum length for a row in basic row format, "record" includes fields for the lengths of varying length columns and for null indicators. In counting the maximum length for a row in reordered row format, "record" includes fields for the offsets to the varying-length columns and for null indicators. The maximum length of the record does not include the 6-byte record header.

In either case, put the result in the output area, pointed to by EDITOPTR, and put the length of your result in EDITOLTH. The length of your result must not be greater than the length of the output area, as given in EDITOLTH on invocation, and your routine must not modify storage beyond the end of the output area.

Required return code: Your routine must also leave a return code in EXPLRC1, with the meanings that are listed in Table 137:

Table 137. Required return code in EXPLRC1

Value	Meaning
0	Function performed successfully.
Nonzero	Function failed.

If the function fails, the routine might also leave a reason code in EXPLRC2. DB2 returns SQLCODE -652 (SQLSTATE '23506') to the application program and puts the reason code in field SQLERRD(6) of the SQL communication area (SQLCA).

Validation routines

Validation routines are assigned to a table by the VALIDPROC clause of CREATE TABLE and ALTER TABLE. A validation routine receives an entire row of a base table as input, and can return an indication of whether or not to allow a following INSERT, UPDATE, DELETE, FETCH, or SELECT operation.

PSPI Typically, a validation routine is used to impose limits on the information that can be entered in a table; for example, allowable salary ranges, perhaps dependent on job category, for the employee sample table.

Although VALIDPROCs can be specified for a table that contains a LOB or XML column, the LOB or XML values are not passed to the validation routine. The LOB indicator column takes the place of the LOB column, and the XML indicator column takes the place of the XML column. You cannot use VALIDPROC on a table if the table contains a column name that is longer than 18 EBCDIC bytes.

The return code from a validation routine is checked for a 0 value before any insert, update, or delete is allowed.

“General guidelines for writing exit routines” on page 810 applies to validation routines. **PSPI**

Specifying the validation routine

To name a validation routine for a table, use the VALIDPROC clause of the CREATE TABLE or ALTER TABLE statement, followed by the name of the routine. In operation, the routine is loaded on demand.

PSPI You can add a validation routine to a table that is already in existence, but it is not invoked to validate data already in the table. You can also cancel any validation routine for a table, by using VALIDPROC NULL in an ALTER TABLE statement. **PSPI**

When the validation routine is taken

A validation routine for a table is invoked when DB2 inserts or updates a row, including inserts made by the LOAD utility.

PSPI The routine is invoked for most delete operations, including a mass delete of all the rows of a table. If there are other exit routines, the validation routine is invoked *before* any edit routine, and *after* any date routine, time routine, or field procedure. **PSPI**

Parameter lists for the validation routine

At invocation, registers are set, and the validation routine uses the standard exit parameter list (EXPL).

PSPI

Table 138 shows the exit-specific parameter list, described by macro DSNDRVAL.

Table 138. Parameter list for a validation routine

Name	Hex offset	Data type	Description
	0	Signed 4-byte integer	Reserved
RVALROW	4	Address	Address of a row description.
	8	Signed 4-byte integer	Reserved
RVALROWL	C	Signed 4-byte integer	Length of the input row to be validated
RVALROWP	10	Address	Address of the input row to be validated
	14	Signed 4-byte integer	Reserved
	18	Signed 4-byte integer	Reserved
RVALPLAN	1C	Character, 8 bytes	Name of the plan issuing the request
RVALOPER	24	Unsigned 1-byte integer	Code identifying the operation being performed, as follows: 1 Insert, update, or load 2 Delete
RVALFL1	25	Character, 1 byte	The high-order bit is on if the requester has installation SYSADM authority. The remaining 7 bits are reserved.
RVALCSTC	26	Character, 2 bytes	Connection system type code. Values are defined in macro DSNDCSTC.

PSPI

Processing requirements for validation routines

Your routine must be based on the DB2 data formats. See “Row formats for edit and validation routines” on page 813 for more information.

Incomplete rows and validation routines

If DB2 passes, to a validation routine, an input row that has fewer fields than the number of columns in the table, the routine must stop processing the row after the last input field.

PSPI Columns for which no input field is provided and that are not in reordered row format are always at the end of the row and are never defined as NOT NULL. In this case, the columns allow nulls, they are defined as NOT NULL WITH DEFAULT, or the columns are ROWID or DOCID columns.

Use macro DSNDRVAL to get the starting address and row length for validation exits. Add the row length to the starting address to get the first invalid address beyond the end of the input buffer; your routine must **not** process any address as large as that. **PSPI**

Expected output for validation routines

PSPI

Your routine must leave a return code in EXPLRC1, with the meanings that are listed in Table 139.

Table 139. Required return code in EXPLRC1

Value	Meaning
0	Allow insert, update, or delete
Nonzero	Do not allow insert, update, or delete

If the operation is not allowed, the routine might also leave a reason code in EXPLRC2. DB2 returns SQLCODE -652 (SQLSTATE '23506') to the application program and puts the reason code in field SQLERRD(6) of the SQL communication area (SQLCA).

Figure 82 on page 788 shows how the parameter list points to other information.

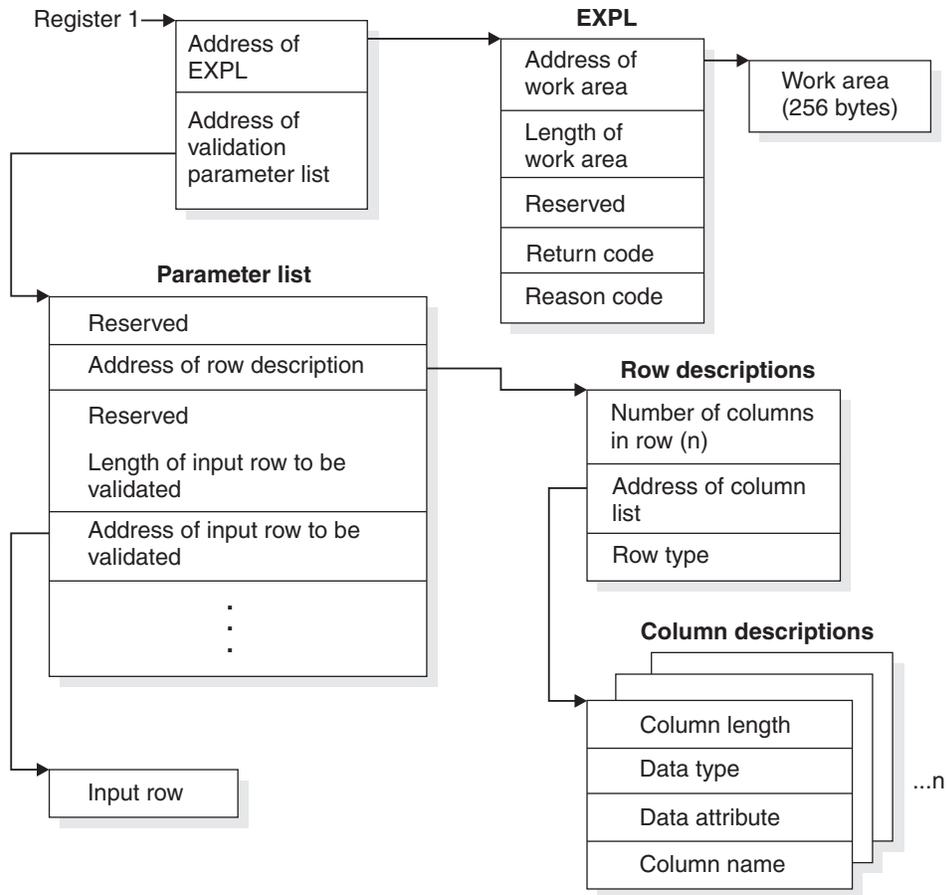


Figure 82. How a validation parameter list points to information. The address of the n th column description is given by: $RFMTAFLD + (n-1) * (FFMTE - FFMT)$.



Date and time routines

A date routine is a user-written exit routine to change date values from a locally defined format into a format recognized by DB2 and from the ISO format into the locally defined format. Similarly, a time routine changes time values from a locally defined format into one recognized by DB2, and from ISO into the locally-defined format.



Table 140 shows the formats recognized by DB2.

Table 140. Date and time formats

Format name	Abbreviation	Typical date	Typical time
IBM European standard	EUR	25.12.2004	13.30.05
International Standards Organization	ISO	2004-12-25	13.30.05
Japanese Industrial Standard Christian Era	JIS	2004-12-25	13:30:05
IBM USA standard	USA	12/25/2004	1:30 PM

Example: Suppose that you want to insert and retrieve dates in a format like “September 21, 2006”. You can use a date routine that transforms the date to a format that is recognized by DB2 on insertion, such as ISO: “2006-09-21”. On retrieval, the routine can transform “2006-09-21” to “September 21, 2006”.

You can have either a date routine, a time routine, or both. These routines do not apply to timestamps. Special rules apply if you execute queries at a remote DBMS, through the distributed data facility. 

Specifying the date and time routine

To establish a date or time routine, set LOCAL DATE LENGTH or LOCAL TIME LENGTH to the length of the longest field required to hold a date or time in your local format during DB2 installation.

 Allowable values range from 10 to 254. For example, if you intend to insert and retrieve dates in the form “September 21, 2006”, then you need an 18-byte field. Set LOCAL DATE LENGTH to 18.

Also, replace all of the IBM-supplied exit routines, using CSECTs DSNXVDTX, DSNXVDTA, and DSNXVDTU for a date routine, and DSNXVTMX, DSNXVTMA, and DSNXVTMU for a time routine. The routines are loaded when DB2 starts.

To make the local date or time format the default for retrieval, set DATE FORMAT or TIME FORMAT to LOCAL when installing DB2. That has the effect that DB2 *always* takes the exit routine when you retrieve from a DATE or TIME column. In our example, suppose that you want to retrieve dates in your local format only occasionally; most of the time you use the USA format. Set DATE FORMAT to USA.

The install parameters for LOCAL DATE LENGTH, LOCAL TIME LENGTH, DATE FORMAT, and TIME FORMAT can also be updated after DB2 is installed. If you change a length parameter, you might need to rebind the applications. 

When date and time routines are taken

On insertion, a date or time routine is invoked to change a value from the locally-defined format to a format recognized by DB2 in the following circumstances:

-  When a date or time value is entered by an INSERT or UPDATE statement, or by the LOAD utility
- When a constant or host variable is compared to a column with a data type of DATE, TIME, or TIMESTAMP
- When the DATE or TIME scalar function is used with a string representation of a date or time in LOCAL format
- When a date or time value is supplied for a limit of a partitioned index in a CREATE INDEX statement

The exit is taken before any edit or validation routine.

- If the default is LOCAL, DB2 takes the exit immediately. If the exit routine does not recognize the data (EXPLRC1=8), DB2 then tries to interpret it as a date or time in one of the recognized formats (EUR, ISO JIS, or USA). DB2 rejects the data only if that interpretation also fails.

- If the default is not LOCAL, DB2 first tries to interpret the data as a date or time in one of the recognized formats. If that interpretation fails, DB2 then takes the exit routine, if it exists.

DB2 checks that the value supplied by the exit routine represents a valid date or time in some recognized format, and then converts it into an internal format for storage or comparison. If the value is entered into a column that is a key column in an index, the index entry is also made in the internal format.

On retrieval, a date or time routine can be invoked to change a value from ISO to the locally-defined format when a date or time value is retrieved by a SELECT or FETCH statement. If LOCAL is the default, the routine is always invoked unless overridden by a precompiler option or by the CHAR function, as by specifying CHAR(HIREDATE, ISO); that specification always retrieves a date in ISO format. If LOCAL is not the default, the routine is invoked only when specifically called for by CHAR, as in CHAR(HIREDATE, LOCAL); that always retrieves a date in the format supplied by your date exit routine.

On retrieval, the exit is invoked after any edit routine or DB2 sort. A date or time routine is not invoked for a DELETE operation without a WHERE clause that deletes an entire table in a segmented table space. 

Parameter lists for date and time routines

At invocation, registers are set, and the date or time routine uses the standard exit parameter list (EXPL).



Table 141 shows its exit-specific parameter list, described by macro DSNDDTXP.

Table 141. Parameter list for a date or time routine

Name	Hex offset	Data type	Description
DTXPFN	0	Address	Address of a 2-byte integer containing a function code. The codes and their meanings are: 4 Convert from local format to ISO. 8 Convert from ISO to local format.
DTXPLN	4	Address	Address of a 2-byte integer containing the length in bytes of the local format. This is the length given as LOCAL DATE LENGTH or LOCAL TIME LENGTH when installing DB2.
DTXPLOC	8	Address	Address of the date or time value in local format
DTXPISO	C	Address	Address of the date or time value in ISO format (DTXPISO). The area pointed to is 10 bytes long for a date, 8 bytes for a time.



Expected output for date and time routines

PSPI

If the function code is 4, the input value is in local format, in the area pointed to by DTXPLOC. Your routine must change it to ISO, and put the result in the area pointed to by DTXPISO.

If the function code is 8, the input value is in ISO, in the area pointed to by DTXPISO. Your routine must change it to your local format, and put the result in the area pointed to by DTXPLOC.

Your routine must also leave a return code in EXPLRC1, a 4-byte integer and the third word of the EXPL area. The return code can have the meanings that are shown in Table 142.

Table 142. Required return code in EXPLRC1

Value	Meaning
0	No errors; conversion was completed.
4	Invalid date or time value.
8	Input value not in valid format; if the function is insertion, and LOCAL is the default, DB2 next tries to interpret the data as a date or time in one of the recognized formats (EUR, ISO, JIS, or USA).
12	Error in exit routine.

Figure 83 shows how the parameter list points to other information.

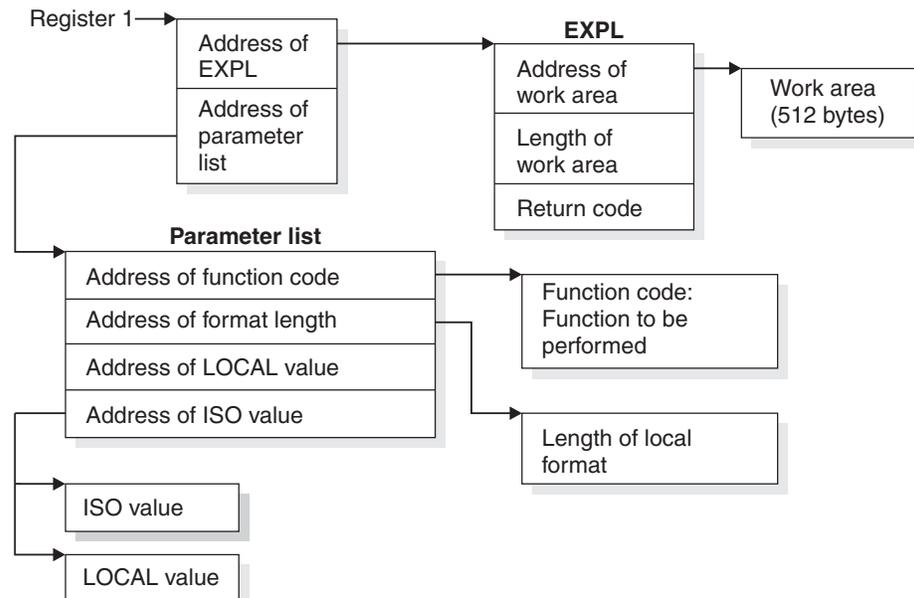


Figure 83. How a date or time parameter list points to other information

PSPI

Conversion procedures

A conversion procedure is a user-written exit routine that converts characters from one coded character set to another coded character set.

PSPI In most cases, any conversion that is needed can be done by routines provided by IBM. The exit for a user-written routine is available to handle exceptions. “General guidelines for writing exit routines” on page 810 applies to conversion routines. **PSPI**

Specifying a conversion procedure

To establish a conversion procedure, insert a row into the catalog table SYSIBM.SYSSTRINGS.

PSPI The row must contain values for the following columns:

INCCSID

The coded character set identifier (CCSID) of the source string.

OUTCCSID

The CCSID of the converted string.

TRANSTYPE

The nature of the conversion. Values can be:

GG	ASCII GRAPHIC to EBCDIC GRAPHIC
MM	EBCDIC MIXED to EBCDIC MIXED
MP	EBCDIC MIXED to ASCII MIXED
MS	EBCDIC MIXED to EBCDIC SBCS
PM	ASCII MIXED to EBCDIC MIXED
PP	ASCII MIXED to ASCII MIXED
PS	ASCII MIXED to EBCDIC SBCS
SM	EBCDIC SBCS to EBCDIC MIXED
SP	SBCS (ASCII or EBCDIC) to ASCII MIXED
SS	EBCDIC SBCS to EBCDIC SBCS

TRANSPROC

The name of your conversion procedure.

IBMREQD

Must be N.

DB2 does not use the following columns, but checks them for the allowable values listed. Values you insert can be used by your routine in any way. If you insert no value in one of these columns, DB2 inserts the default value listed.

ERRORBYTE

Any character, or null. The default is null.

SUBBYTE

Any character not equal to the value of ERRORBYTE, or null. The default is null.

TRANSTAB

Any character string of length 256 or the empty string. The default is an empty string. **PSPI**

When conversion procedures are taken

The exit is taken, and your procedure invoked, whenever a conversion is required from the coded character set identified by INCCSID to the coded character set identified by OUTCCSID.

Parameter lists for conversion procedures

At invocation, registers are set, and the conversion procedure uses the standard exit parameter list (EXPL).

PSPI A conversion procedure does *not* use an exit-specific parameter list, as described in “Parameter lists for exit routines” on page 812. Instead, the area pointed to by register 1 at invocation includes three words, which contain the addresses of the following items:

1. The EXPL parameter list
2. A string value descriptor that contains the character string to be converted
3. A copy of a row from SYSIBM.SYSSTRINGS that names the conversion procedure identified in TRANSPROC.

The length of the work area pointed to by the exit parameter list is generally 512 bytes. However, if the string to be converted is ASCII MIXED data (the value of TRANSTYPE in the row from SYSSTRINGS is PM or PS), then the length of the work area is 256 bytes, plus the length attribute of the string.

The string value descriptor: The descriptor has the format shown in Table 143.

Table 143. Format of string value descriptor for a conversion procedure

Name	Hex offset	Data type	Description
FPVDTYPE	0	Signed 2-byte integer	Data type of the value: Code Means 20 VARCHAR 28 VARGRAPHIC
FPVDVLEN	2	Signed 2-byte integer	The maximum length of the string
FPVDVALE	4	None	The string. The first halfword is the string's actual length in characters. If the string is ASCII MIXED data, it is padded out to the maximum length by undefined bytes.

The row from SYSSTRINGS: The row copied from the catalog table SYSIBM.SYSSTRINGS is in the standard DB2 row format. The fields ERRORBYTE and SUBBYTE each include a null indicator. The field TRANSTAB is of varying length and begins with a 2-byte length field. **PSPI**

Expected output for conversion procedures

Except in the case of certain errors, your conversion procedure should replace the string in FPVDVALE with the converted string.

PSPI

When converting MIXED data, your procedure must ensure that the result is well-formed. In any conversion, if you change the length of the string, you must set the length control field in FPVDVALE to the proper value. Over-writing storage beyond the maximum length of the FPVDVALE causes an abend.

Your procedure must also set a return code in field EXPLRC1 of the exit parameter list.

With the two codes that are shown in Table 144, provide the converted string in FPVDVALE.

Table 144. Codes for the converted string in FPVDVALE

Code	Meaning
0	Successful conversion
4	Conversion with substitution

For the remaining codes that are shown in Table 145, DB2 does not use the converted string.

Table 145. Remaining codes for the FPVDVALE

Code	Meaning
8	Length exception
12	Invalid code point
16	Form exception
20	Any other error
24	Invalid CCSID

Exception conditions: Return a length exception (code 8) when the converted string is longer than the maximum length allowed.

For an invalid code point (code 12), place the 1- or 2-byte code point in field EXPLRC2 of the exit parameter list.

Return a form exception (code 16) for EBCDIC MIXED data when the source string does not conform to the rules for MIXED data.

Any other uses of codes 8 and 16, or of EXPLRC2, are optional.

Error conditions: On return, DB2 considers any of the following conditions as a "conversion error":

- EXPLRC1 is greater than 16.
- EXPLRC1 is 8, 12, or 16 and the operation that required the conversion is *not* an assignment of a value to a host variable with an indicator variable.
- FPVDTYPE or FPVDVLEN has been changed.
- The length control field of FPVDVALE is greater than the original value of FPVDVLEN or is negative.

In the case of a conversion error, DB2 sets the SQLERRMC field of the SQLCA to HEX(EXPLRC1) CONCAT X'FF' CONCAT HEX(EXPLRC2).

Figure 84 on page 795 shows how the parameter list points to other information.

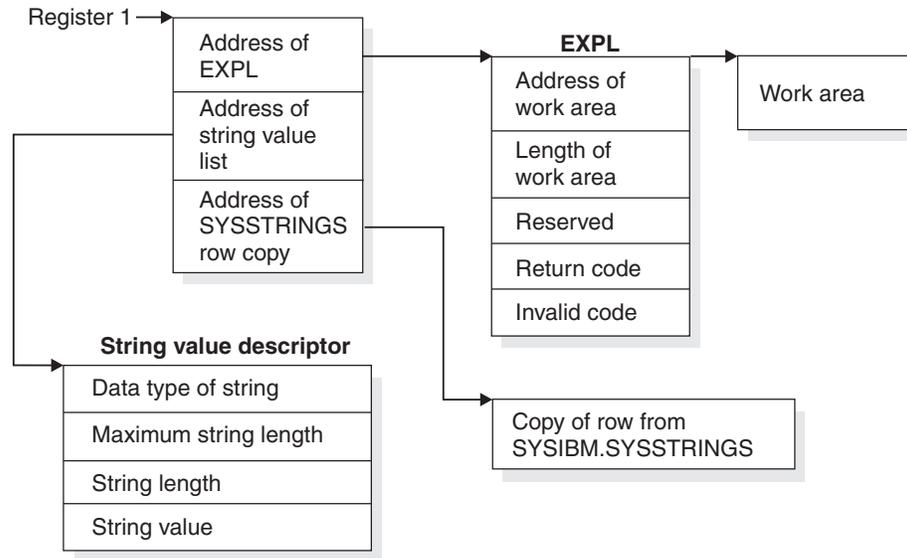


Figure 84. Pointers at entry to a conversion procedure

PSPI

Field procedures

Field procedures are assigned to a table by the `FIELDPROC` clause of `CREATE TABLE` and `ALTER TABLE`. A field procedure is a user-written exit routine to transform values in a single short-string column.

PSPI

When values in the column are changed, or new values inserted, the field procedure is invoked for each value, and can transform that value (encode it) in any way. The encoded value is then stored. When values are retrieved from the column, the field procedure is invoked for each value, which is encoded, and must decode it back to the original string value.

Any indexes, including partitioned indexes, defined on a column that uses a field procedure are built with encoded values. For a partitioned index, the encoded value of the limit key is put into the `LIMITKEY` column of the `SYSINDEXPART` table. Hence, a field procedure might be used to alter the sorting sequence of values entered in a column. For example, telephone directories sometimes require that names like “McCabe” and “MacCabe” appear next to each other, an effect that the standard EBCDIC sorting sequence does not provide. And languages that do not use the Roman alphabet have similar requirements. However, if a column is provided with a suitable field procedure, it can be correctly ordered by `ORDER BY`.

The transformation your field procedure performs on a value is called *field-encoding*. The same routine is used to undo the transformation when values are retrieved; that operation is called *field-decoding*. Values in columns with a field procedure are described to DB2 in two ways:

1. The description of the column as defined in `CREATE TABLE` or `ALTER TABLE` appears in the catalog table `SYSIBM.SYSCOLUMNS`. That is the description of the field-decoded value, and is called the *column description*.

2. The description of the encoded value, as it is stored in the data base, appears in the catalog table SYSIBM.SYSFIELDS. That is the description of the field-encoded value, and is called the *field description*.

Important: The field-decoding function must be the exact inverse of the field-encoding function. For example, if a routine encodes 'ALABAMA' to '01', it must decode '01' to 'ALABAMA'. A violation of this rule can lead to an abend of the DB2 connecting thread, or other undesirable effects.

“General guidelines for writing exit routines” on page 810 applies to field procedures. 

Field definition for field procedures

The field procedure is also invoked when the table is created or altered, to define the data type and attributes of an encoded value to DB2; that operation is called *field-definition*.

 The data type of the encoded value can be any valid SQL data type except DATE, TIME, TIMESTAMP, LONG VARCHAR, or LONG VARGRAPHIC; the allowable types are listed in the description of field FPVDTYPE in Table 148 on page 801. The length, precision, or scale of the encoded value must be compatible with its data type.

A user-defined data type can be a valid field if the source type of the data type is a short string column that has a null default value. DB2 casts the value of the column to the source type before it passes it to the field procedure. 

Specifying the field procedure

To name a field procedure for a column, use the FIELDPROC clause of the CREATE TABLE or ALTER TABLE statement, followed by the name of the procedure and, optionally, a list of parameters.

 You can use a field procedure only with a short string column. You cannot use a field procedure on a column defined using NOT NULL WITH DEFAULT.

If you plan to use a field procedure, specify it when you create the table. In operation, the procedure is loaded on demand. You cannot add a field procedure to an existing column of a table; you can, however, use ALTER TABLE to add to an existing table a new column that uses a field procedure.

You cannot use a field procedure on an LOB, ROWID, or ROW CHANGE TIMESTAMP column of a table. However, you can specify it for other columns in the same table. Also, you cannot use a field procedure on a column if the column name is longer than 18 EBCDIC bytes.

The optional parameter list that follows the procedure name is a list of constants, enclosed in parentheses, called the *literal list*. The literal list is converted by DB2 into a data structure called the *field procedure parameter value list* (FPPVL). That structure is passed to the field procedure during the field-definition operation. At that time, the procedure can modify it or return it unchanged. The output form of the FPPVL is called the *modified FPPVL*; it is stored in the DB2 catalog as part of

the field description. The modified FPPVL is passed again to the field procedure whenever that procedure is invoked for field-encoding or field-decoding. 

When field procedures are taken

A field procedure that is specified for a column is invoked in three general situations:

-  For field-definition, when the CREATE TABLE or ALTER TABLE statement that names the procedure is executed. During this invocation, the procedure is expected to:
 - Determine whether the data type and attributes of the column are valid.
 - Verify the literal list, and change it if wanted.
 - Provide the field description of the column.
 - Define the amount of working storage needed by the field-encoding and field-decoding processes.
- For field-encoding, when a column value is to be field-encoded. That occurs for any value that:
 - Is inserted in the column by an SQL INSERT statement, or loaded by the DB2 LOAD utility.
 - Is changed by an SQL UPDATE statement.
 - Is compared to a column with a field procedure, unless the comparison operator is LIKE. The value being encoded is a host variable or constant. (When the comparison operator is LIKE, the column value is decoded.)
 - Defines the limit of a partition of an index. The value being encoded follows ENDING AT in the PARTITION clause of CREATE INDEX.

If there are any other exit routines, the field procedure is invoked *before* any of them.

- For field-decoding, when a stored value is to be field-decoded back into its original string value. This occurs for any value that is:
 - Retrieved by an SQL SELECT or FETCH statement, or by the unload phase of the REORG utility.
 - Compared to another value with the LIKE comparison operator. The value being decoded is from the column that uses the field procedure.

In this case, the field procedure is invoked *after* any edit routine or DB2 sort.

A field procedure is never invoked to process a null value, nor for a DELETE operation without a WHERE clause on a table in a segmented table space.

Recommendation: Avoid encoding blanks in a field procedure. When DB2 compares the values of two strings with different lengths, it temporarily pads the shorter string with blanks (in EBCDIC or double-byte characters, as needed) up to the length of the longer string. If the shorter string is the value of a column with a field procedure, the padding is done to the encoded value, but the pad character is **not** encoded. Therefore, if the procedure changes blanks to some other character, encoded blanks at the end of the longer string are not equal to padded blanks at the end of the shorter string. That situation can lead to errors; for example, some strings that ought to be equal might not be recognized as such. Therefore, encoding blanks in a field procedure is not recommended. 

Control blocks for execution of field procedures

Certain control blocks are used to communicate to a field procedure.

The field procedure parameter list (FPPL)

The field procedure parameter list is pointed to by register 1 on entry to a field procedure, and in turn, it contains the addresses of five other areas.



The following diagram shows those areas. The FPPL and the areas are described by the mapping macro DSNDFPPB.

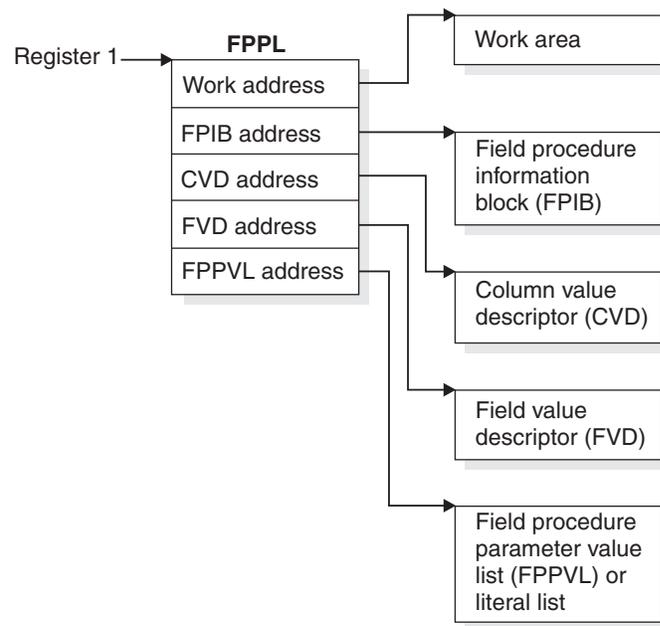


Figure 85. Field procedure parameter list



The work area for field procedures

The work area is a contiguous, uninitialized area of locally addressable, pageable, swappable, and fetch-protected storage that is obtained in storage key 7 and subpool 229.



The work area can be used by a field procedure as working storage. A new area is provided each time the procedure is invoked. The size of the area that you need depends on the way you program your field-encoding and field-decoding operations.

At field-definition time, DB2 allocates a 512-byte work area and passes the value of 512 bytes as the work area size to your routine for the field-definition operation. If subsequent field-encoding and field-decoding operations need a work area of 512 bytes or less, your field definition doesn't need to change the value as provided by

DB2. If those operations need a work area larger than 512 bytes (i.e. 1024 bytes), your field definition must change the work area size to the larger size and pass it back to DB2 for allocation.

Whenever your field procedure is invoked for encoding or decoding operations, DB2 allocates a work area based on the size (i.e. 1024 bytes) that was passed back to it. Your field definition must not use a work area larger than what is allocated by DB2, even though subsequent operations need the larger work area. 

The field procedure information block (FPIB)

The field procedure information block communicates general information to a field procedure.



The information block tells what operation is to be done, allows the field procedure to signal errors, and gives the size of the work area. It has the format shown in Table 146.

Table 146. Format of FPIB, defined in copy macro DSNDFPPB

Name	Hex offset	Data type	Description								
FPBFcode	0	Signed 2-byte integer	Function code <table border="0"> <tr> <td>Code</td> <td>Means</td> </tr> <tr> <td>0</td> <td>Field-encoding</td> </tr> <tr> <td>4</td> <td>Field-decoding</td> </tr> <tr> <td>8</td> <td>Field-definition</td> </tr> </table>	Code	Means	0	Field-encoding	4	Field-decoding	8	Field-definition
Code	Means										
0	Field-encoding										
4	Field-decoding										
8	Field-definition										
FPBWkLN	2	Signed 2-byte integer	Length of work area; the maximum is 32767 bytes.								
FPBSORC	4	Signed 2-byte integer	Reserved								
FPBRtNC	6	Character, 2 bytes	Return code set by field procedure								
FPBRsNCD	8	Character, 4 bytes	Reason code set by field procedure								
FPBTOKPT	C	Address	Address of a 40-byte area, within the work area or within the field procedure's static area, containing an error message								



The field procedure parameter value list (FPPVL)

The field procedure parameter value list communicates the literal list, supplied in the CREATE TABLE or ALTER TABLE statement, to the field procedure during field-definition.



At that time, the field procedure can reformat the FPPVL; it is the reformatted FPPVL that is stored in SYSIBM.SYSFIELDS and communicated to the field procedure during field-encoding and field-decoding as the *modified FPPVL*.

The FPPVL has the format shown in Table 147.

Table 147. Format of FPPVL, defined in copy macro DSNDFPPB

Name	Hex offset	Data type	Description
FPPVLEN	0	Signed 2-byte integer	Length in bytes of the area containing FPPVCNT and FPPVVDS. At least 254 for field-definition.
FPPVCNT	2	Signed 2-byte integer	Number of value descriptors that follow, equal to the number of parameters in the FIELDPROC clause. Zero if no parameters were listed.
FPPVVDS	4	Structure	Each parameter in the FIELDPROC clause has: 1. A signed 4-byte integer giving the length of the following value descriptor, which includes the lengths of FPVDTYPE, FPVDVLEN, and FPVDVALE. 2. A value descriptor

PSPI

Value descriptors for field procedures

A value descriptor describes the data type and other attributes of a value.

PSPI

Value descriptors are used with field procedures in these ways:

- During field-definition, they describe each constant in the field procedure parameter value list (FPPVL). The set of these value descriptors is part of the FPPVL control block.
- During field-encoding and field-decoding, the decoded (column) value and the encoded (field) value are described by the column value descriptor (CVD) and the field value descriptor (FVD).

The *column value descriptor (CVD)* contains a description of a column value and, if appropriate, the value itself. During field-encoding, the CVD describes the value to be encoded. During field-decoding, it describes the decoded value to be supplied by the field procedure. During field-definition, it describes the column as defined in the CREATE TABLE or ALTER TABLE statement.

The *field value descriptor (FVD)* contains a description of a field value and, if appropriate, the value itself. During field-encoding, the FVD describes the encoded value to be supplied by the field procedure. During field-decoding, it describes the value to be decoded. Field-definition must put into the FVD a description of the encoded value.

Value descriptors have the format shown in Table 148.

Table 148. Format of value descriptors

Name	Hex offset	Data type	Description																		
FPVDTYPE	0	Signed 2-byte integer	Data type of the value: <table border="1"> <thead> <tr> <th>Code</th> <th>Means</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>INTEGER</td> </tr> <tr> <td>4</td> <td>SMALLINT</td> </tr> <tr> <td>8</td> <td>FLOAT</td> </tr> <tr> <td>12</td> <td>DECIMAL</td> </tr> <tr> <td>16</td> <td>CHAR</td> </tr> <tr> <td>20</td> <td>VARCHAR</td> </tr> <tr> <td>24</td> <td>GRAPHIC</td> </tr> <tr> <td>28</td> <td>VARGRAPHIC</td> </tr> </tbody> </table>	Code	Means	0	INTEGER	4	SMALLINT	8	FLOAT	12	DECIMAL	16	CHAR	20	VARCHAR	24	GRAPHIC	28	VARGRAPHIC
Code	Means																				
0	INTEGER																				
4	SMALLINT																				
8	FLOAT																				
12	DECIMAL																				
16	CHAR																				
20	VARCHAR																				
24	GRAPHIC																				
28	VARGRAPHIC																				
FPVDVLEN	2	Signed 2-byte integer	<ul style="list-style-type: none"> For a varying-length string value, its maximum length For a decimal number value, its precision (byte 1) and scale (byte 2) For any other value, its length 																		
FPVDVALE	4	None	The value. The value is in external format, not DB2 internal format. If the value is a varying-length string, the first halfword is the value's actual length in bytes. This field is not present in a CVD, or in an FVD used as input to the field-definition operation. An empty varying-length string has a length of zero with no data following.																		



Field-definition (function code 8)

The input provided to the field-definition operation, and the output required, are as follows:

On entry



The registers have the information that is listed in Table 149:

Table 149. Contents of the registers on entry

Register	Contains
1	Address of the field procedure parameter list (FPPL)
2 through 12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

The contents of all other registers, and of fields not listed in the following tables, are unpredictable.

The work area consists of 512 contiguous uninitialized bytes.

The FPIB has the information that is listed in Table 150:

Table 150. Contents of the FPIB on entry

Field	Contains
FPBFCODE	8, the function code
FPBWKLN	512, the length of the work area

The CVD has the information that is listed in Table 151:

Table 151. Contents of the CVD on entry

Field	Contains										
FPVDTYPE	One of these codes for the data type of the column value: <table border="1" data-bbox="646 680 909 827"> <thead> <tr> <th>Code</th> <th>Means</th> </tr> </thead> <tbody> <tr> <td>16</td> <td>CHAR</td> </tr> <tr> <td>20</td> <td>VARCHAR</td> </tr> <tr> <td>24</td> <td>GRAPHIC</td> </tr> <tr> <td>28</td> <td>VARGRAPHIC</td> </tr> </tbody> </table>	Code	Means	16	CHAR	20	VARCHAR	24	GRAPHIC	28	VARGRAPHIC
Code	Means										
16	CHAR										
20	VARCHAR										
24	GRAPHIC										
28	VARGRAPHIC										
FPVDVLEN	The length attribute of the column										

The FPVDVALE field is omitted. The FVD provided is 4 bytes long. The FPPVL has the information that is listed in Table 152:

Table 152. Contents of the FPPVL on entry

Field	Contains
FPPVLEN	The length, in bytes, of the area containing the parameter value list. The minimum value is 254, even if there are no parameters.
FPPVCNT	The number of value descriptors that follow; zero if there are no parameters.
FPPVVDS	A contiguous set of value descriptors, one for each parameter in the parameter value list, each preceded by a 4-byte length field.



On exit



The registers must have the information that is listed in Table 153:

Table 153. Contents of the registers on exit

Register	Contains
2 through 12	The values that they contained on entry.
15	The integer zero if the column described in the CVD is valid for the field procedure; otherwise the value must <i>not</i> be zero.

The following fields must be set as shown; all other fields must remain as on entry.

The FPIB must have the information that is listed in Table 154:

Table 154. Contents of the FPIB on exit

Field	Contains
FPBWKLN	The length, in bytes, of the work area to be provided to the field-encoding and field-decoding operations; 0 if no work area is required.
FPBRTNC	An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given.
FPBTOKP	Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given.

Errors signalled by a field procedure result in SQLCODE -681 (SQLSTATE '23507'), which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKP, are also placed into the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

The FVD must have the information that is listed in Table 155:

Table 155. Contents of the FVD on exit

Field	Contains
FPVDTYPE	The numeric code for the data type of the field value. Any of the data types listed in Table 148 on page 801 is valid.
FPVDVLEN	The length of the field value.

Field FPVDVALE must not be set; the length of the FVD is 4 bytes only.

The FPPVL can be redefined to suit the field procedure, and returned as the *modified FPPVL*, subject to the following restrictions:

- The field procedure must not increase the length of the FPPVL.
- FPPVLEN must contain the actual length of the modified FPPVL, or 0 if no parameter list is returned.

The modified FPPVL is recorded in the catalog table SYSIBM.SYSFIELDS, and is passed again to the field procedure during field-encoding and field-decoding. The modified FPPVL need not have the format of a field procedure parameter list, and it need not describe constants by value descriptors.



Field-encoding (function code 0)

The input provided to the field-encoding operation, and the output required, are as follows:

On entry

 The registers have the information that is listed in Table 156:

Table 156. Contents of the registers on entry

Register	Contains
1	Address of the field procedure parameter list (FPPL); see “The field procedure parameter list (FPPL)” on page 798 for a schematic diagram.
2 through 12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

The contents of all other registers, and of fields not listed, are unpredictable.

The **work area** is contiguous, uninitialized, and of the length specified by the field procedure during field-definition.

The **FPIB** has the information that is listed in Table 157:

Table 157. Contents of the FPIB on entry

Field	Contains
FPBFCODE	0, the function code
FPBWKLN	The length of the work area

The **CVD** has the information that is listed in Table 158:

Table 158. Contents of the CVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the column value, as shown in Table 148 on page 801.
FPVDVLEN	The length of the column value.
FPVDVALE	The column value; if the value is a varying-length string, the first halfword contains its length.

The **FVD** has the information that is listed in Table 159:

Table 159. Contents of the FVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the field value.
FPVDVLEN	The length of the field value.
FPVDVALE	An area of unpredictable content that is as long as the field value.

The *modified* FPPVL, produced by the field procedure during field-definition, is provided. 

On exit

PSPI The registers have the information that is listed in Table 160:

Table 160. Contents of the registers on exit

Register	Contains
2 through 12	The values that they contained on entry.
15	The integer zero if the column described in the CVD is valid for the field procedure; otherwise the value must not be zero.

The **FVD** must contain the encoded (field) value in field **FPVDVALE**. If the value is a varying-length string, the first halfword must contain its length.

The **FPIB** can have the information that is listed in Table 161:

Table 161. Contents of the **FPIB** on exit

Field	Contains
FPBRTNC	An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given.
FPBTOKP	Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given.

Errors signalled by a field procedure result in **SQLCODE** -681 (**SQLSTATE** '23507'), which is set in the **SQL** communication area (**SQLCA**). The contents of **FPBRTNC** and **FPBRSNC**, and the error message pointed to by **FPBTOKP**, are also placed into the tokens, in **SQLCA**, as field **SQLERRMT**. The meaning of the error message is determined by the field procedure.

All other fields must remain as on entry. **PSPI**

Field-decoding (function code 4)

The input provided to the field-decoding operation, and the output required, are as follows:

On entry

PSPI The registers have the information that is listed in Table 162:

Table 162. Contents of the registers on entry

Register	Contains
1	Address of the field procedure parameter list (FPPL); see Figure 85 on page 798 for a schematic diagram.
2 through 12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

The contents of all other registers, and of fields not listed, are unpredictable.

The **work area** is contiguous, uninitialized, and of the length specified by the field procedure during field-definition.

The **FPIB** has the information that is listed in Table 163:

Table 163. Contents of the FPIB on entry

Field	Contains
FPBFCODE	4, the function code
FPBWKLN	The length of the work area

The **CVD** has the information that is listed in Table 164:

Table 164. Contents of the CVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the column value, as shown in Table 148 on page 801.
FPVDVLEN	The length of the column value.
FPVDVALE	The column value. If the value is a varying-length string, the first halfword contains its length.

The **FVD** has the information that is listed in Table 165:

Table 165. Contents of the FVD on entry

Field	Contains
FPVDTYPE	The numeric code for the data type of the field value.
FPVDVLEN	The length of the field value.
FPVDVALE	The field value. If the value is a varying-length string, the first halfword contains its length.

The *modified* FPPVL, produced by the field procedure during field-definition, is provided. 

On exit



The registers have the information that is listed in Table 166:

Table 166. Contents of the registers on exit

Register	Contains
2 through 12	The values they contained on entry.
15	The integer zero if the column described in the FVD is valid for the field procedure; otherwise the value must not be zero.

The **CVD** must contain the decoded (column) value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The **FPIB** can have the information that is listed in Table 167:

Table 167. Contents of the *FPIB* on exit

Field	Contains
FPBRTNC	An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given.
FPBTOKP	Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given.

Errors signalled by a field procedure result in `SQLCODE -681 (SQLSTATE '23507')`, which is set in the SQL communication area (`SQLCA`). The contents of `FPBRTNC` and `FPBRSNC`, and the error message pointed to by `FPBTOKP`, are also placed into the tokens, in `SQLCA`, as field `SQLERRMT`. The meaning of the error message is determined by the field procedure.

All other fields must remain as on entry. 

Log capture routines

A log capture exit routine makes DB2 log data available for recovery purposes in real time.

 The routine receives data when DB2 writes data to the active log. Your local specifications determine what the routine does with that data. The routine does not enter or return data to DB2.

Performance factor: Your log capture routine receives control often. Design it with care: a poorly designed routine can seriously degrade system performance. Whenever possible, use the instrumentation facility interface (IFI), rather than a log capture exit routine, to read data from the log.

“General guidelines for writing exit routines” on page 810 applies, but with the following exceptions to the description of execution environments:

A log capture routine can execute in either TCB mode or SRB mode, depending on the function it is performing. When in SRB mode, it must not perform any

I/O operations nor invoke any SVC services or ESTAE routines. 

Specifying the log capture routine

The module name for the log capture routine is `DSNJL004`, and its entry point is `DSNJV117`.

 The module is loaded during DB2 initialization and deleted during DB2 termination. You must link the module into either the `prefix.SDSNEXIT` or the DB2 `prefix.SDSNLOAD` library. Specify the `REPLACE` parameter of the link-edit job to replace a module that is part of the standard DB2 library for this release. The module should have attributes `AMODE(31)` and `RMODE(ANY)`. 

When log capture routines are taken

The log capture exit routine is taken in three possible situations, identified by a character in the exit parameter list.

PSPI In two of those situations, processing operates in TCB mode; in one situation, processing operates in SRB mode. The two modes have different processing capabilities, which your routine must be aware of. The character identifications, situations, and modes are:

- I=Initialization, Mode=TCB

The TCB mode allows all z/OS DFSMSdfp functions to be utilized, including ENQ, ALLOCATION, and OPEN. No buffer addresses are passed in this situation. The routine runs in supervisor state, key 7, and enabled.

This is the **only** situation in which DB2 checks a return code from the user's log capture exit routine. The DB2 subsystem is sensitive to a return code of X'20' here. **Never return X'20'** in register 15 in this situation.

- W=Write, Mode=SRB (service request block)

The SRB mode restricts the exit routine's processing capabilities. No supervisor call (SVC) instructions can be used, including ALLOCATION, OPEN, WTO, any I/O instruction, and so on. At the exit point, DB2 is running in supervisor state, key 7, and is enabled.

On entry, the exit routine has access to buffers that have log control intervals with "blocked log records". The first and last buffer address and control interval size fields can be used to determine how many buffers are being passed.

Performance consideration: All processing time that is required by the exit routine lengthens the time required to write the DB2 log. The DB2 address space usually has a high priority, and all work done in it in SRB mode precedes all TCB access. Any errors or long processing times can impact all DB2 processing and cause system-wide performance problems. The performance of your routine is *extremely* critical in this phase.

- T=Termination, Mode=TCB

Processing capabilities are the same as for initialization.

A log control interval can be passed more than once. Use the timestamp to determine the last occurrence of the control interval. This last occurrence should replace all others. The timestamp is found in the control interval. **PSPI**

Parameter lists for log capture routines

At invocation, registers are set, and the log capture routine uses the standard exit parameter list (EXPL). The reason and return codes in that list can be ignored.

PSPI

Table 168 shows the exit-specific parameter list; it is mapped by macro DSNDLOGX.

Table 168. Log capture routine specific parameter list

Name	Hex offset	Data type	Description
LOGXEYE	00	Character, 4 bytes	Eye catcher: LOGX

Table 168. Log capture routine specific parameter list (continued)

Name	Hex offset	Data type	Description
LOGXLNG	04	Signed 2-byte integer	Length of parameter list
	06		Reserved
	08		Reserved
LOGXTYPE	10	Character, 1 byte	Situation identifier: I Initialization W Write T Termination P Partial control interval (CI) call
LOGXFLAG	11	Hex	Mode identifier. X'00' SRB mode X'01' TCB mode
LOGXSRBA	12	Character, 6 bytes	First log RBA, set when DB2 is started. The value remains constant while DB2 is active.
LOGXARBA	18	Character, 6 bytes	Highest log archive RBA used. The value is updated after completion of each log archive operation.
	1E		Reserved
LOGXRBUF	20	Character, 8 bytes	Range of consecutive log buffers: Address of first log buffer Address of last log buffer
LOGXBUFL	28	Signed 4-byte integer	Length of single log buffer (constant 4096)
LOGXSSID	2C	Character, 4 bytes	DB2 subsystem ID, 4 characters left justified
LOGXSTIM	30	Character, 8 bytes	DB2 subsystem startup time (TIME format with DEC option: 0CYDDDFHHMMSSTH)
LOGXREL	38	Character, 3 bytes	DB2 subsystem release level
LOGXMAXB	3B	Character, 1 byte	Maximum number of buffers that can be passed on one call. The value remains constant while DB2 is active.
	3C	8 bytes	Reserved
LOGXUSR1	44	Character, 4 bytes	First word of a doubleword work area for the user routine. (The content is not changed by DB2.)
LOGXUSR2	48	Character, 4 bytes	Second word of user work area.



Routines for dynamic plan selection in CICS

You can create application packages and plans that allow application programs to access DB2 data at execution time. In a CICS environment, you can design CICS transactions around the application packages and plans or use dynamic plan allocation.

PSPI You can enable dynamic plan allocation by using one of the following techniques:

- Use DB2 packages and versioning to manage the relationship between CICS transactions and DB2 plans. This technique can help minimize plan outage time, processor time, and catalog contention.
- Use a dynamic plan exit routine to determine the plan to use for each CICS transaction.

Recommendation: Use DB2 packages and versioning, instead of a CICS dynamic plan exit routine, for dynamic plan allocation. For more information, see *CICS*

Transaction Server for z/OS DB2 Guide. **PSPI**

Routine for CICS transaction invocation stored procedure

The DB2-supplied CICS transaction routine stored procedure invokes a user exit that you use to change values that the stored procedure caller provides.

General guidelines for writing exit routines

The rules, requirements, and suggestions in these topics apply to most of the foregoing exit routines.

PSPI **Important:** Using an exit routine requires coordination with your system programmers. An exit routine runs as an extension of DB2 and has all the privileges of DB2. It can impact the security and integrity of the database. Conceivably, an exit routine could also expose the integrity of the operating system. Instructions for avoiding that exposure can be found in the appropriate z/OS publication. **PSPI**

Coding rules for exit routines

You must follow these rules and requirements when coding an exit routine for DB2.

- **PSPI** It must be written in assembler.
- It must reside in an authorized program library, either the library containing DB2 modules (*prefix.SDSNLOAD*) or in a library concatenated ahead of *prefix.SDSNLOAD* in the procedure for the database services started task (the procedure named *ssnmDBM1*, where *ssnm* is the DB2 subsystem name). Authorization routines must be accessible to the *ssnmMSTR* procedure. For all routines, we recommend using the library *prefix.SDSNEXIT*, which is concatenated ahead of *prefix.SDSNLOAD* in both started-task procedures.
- Routines that are listed in Table 169 *must* have the names shown. The name of other routines should not start with “DSN”, to avoid conflict with the DB2 modules.

Table 169. Required load module name

Type of routine	Required load module name
Date	DSNXVDTX
Time	DSNXVTMX
Connection	DSN3@ATH
Sign-on	DSN3@SGN

- It must be written to be reentrant and must restore registers before return.
- It must be link-edited with the REENTRANT parameter.
- It must be written and link-edited to execute AMODE(31),RMODE(ANY).
- It must not invoke any DB2 services—for example, through SQL statements.
- It must not invoke any SVC services or ESTAE routines.

Even though DB2 has functional recovery routines of its own, you can establish your own functional recovery routine (FRR), specifying MODE=FULLXM and EUT=YES. 

Modifying exit routines

Because exit routines operate as extensions of DB2, they should not be changed or modified while DB2 is running.

Execution environment for exit routines

Exit routines are invoked by standard CALL statements.

 With some exceptions, which are noted under “General Considerations” in the description of particular types of routine, the execution environment is:

- Supervisor state
- Enabled for interrupts
- PSW key 7
- No MVS locks held
- For local requests, under the TCB of the application program that requested the DB2 connection
- For remote requests, under a TCB within the DB2 distributed data facility address space
- 31-bit addressing mode
- Cross-memory mode

In cross-memory mode, the current primary address space is not equal to the home address space. Therefore, some z/OS macro services you cannot use at all, and some you can use only with restrictions. For more information about cross-memory restrictions for macro instructions, which macros can be used fully, and the complete description of each macro, refer to the appropriate z/OS publication. 

Registers at invocation for exit routines



When DB2 passes control to an exit routine, the registers are set as listed in Table 170:

Table 170. Contents of registers when DB2 passes control to an exit routine

Register	Contains
1	Address of pointer to the exit parameter list. <i>For a field procedure</i> , the address is that of the field procedure parameter list.

Table 170. Contents of registers when DB2 passes control to an exit routine (continued)

Register	Contains
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.



Parameter lists for exit routines



Register 1 points to the address of parameter list EXPL, described by macro DSNDEXPL and shown in Figure 86. The word following points to a second parameter list, which differs for each type of exit routine.

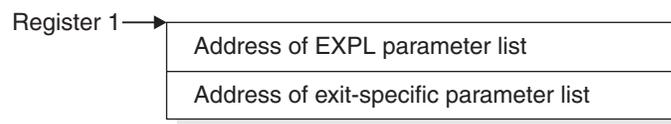


Figure 86. Use of register 1 on invoking an exit routine. (Field procedures and translate procedures do not use the standard exit-specific parameter list.)

Table 171 shows the EXPL parameter list. Its description is given by macro DSNDEXPL.

Table 171. Contents of EXPL parameter list

Name	Hex offset	Data type	Description
EXPLWA	0	Address	Address of a work area to be used by the routine
EXPLWL	4	Signed 4-byte integer	Length of the work area. The value is: 2048 for connection routines and sign-on routines 512 for date and time routines and translate procedures (see Note 1). 256 for edit, validation, and log capture routines
EXPLRSV1	8	Signed 2-byte integer	Reserved
EXPLRC1	A	Signed 2-byte integer	Return code
EXPLRC2	C	Signed 4-byte integer	Reason code
EXPLARC	10	Signed 4-byte integer	Used only by connection routines and sign-on routines
EXPLSSNM	14	Character, 8 bytes	Used only by connection routines and sign-on routines
EXPLCONN	1C	Character, 8 bytes	Used only by connection routines and sign-on routines

Table 171. Contents of EXPL parameter list (continued)

Name	Hex offset	Data type	Description
EXPLTYPE	24	Character, 8 bytes	Used only by connection routines and sign-on routines
EXPLSITE	2C	Character, 16 bytes	For SNA protocols, this is the location name of the requesting location or <luname>. For TCP/IP protocols, this is the dotted decimal IP address of the requester.
EXPLLUNM	3C	Character, 8 bytes	For SNA protocols, the locally known LU name of the requesting location. For TCP/IP protocols, the character string 'TCPIP'.
EXPLNTID	44	Character, 17 bytes	For SNA protocols, the fully qualified network name of the requesting location. For TCP/IP protocols, field reserved.
EXPLVIDS			DB2 version identifier

Notes: When translating a string of type PC MIXED, a translation procedure has a work area of 256 bytes plus the length attribute of the string.



Row formats for edit and validation routines

In writing an edit or validation routine, you must be aware of the format in which DB2 stores the rows of tables. This topic describes the special features of that format.

Column boundaries for edit and validation routines

DB2 stores columns contiguously, regardless of word boundaries in physical storage, except LOB and XML columns. LOB or XML values are not stored contiguously; an indicator column is stored in a base table in place of the LOB or XML values.



You cannot specify edit procedures for any table that contains a LOB column or a ROWID column. In addition, LOB values are not available to validation routines; indicator columns and ROWID columns represent LOB columns as input to a validation procedure.

Similarly, you cannot specify edit procedures for any table that contains an XML column. XML values are not available to validation routines. DOCID and XML indicator columns represent XML columns as input to a validation procedure.



Null values for edit procedures, field procedures, and validation routines

If null values are allowed for a column, an extra byte is stored before the actual column value.

PSPI This byte is X'00' if the column value is not null; it is X'FF' if the value is null. This extra byte is included in the column length attribute (parameter FFMTFLEN in Table 180 on page 818). **PSPI**

Fixed-length rows for edit and validation routines

If all columns in a table are fixed-length, its rows are stored in fixed-length format. The rows are byte strings.

PSPI

Example: The sample project activity table has five fixed-length columns. The first two columns do not allow nulls; the last three do. Table 172 shows a row in the table.

Table 172. A row in fixed-length format

Column 1	Column 2	Column 3		Column 4		Column 5	
MA2100	10	00	0.5	00	820101	00	821101

PSPI

Varying-length rows for edit and validation routines

The rows of a table with varying-length columns are varying-length rows if they contain varying-length values. In basic row format, each varying-length value has a 2-byte length field in front of it. Those 2 bytes are not included in the column length attribute (FFMTFLEN).

PSPI

Table 173 shows a row of the sample department table in basic row format. The first value in the DEPTNAME column indicates the column length as a hexadecimal value.

Table 173. A varying-length row in basic row format in the sample department table

DEPTNO	DEPTNAME		MGRNO		ADMRDEPT	LOCATION	
C01	0012	Information center	00	000030	A00	00	New York

Varying-length columns have no gaps after them. Hence, columns that appear after varying-length columns are at variable offsets in the row. To get to such a column, you must scan the columns sequentially after the first varying-length column. An empty string has a length of zero with no data following.

ROWID and indicator columns are treated like varying length columns. Row IDs are VARCHAR(17). A LOB indicator column is VARCHAR(4), and an XML indicator column is VARCHAR(6). It is stored in a base table in place of a LOB or XML column, and indicates whether the LOB or XML value for the column is null or zero length.

In reordered row format, if a table has any varying-length columns, all fixed length columns are placed at the beginning of the row, followed by the offsets to the varying length columns, followed by the values of the varying length columns.

Table 174 shows the same row of the sample department table, but in reordered row format. The value in the offset column indicates the offset value as a hexadecimal value.

Table 174. A varying-length row in reordered row format in the sample department table

DEPTNO	MGRNO		ADMRDEPT	LOCATION		Offset column	DEPTNAME
C01	00	000030	A00	00	New York	20	Information center



Varying-length rows with nulls for edit and validation routines

A varying-length column can also allow null values. In basic row format, the value in the length field includes the length of the null indicator byte but does not include the length field itself.



Table 175 shows how the row in Table 173 on page 814 would look in storage if nulls were allowed in DEPTNAME. The first value in the DEPTNAME column indicates the column length as a hexadecimal value.

Table 175. A varying-length row in basic row format in the sample department table

DEPTNO	DEPTNAME		MGRNO		ADMRDEPT	LOCATION	
C01	0013	Information center	00	000030	A00	00	New York

An empty string has a length of one, a X'00' null indicator, and no data following.

In reordered row format, if a table has any varying-length columns, with or without nulls, all fixed length columns are placed at the beginning of the row, followed by the offsets to the varying length columns, followed by the values of the varying length columns. 

EDITPROCs and VALIDPROCs for handling basic and reordered row formats

Checking the row format type (RFMTTYPE) ensures that edit procedures (EDITPROC) and validation procedures (VALIDPROC) produce predictable results.

If you write new edit and validation routines on tables with rows in basic row format (BRF) or reordered row format (RRF), make sure that EDITPROCs and VALIDPROCs are coded to check RFMTTYPE and handle both BRF and RRF formats.

If an EDITPROC or VALIDPROC handles only RRF, make sure that it checks RFMTTYPE and returns an error or warning if it detects BRF. If an EDITPROC or VALIDPROC that handles only BRF is to be used on tables in RRF, make sure that it checks RFMTTYPE and returns an error or warning if it detects RRF.

Converting basic row format table spaces with edit and validation routines to reordered row format

If you migrate table spaces to DB2 Version 9.1, you cannot convert the table spaces with edit and validation routines from basic row format to reordered row format directly. You must execute additional steps to convert the table spaces.

Converting basic row format table spaces that have edit routines to reordered row format

If a table space cannot be directly converted to reordered row format because some tables in the table space have edit routines, execute the following steps for each table that has an edit routine:

PSPI

1. Use the UNLOAD utility to unload data from the table or tables that have edit routines.
2. Use the DROP statement to drop the table or tables that have edit routines.
3. Make any necessary modifications to the edit routines so that they can be used with rows in reordered row format.
4. Use the REORG utility to reorganize the table space. Using the REORG utility converts the table space to reordered row format.
5. Re-create tables with your modified edit routines. Also re-create any additional related objects, such as indexes and check constraints.
6. Use the LOAD RESUME utility to load the data into the tables that have the modified edit routines.

PSPI

Converting basic row format table spaces that have validation routines to reordered row format

If a table space cannot be directly converted to reordered row format because some tables in the table space have validation routines, execute the following steps for each table that has a validation routine:

PSPI

1. Use the ALTER TABLE statement to alter the validation routine to NULL.
2. Run the REORG utility or the LOAD REPLACE utility to convert the table space to reordered row format.
3. Make any necessary modifications to the validation routine so that it can be used with rows in reordered row format.
4. Use the ALTER TABLE statement to add the modified validation routine to the converted table.

PSPI

Dates, times, and timestamps for edit and validation routines

The values in columns with data types of DATE, TIME, and TIMESTAMP are stored in the formats that are shown in the following tables. For each format, each byte consists of two packed decimal digits.

PSPI

Table 176 shows the DATE format, which consists of 4 total bytes

Table 176. DATE format

Year	Month	Day
2 bytes	1 byte	1 byte

Table 177 shows the TIME format, which consists of 3 total bytes.

Table 177. TIME format

Hours	Minutes	Seconds
1 byte	1 byte	1 byte

Table 178 shows the TIMESTAMP format, which consists of 10 total bytes.

Table 178. TIMESTAMP format

Year	Month	Day	Hours	Minutes	Seconds	Microseconds
2 bytes	1 byte	1 byte	1 byte	1 byte	1 byte	3 bytes



Parameter list for row format descriptions

DB2 passes a description of the row format to an edit or validation routine through a parameter list, generated by macro DSNDROW. The description includes both the general row characteristics and the characteristics of each column.



DSNDROW defines the columns in the order as they are defined in the CREATE TABLE statement or possibly the ALTER TABLE statement. For rows in the reordered row format, the new column order in DSNDROW does not necessarily correspond to the order in which the columns are stored in the row.

Table 179 shows the general row description.

Table 179. Description of a row format

Name	Hex offset	Data type	Description
RFMTNFLD	0	Signed fullword integer	Number of columns in a row
RFMTAFLD	4	Address	Address of a list of column descriptions
RFMTTYPE	8	Character, 1 byte	Row type: X'00' = row with fixed-length columns X'04' = row with varying-length columns in basic row format X'08' = row with varying-length columns in reordered row format
	9	Character, 3 bytes	Reserved

|
|
|
|
|
|

Table 180 shows the description of each column.

Table 180. Description of a column format

Name	Hex offset	Data type	Description
FFMTFLEN	0	Signed fullword integer	Column length attribute
FFMTFTYP	4	Character, 1 byte	Data type code
FFMTNULL	5	Character, 1 byte	Data attribute: X'00' = Null values are allowed. X'04' = Null values are not allowed.
FFMTFNAM	6	Character, 18 bytes	Column name

Table 181 shows a description of data type codes and length attributes.

Table 181. Description of data type codes and length attributes

Data type	Code (FFMTFTYP)	Length attribute (FFMTFLEN)
BIGINT	X'32'	8
BINARY	X'34'	Length of string
VARBIN	X'38'	Length of string
DECFLOAT	X'40'	8 for DECFLOAT(16) or 16 for DECFLOAT(34)
INTEGER	X'00'	4
SMALLINT	X'04'	2
FLOAT (single precision)	X'08'	4
FLOAT (double precision)	X'08'	8
DECIMAL	X'0C'	INTEGER($p/2$), where p is the precision
CHAR	X'10'	The length of the string
VARCHAR	X'14'	The length of the string
DATE	X'20'	4
TIME	X'24'	3
TIMESTAMP	X'28'	10
ROWID	X'2C'	17
INDICATOR COLUMN	X'30'	4 for a LOB indicator column or 6 for an XML indicator column



DB2 codes for numeric data in edit and validation routines

DB2 stores numeric data in a specially encoded format that is called *DB2-coded*.



To retrieve numeric data in its original form, you must DB2-decode it, according to its data type, as is listed in Table 182 on page 819:

Table 182. DB2 decoding procedure according to data type

Data type	DB2 decoding procedure
SMALLINT	Invert the sign bit (high-order bit).
	Value Meaning
	8001 0001 (+1 decimal) 7FF3 FFF3 (-13 decimal)
INTEGER	Invert the sign bit (high-order bit).
	Value Meaning
	800001F2 000001F2 (+498 decimal) 7FFFFFFF85 FFFFFFFF85 (-123 decimal)
FLOAT	If the sign bit (high-order bit) is 1, invert only that bit. Otherwise, invert all bits.
	Value Meaning
	C110000000000000 4110000000000000 (+1.0 decimal) 3EEEEEEEEEEEE C1100000000000000 (-1.0 decimal)
DECIMAL	Save the high-order hexadecimal digit (sign digit). Shift the number to the left one hexadecimal digit. If the sign digit is X'F', put X'C' in the low-order position. Otherwise, invert all bits in the number and put X'D' in the low-order position.
	Value Meaning
	F001 001C (+1) 0FFE 001D (-1)
BIGINT	Invert the sign bit (high order bit).
	Value Meaning
	8000000000000854 0000000000000854 (2132 decimal) 7FFFFFFFFFFFFE0 FFFFFFFFFFFFFE0 (-32 decimal)
DECFLOAT	Convert and return a DECFLOAT representation of a number or string representation of a number. For more information about DECFLOAT, see "Sortable decimal formats" in <i>DB2 Diagnosis Guide and Reference</i> .
	Value Meaning
	D8F77D0000000000C 222C000000001E80 (+7.500 decfloat) 270882FFFFFFFFF2 A2300000000003D0 (-7.50 decfloat)



RACF access control module

The RACF access control module allows you to use RACF as an alternative to DB2 authorization checking for DB2 objects, authorities, and utilities.

PSPI You can activate the RACF access control module at the DB2 access control authorization exit point (DSNX@XAC), where you can replace the default routine. The RACF access control module is provided as an assembler source module in the DSNXRAC member of DB2.SDSNSAMP.

| The RACF access control module (DSNXRXAC) does not provide full support of
| role on z/OS 1.7.

For more information about the RACF access control module, see *DB2 RACF Access Control Module Guide*. **PSPI**

Appendix B. Stored procedures for administration

DB2 provides several stored procedures that you can call in your application programs to perform administrative functions.

Restriction: These stored procedures do not propagate the transaction identifier (XID) of the thread. These stored procedures run under a new private context rather than under the native context of the task that called it.

Tip: DB2 also provides stored procedures for other functions. See *DB2 Utility Guide and Reference* and *DB2 Application Programming and SQL Guide* for additional stored procedures.

Related reference

“ADMIN_TASK_ADD” on page 379

“ADMIN_TASK_REMOVE” on page 396

DSNACICS stored procedure

The CICS transaction invocation stored procedure (DSNACICS) invokes CICS server programs.

GUPI DSNACICS gives workstation applications a way to invoke CICS server programs while using TCP/IP as their communication protocol. The workstation applications use TCP/IP and DB2 Connect to connect to a DB2 for z/OS subsystem, and then call DSNACICS to invoke the CICS server programs.

The DSNACICS input parameters require knowledge of various CICS resource definitions with which the workstation programmer might not be familiar. For this reason, DSNACICS invokes the DSNACICX user exit routine. The system programmer can write a version of DSNACICX that checks and overrides the parameters that the DSNACICS caller passes. If no user version of DSNACICX is provided, DSNACICS invokes the default version of DSNACICX, which does not modify any parameters.

Environment

DSNACICS runs in a WLM-established stored procedure address space and uses the Resource Recovery Services attachment facility to connect to DB2.

If you use CICS Transaction Server for OS/390® Version 1 Release 3 or later, you can register your CICS system as a resource manager with recoverable resource management services (RRMS). When you do that, changes to DB2 databases that are made by the program that calls DSNACICS and the CICS server program that DSNACICS invokes are in the same two-phase commit scope. This means that when the calling program performs an SQL COMMIT or ROLLBACK, DB2 and RRS inform CICS about the COMMIT or ROLLBACK.

If the CICS server program that DSNACICS invokes accesses DB2 resources, the server program runs under a separate unit of work from the original unit of work that calls the stored procedure. This means that the CICS server program might deadlock with locks that the client program acquires.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on stored procedure DSNACICS
- Ownership of the stored procedure
- SYSADM authority

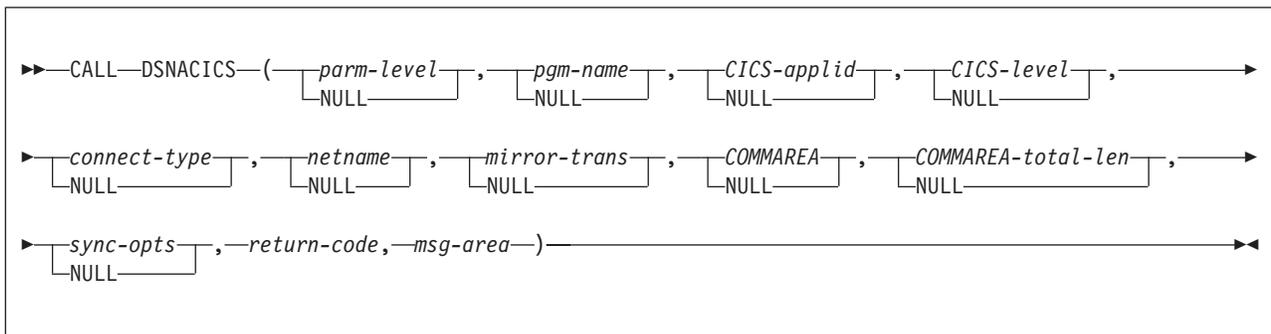
The CICS server program that DSNACICS calls runs under the same user ID as DSNACICS. That user ID depends on the SECURITY parameter that you specify when you define DSNACICS.

The DSNACICS caller also needs authorization from an external security system, such as RACF, to use CICS resources.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure.

Because the linkage convention for DSNACICS is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.



Option descriptions

parm-level

Specifies the level of the parameter list that is supplied to the stored procedure. This is an input parameter of type INTEGER. The value must be 1.

pgm-name

Specifies the name of the CICS program that DSNACICS invokes. This is the name of the program that the CICS mirror transaction calls, not the CICS transaction name.

This is an input parameter of type CHAR(8).

CICS-applid

Specifies the applid of the CICS system to which DSNACICS connects.

This is an input parameter of type CHAR(8).

CICS-level

Specifies the level of the target CICS subsystem:

- 1 The CICS subsystem is CICS for MVS/ESA™ Version 4 Release 1, CICS Transaction Server for OS/390 Version 1 Release 1, or CICS Transaction Server for OS/390 Version 1 Release 2.
- 2 The CICS subsystem is CICS Transaction Server for OS/390 Version 1 Release 3 or later.

This is an input parameter of type INTEGER.

connect-type

Specifies whether the CICS connection is generic or specific. Possible values are GENERIC or SPECIFIC.

This is an input parameter of type CHAR(8).

netname

If the value of *connection-type* is SPECIFIC, specifies the name of the specific connection that is to be used. This value is ignored if the value of *connection-type* is GENERIC.

This is an input parameter of type CHAR(8).

mirror-trans

Specifies the name of the CICS mirror transaction to invoke. This mirror transaction calls the CICS server program that is specified in the *pgm-name* parameter. *mirror-trans* must be defined to the CICS server region, and the CICS resource definition for *mirror-trans* must specify DFHMIRS as the program that is associated with the transaction.

If this parameter contains blanks, DSNACICS passes a mirror transaction parameter value of null to the CICS EXCI interface. This allows an installation to override the transaction name in various CICS user-replaceable modules. If a CICS user exit routine does not specify a value for the mirror transaction name, CICS invokes CICS-supplied default mirror transaction CSML.

This is an input parameter of type CHAR(4).

COMMAREA

Specifies the communication area (COMMAREA) that is used to pass data between the DSNACICS caller and the CICS server program that DSNACICS calls.

This is an input/output parameter of type VARCHAR(32704). In the length field of this parameter, specify the number of bytes that DSNACICS sends to the CICS server program.

commarea-total-len

Specifies the total length of the COMMAREA that the server program needs.

This is an input parameter of type INTEGER. This length must be greater than or equal to the value that you specify in the length field of the COMMAREA parameter and less than or equal to 32704. When the CICS server program completes, DSNACICS passes the server program's entire COMMAREA, which is *commarea-total-len* bytes in length, to the stored procedure caller.

sync-opts

Specifies whether the calling program controls resource recovery, using two-phase commit protocols that are supported by RRS. Possible values are:

- 1 The client program controls commit processing. The CICS server region does not perform a syncpoint when the server program returns control to CICS. Also, the server program cannot take any explicit syncpoints. Doing so causes the server program to abnormally terminate.

- 2 The target CICS server region takes a syncpoint on successful completion of the server program. If this value is specified, the server program can take explicit syncpoints.

When CICS has been set up to be an RRS resource manager, the client application can control commit processing using SQL COMMIT requests. DB2 for z/OS ensures that CICS is notified to commit any resources that the CICS server program modifies during two-phase commit processing.

When CICS has not been set up to be an RRS resource manager, CICS forces syncpoint processing of all CICS resources at completion of the CICS server program. This commit processing is not coordinated with the commit processing of the client program.

This option is ignored when *CICS-level* is 1. This is an input parameter of type INTEGER.

return-code

Return code from the stored procedure. Possible values are:

- 0 The call completed successfully.
- 12 The request to run the CICS server program failed. The *msg-area* parameter contains messages that describe the error.

This is an output parameter of type INTEGER.

msg-area

Contains messages if an error occurs during stored procedure execution. The first messages in this area are generated by the stored procedure. Messages that are generated by CICS or the DSNACICX user exit routine might follow the first messages. The messages appear as a series of concatenated, viewable text strings.

This is an output parameter of type VARCHAR(500).

User exit routine

DSNACICS always calls user exit routine DSNACICX. You can use DSNACICX to change the values of DSNACICS input parameters before you pass those parameters to CICS. If you do not supply your own version of DSNACICX, DSNACICS calls the default DSNACICX, which modifies no values and does an immediate return to DSNACICS. The source code for the default version of DSNACICX is in member DSNASCIX in data set *prefix.SDSNSAMP*. The source code for a sample version of DSNACICX that is written in COBOL is in member DSNASCIO in data set *prefix.SDSNSAMP*.

Example

The following PL/I example shows the variable declarations and SQL CALL statement for invoking the CICS transaction that is associated with program CICSPGM1.

```
/* *****  
/* DSNACICS PARAMETERS */  
/* *****  
DECLARE PARM_LEVEL BIN FIXED(31);  
DECLARE PGM_NAME CHAR(8);  
DECLARE CICS_APPLID CHAR(8);  
DECLARE CICS_LEVEL BIN FIXED(31);  
DECLARE CONNECT_TYPE CHAR(8);  
DECLARE NETNAME CHAR(8);  
DECLARE MIRROR_TRANS CHAR(4);
```

```

DECLARE COMMAREA_TOTAL_LEN BIN FIXED(31);
DECLARE SYNC_OPTS BIN FIXED(31);
DECLARE RET_CODE BIN FIXED(31);
DECLARE MSG_AREA CHAR(500) VARYING;

DECLARE1 COMMAREA BASED(P1),
      3 COMMAREA_LEN BIN FIXED(15),
      3 COMMAREA_INPUT CHAR(30),
      3 COMMAREA_OUTPUT CHAR(100);

/*****
/* INDICATOR VARIABLES FOR DSNACICS PARAMETERS */
*****/
DECLARE 1 IND_VARS,
      3 IND_PARM_LEVEL BIN FIXED(15),
      3 IND_PGM_NAME BIN FIXED(15),
      3 IND_CICS_APPLID BIN FIXED(15),
      3 IND_CICS_LEVEL BIN FIXED(15),
      3 IND_CONNECT_TYPE BIN FIXED(15),
      3 IND_NETNAME BIN FIXED(15),
      3 IND_MIRROR_TRANS BIN FIXED(15),
      3 IND_COMMAREA_LEN BIN FIXED(15),
      3 IND_COMMAREA_TOTAL_LEN BIN FIXED(15),
      3 IND_SYNC_OPTS BIN FIXED(15),
      3 IND_RET_CODE BIN FIXED(15),
      3 IND_MSG_AREA BIN FIXED(15);

/*****
/* LOCAL COPY OF COMMAREA */
*****/
DECLARE P1 POINTER;
DECLARE COMMAREA_STG CHAR(130) VARYING;

/*****
/* ASSIGN VALUES TO INPUT PARAMETERS PARM_LEVEL, PGM_NAME,
/* MIRROR TRANS, COMMAREA, COMMAREA_TOTAL_LEN, AND SYNC_OPTS.
/* SET THE OTHER INPUT PARAMETERS TO NULL. THE DSNACICX
/* USER EXIT MUST ASSIGN VALUES FOR THOSE PARAMETERS.
*****/
PARM_LEVEL = 1;
IND_PARM_LEVEL = 0;

PGM_NAME = 'CICSPGM1';
IND_PGM_NAME = 0;

MIRROR_TRANS = 'MIRT';
IND_MIRROR_TRANS = 0;

P1 = ADDR(COMMAREA_STG);
COMMAREA_INPUT = 'THIS IS THE INPUT FOR CICSPGM1';
COMMAREA_OUTPUT = ' ';
COMMAREA_LEN = LENGTH(COMMAREA_INPUT);
IND_COMMAREA_LEN = 0;

COMMAREA_TOTAL_LEN = COMMAREA_LEN + LENGTH(COMMAREA_OUTPUT);
IND_COMMAREA_TOTAL_LEN = 0;

SYNC_OPTS = 1;
IND_SYNC_OPTS = 0;

IND_CICS_APPLID = -1;
IND_CICS_LEVEL = -1;
IND_CONNECT_TYPE = -1;
IND_NETNAME = -1;
/*****
/* INITIALIZE
OUTPUT PARAMETERS TO NULL.
*****/

```

```

IND_RETCODE = -1;
IND_MSG_AREA= -1;
/*****
/* CALL DSNACICS TO INVOKE CICS PGM1.      */
/*****
EXEC SQL
CALL SYSPROC.DSNACICS(:PARM_LEVEL          :IND_PARM_LEVEL,
                     :PGM_NAME            :IND_PGM_NAME,
                     :CICS_APPLID        :IND_CICS_APPLID,
                     :CICS_LEVEL         :IND_CICS_LEVEL,
                     :CONNECT_TYPE       :IND_CONNECT_TYPE,
                     :NETNAME            :IND_NETNAME,
                     :MIRROR_TRANS       :IND_MIRROR_TRANS,
                     :COMMAREA_STG       :IND_COMMAREA,
                     :COMMAREA_TOTAL_LEN :IND_COMMAREA_TOTAL_LEN,
                     :SYNC_OPTS          :IND_SYNC_OPTS,
                     :RET_CODE           :IND_RETCODE,
                     :MSG_AREA           :IND_MSG_AREA);

```

Output

DSNACICS places the return code from DSNACICS execution in the *return-code* parameter. If the value of the return code is non-zero, DSNACICS puts its own error messages and any error messages that are generated by CICS and the DSNACICX user exit routine in the *msg-area* parameter.

The *COMMAREA* parameter contains the *COMMAREA* for the CICS server program that DSNACICS calls. The *COMMAREA* parameter has a *VARCHAR* type. Therefore, if the server program puts data other than character data in the *COMMAREA*, that data can become corrupted by code page translation as it is passed to the caller. To avoid code page translation, you can change the *COMMAREA* parameter in the *CREATE PROCEDURE* statement for DSNACICS to *VARCHAR(32704) FOR BIT DATA*. However, if you do so, the client program might need to do code page translation on any character data in the *COMMAREA* to make it readable.

Restrictions

Because DSNACICS uses the distributed program link (DPL) function to invoke CICS server programs, server programs that you invoke through DSNACICS can contain only the CICS API commands that the DPL function supports. The list of supported commands is documented in *CICS Transaction Server for z/OS Application Programming Reference*.

DSNACICS does not propagate the transaction identifier (XID) of the thread. The stored procedure runs under a new private context rather than under the native context of the task that called it.

Debugging

If you receive errors when you call DSNACICS, ask your system administrator to add a *DSNDUMP DD* statement in the startup procedure for the address space in which DSNACICS runs. The *DSNDUMP DD* statement causes DB2 to generate an SVC dump whenever DSNACICS issues an error message. 

The DSNACICX user exit routine

Use DSNACICX to change the values of DSNACICS input parameters before you pass those parameters to CICS.

General considerations

The DSNACICX exit routine must follow these rules:

- It can be written in assembler, COBOL, PL/I, or C.
- It must follow the Language Environment calling linkage when the caller is an assembler language program.
- The load module for DSNACICX must reside in an authorized program library that is in the STEPLIB concatenation of the stored procedure address space startup procedure.

You can replace the default DSNACICX in the *prefix*.SDSNLOAD, library, or you can put the DSNACICX load module in a library that is ahead of *prefix*.SDSNLOAD in the STEPLIB concatenation. It is recommended that you put DSNACICX in the *prefix*.SDSNEXIT library. Sample installation job DSNTIJEX contains JCL for assembling and link-editing the sample source code for DSNACICX into *prefix*.SDSNEXIT. You need to modify the JCL for the libraries and the compiler that you are using.

- The load module must be named DSNACICX.
- The exit routine must save and restore the caller's registers. Only the contents of register 15 can be modified.
- It must be written to be reentrant and link-edited as reentrant.
- It must be written and link-edited to execute as AMODE(31),RMODE(ANY).
- DSNACICX can contain SQL statements. However, if it does, you need to change the DSNACICS procedure definition to reflect the appropriate SQL access level for the types of SQL statements that you use in the user exit routine.

Specifying the exit routine

DSNACICS always calls an exit routine named DSNACICX. DSNACICS calls your DSNACICX exit routine if it finds it before the default DSNACICX exit routine. Otherwise, it calls the default DSNACICX exit routine.

When the exit routine is taken

The DSNACICX exit routine is taken whenever DSNACICS is called. The exit routine is taken before DSNACICS invokes the CICS server program.

Loading a new version of the exit routine

DB2 loads DSNACICX only once, when DSNACICS is first invoked. If you change DSNACICX, you can load the new version by quiescing and then resuming the WLM application environment for the stored procedure address space in which DSNACICS runs:

```
VARY WLM,APPLENV=DSNACICS-applenv-name,QUIESCE VARY
WLM,APPLENV=DSNACICS-applenv-name,RESUME
```

Parameter list

At invocation, registers are set as described in the following table

Table 183. Registers at invocation of DSNACICX

Register	Contains
1	Address of pointer to the exit parameter list (XPL).
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

The following table shows the contents of the DSNACICX exit parameter list, XPL. Member DSNDXPL in data set *prefix.SDSNMACS* contains an assembler language mapping macro for XPL. Sample exit routine DSNASCIO in data set *prefix.SDSNSAMP* includes a COBOL mapping macro for XPL.

Table 184. Contents of the XPL exit parameter list

Name	Hex offset	Data type	Description	Corresponding DSNACICS parameter
XPL_EYEC	0	Character, 4 bytes	Eye-catcher: 'XPL '	
XPL_LEN	4	Character, 4 bytes	Length of the exit parameter list	
XPL_LEVEL	8	4-byte integer	Level of the parameter list	<i>parm-level</i>
XPL_PGMNAME	C	Character, 8 bytes	Name of the CICS server program	<i>pgm-name</i>
XPL_CICSAPPLID	14	Character, 8 bytes	CICS VTAM applid	<i>CICS-applid</i>
XPL_CICSLEVEL	1C	4-byte integer	Level of CICS code	<i>CICS-level</i>
XPL_CONNECTTYPE	20	Character, 8 bytes	Specific or generic connection to CICS	<i>connect-type</i>
XPL_NETNAME	28	Character, 8 bytes	Name of the specific connection to CICS	<i>netname</i>
XPL_MIRRORTRAN	30	Character, 8 bytes	Name of the mirror transaction that invokes the CICS server program	<i>mirror-trans</i>
XPL_COMMAREAPTR	38	Address, 4 bytes	Address of the COMMAREA	¹
XPL_COMMINLEN	3C	4-byte integer	Length of the COMMAREA that is passed to the server program	²
XPL_COMMTOTLEN	40	4-byte integer	Total length of the COMMAREA that is returned to the caller	<i>commarea-total-len</i>
XPL_SYNCOPTS	44	4-byte integer	Syncpoint control option	<i>sync-opts</i>

Table 184. Contents of the XPL exit parameter list (continued)

Name	Hex offset	Data type	Description	Corresponding DSNACICS parameter
XPL_RETCODE	48	4-byte integer	Return code from the exit routine	<i>return-code</i>
XPL_MSGLEN	4C	4-byte integer	Length of the output message area	<i>return-code</i>
XPL_MSGAREA	50	Character, 256 bytes	Output message area	<i>msg-area</i> ³

Notes:

1. The area that this field points to is specified by DSNACICS parameter *COMMAREA*. This area does not include the length bytes.
2. This is the same value that the DSNACICS caller specifies in the length bytes of the *COMMAREA* parameter.
3. Although the total length of *msg-area* is 500 bytes, DSNACICX can use only 256 bytes of that area.



DSNLEUSR stored procedure

The DSNLEUSR stored procedure is a sample stored procedure that lets you store encrypted values in the translated authorization ID (NEWAUTHID) and password fields of the SYSIBM.USERNAMES table.



You provide all the values for a SYSIBM.USERNAMES row as input to DSNLEUSR. DSNLEUSR encrypts the translated authorization ID and password values before it inserts the row into SYSIBM.USERNAMES.

Environment

DSNLEUSR has the following requirements:

- The DB2 subsystem needs to be in DB2 Version 8 new-function mode.
- DSNLEUSR runs in a WLM-established stored procedure address space.
- z/OS Integrated Cryptographic Service Facility (ICSF) must be installed, configured, and active. See *Integrated Cryptographic Service Facility System Programmer's Guide* for more information.

Authorization

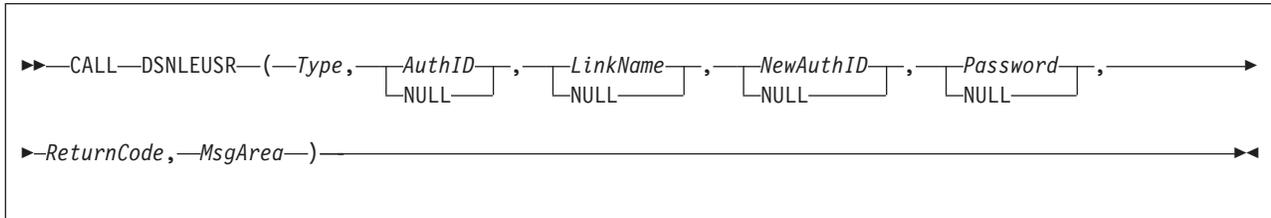
To execute the CALL DSNLEUSR statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on the package for DSNLEUSR
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have INSERT authority on SYSIBM.USERNAMES.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:



Option descriptions

Type

Specifies the value that is to be inserted into the TYPE column of SYSIBM.USERNAMES.

This is an input parameter of type CHAR(1).

AuthID

Specifies the value that is to be inserted into the AUTHID column of SYSIBM.USERNAMES.

This is an input parameter of type VARCHAR(128). If you specify a null value, DSNLEUSR does not insert a value for *AuthID*.

LinkName

Specifies the value that is to be inserted into the LINKNAME column of SYSIBM.USERNAMES.

This is an input parameter of type CHAR(8). If you specify a null value, DSNLEUSR does not insert a value for *LinkName*.

NewAuthID

Specifies the value that is to be inserted into the NEWAUTHID column of SYSIBM.USERNAMES.

This is an input parameter of type VARCHAR(119). Although the NEWAUTHID field of SYSIBM.USERNAMES is VARCHAR(128), your input value is restricted to 119 or fewer bytes.

If you specify a null value, DSNLEUSR does not insert a value for *NewAuthID*.

Password

Specifies the value that is to be inserted into the PASSWORD column of SYSIBM.USERNAMES.

This is an input parameter of type CHAR(8). Although the PASSWORD field of SYSIBM.USERNAMES is VARCHAR(24), your input value is restricted to 8 or fewer bytes.

If you specify a null value, DSNLEUSR does not insert a value for *Password*.

ReturnCode

The return code from DSNLEUSR execution. Possible values are:

- 0 DSNLEUSR executed successfully.
- 8 The request to encrypt the translated authorization ID or password failed. *MsgArea* contains the following fields:
 - An unformatted SQLCA that describes the error.

- A string that contains a DSNL045I message with the ICSF return code, the ICSF reason code, and the ICSF function that failed. The string immediately follows the SQLCA field and does not begin with a length field.

- 12 The insert operation for the SYSIBM.USERNAMES row failed. *MsgArea* contains an SQLCA that describes the error.
- 16 DSNLEUSR terminated because the DB2 subsystem is not in DB2 Version 8 new-function mode. *MsgArea* contains an SQLCA that describes the error.

This is an output parameter of type INTEGER.

MsgArea

Contains information about DSNLEUSR execution. The information that is returned is described in the *ReturnCode* description.

This is an output parameter of type VARCHAR(500).

Example

The following COBOL example shows variable declarations and an SQL CALL for inserting a row into SYSIBM.USERNAMES with an encrypted translated authorization ID and an encrypted password.

```

WORKING-STORAGE SECTION.
:
*****
* DSNLEUSR PARAMETERS *
*****
01 TYPE                                PICTURE X(1).
01 AUTHID.
   49 AUTHID-LN                        PICTURE S9(4).
   49 AUTHID-DTA                       PICTURE X(128).
01 LINKNAME                            PICTURE X(8).
01 NEWAUTHID.
   49 NEWAUTHID-LN                    PICTURE S9(4).
   49 NEWAUTHID-DTA                  PICTURE X(119).
01 PASSWORD                            PICTURE X(8).
01 RETURNCODE                          PICTURE S9(9) COMP VALUE +0.
01 MSGAREA.
   49 MSGAREA-LN                      PICTURE S9(4) COMP VALUE 500.
   49 MSGAREA-DTA                    PICTURE X(500) VALUE SPACES.
*****
* INDICATOR VARIABLES.                *
*****
01 TYPE-IND                            PICTURE S9(4) COMP-4.
01 AUTHID-IND                          PICTURE S9(4) COMP-4.
01 LINKNAME-IND                        PICTURE S9(4) COMP-4.
01 NEWAUTHID-IND                      PICTURE S9(4) COMP-4.
01 PASSWORD-IND                       PICTURE S9(4) COMP-4.
01 RETURNCODE-IND                     PICTURE S9(4) COMP-4.
01 MSGAREA-IND                        PICTURE S9(4) COMP-4.
PROCEDURE DIVISION.
:
*****
* SET VALUES FOR DSNLEUSR INPUT PARAMETERS.      *
* THE SET OF INPUT VALUES REPRESENTS A ROW THAT *
* DSNLEUSR INSERTS INTO SYSIBM.USERNAMES WITH    *
* ENCRYPTED NEWAUTHID AND PASSWORD VALUES.      *
*****
MOVE '0' TO TYPE.
MOVE 0 TO AUTHID-LN.
MOVE SPACES TO AUTHID-DTA.

```

```

MOVE 'SYEC1B ' TO LINKNAME.
MOVE 4 TO NEWAUTHID-LN.
MOVE 'MYID' TO NEWAUTHID-DTA.
MOVE 'MYPASS' TO PASSWORD.
*****
* CALL DSNLEUSR *
*****
EXEC SQL
CALL SYSPROC.DSNLEUSR
(:TYPE           :TYPE-IND,
 :AUTHID         :AUTHID-IND,
 :LINKNAME       :LINKNAME-IND,
 :NEWAUTHID      :NEWAUTHID-IND,
 :PASSWORD       :PASSWORD-IND,
 :RETURNCODE     :RETURNCODE-IND,
 :MSGAREA        :MSGAREA-IND)
END-EXEC.

```

Output

If DSNLEUSR executes successfully, it inserts a row into SYSIBM.USERNAMES with encrypted values for the NEWAUTHID and PASSWORD columns and returns 0 for the *ReturnCode* parameter value. If DSNLEUSR does not execute successfully, it returns a non-zero value for the *ReturnCode* value and additional diagnostic

information for the *MsgArea* parameter value. 

DSNAIMS stored procedure

DSNAIMS is a stored procedure that allows DB2 applications to invoke IMS transactions and commands easily, without maintaining their own connections to IMS.

 DSNAIMS uses the IMS Open Transaction Manager Access (OTMA) API to connect to IMS and execute the transactions.

Environment

DSNAIMS runs in a WLM-established stored procedures address space. DSNAIMS requires DB2 with RRSF enabled and IMS version 7 or later with OTMA Callable Interface enabled.

To use a two-phase commit process, you must have IMS Version 8 with UQ70789 or later.

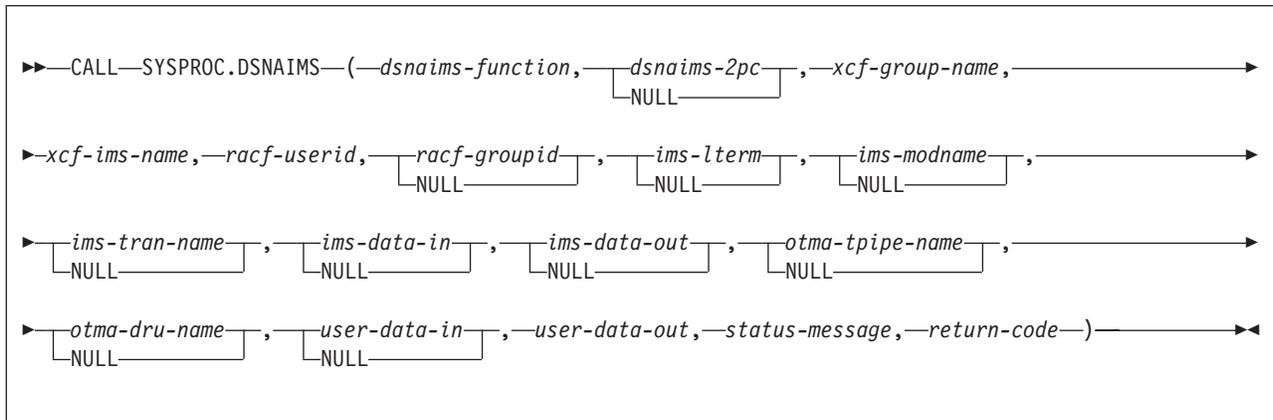
Authorization

To set up and run DSNAIMS, you must be authorized to perform the following steps:

1. Use the job DSNTIJIM to issue the CREATE PROCEDURE statement for DSNAIMS and to grant the execution of DSNAIMS to PUBLIC. DSNTIJIM is provided in the SDSNSAMP data set. You need to customize DSNTIJIM to fit the parameters of your system.
2. Ensure that OTMA C/I is initialized. See *IMS Open Transaction Manager Access Guide and Reference* for an explanation of the C/I initialization.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:



Option descriptions

dsnaims-function

A string that indicates whether the transaction is send-only, receive-only, or send-and-receive. Possible values are:

SENDRECV

Sends and receives IMS data. SENDRECV invokes an IMS transaction or command and returns the result to the caller. The transaction can be an IMS full function or a fast path. SENDRECV does not support multiple iterations of a conversational transaction

SEND Sends IMS data. SEND invokes an IMS transaction or command, but does not receive IMS data. If result data exists, it can be retrieved with the RECEIVE function. A send-only transaction cannot be an IMS fast path transaction or a conversational transaction.

RECEIVE

Receives IMS data. The data can be the result of a transaction or command initiated by the SEND function or an unsolicited output message from an IMS application. The RECEIVE function does not initiate an IMS transaction or command.

dsnaims-2pc

Specifies whether to use a two-phase commit process to perform the transaction syncpoint service. Possible values are Y or N. For N, commits and rollbacks that are issued by the IMS transaction do not affect commit and rollback processing in the DB2 application that invokes DSNAIMS.

Furthermore, IMS resources are not affected by commits and rollbacks that are issued by the calling DB2 application. If you specify Y, you must also specify SENDRECV. To use a two-phase commit process, you must set the IMS control region parameter (RRS) to Y.

This parameter is optional. The default is N.

xcf-group-name

Specifies the XCF group name that the IMS OTMA joins. You can obtain this name by viewing the GRNAME parameter in IMS PROCLIB member DFSPBxxx or by using the IMS command /DISPLAY OTMA.

xcf-ims-name

Specifies the XCF member name that IMS uses for the XCF group. If IMS is not using the XRF or RSR feature, you can obtain the XCF member name from the OTMANM parameter in IMS PROCLIB member DFSPBxxx. If IMS is using the XRF or RSR feature, you can obtain the XCF member name from the USERVAR parameter in IMS PROCLIB member DFSPBxxx.

racf-userid

Specifies the RACF user ID that is used for IMS to perform the transaction or command authorization checking. This parameter is required if DSNAIMS is running APF-authorized. If DSNAIMS is running unauthorized, this parameter is ignored and the EXTERNAL SECURITY setting for the DSNAIMS stored procedure definition determines the user ID that is used by IMS.

racf-groupid

Specifies the RACF group ID that is used for IMS to perform the transaction or command authorization checking. This field is used for stored procedures that are APF-authorized. It is ignored for other stored procedures.

ims-lterm

Specifies an IMS LTERM name that is used to override the LTERM name in the I/O program communication block of the IMS application program.

This field is used as an input and an output field:

- For SENDRECV, the value is sent to IMS on input and can be updated by IMS on output.
- For SEND, the parameter is IN only.
- For RECEIVE, the parameter is OUT only.

An empty or NULL value tells IMS to ignore the parameter.

ims-modname

Specifies the formatting map name that is used by the server to map output data streams, such as 3270 streams. Although this invocation does not have IMS MFS support, the input MODNAME can be used as the map name to define the output data stream. This name is an 8-byte message output descriptor name that is placed in the I/O program communication block. When the message is inserted, IMS places this name in the message prefix with the map name in the program communication block of the IMS application program.

For SENDRECV, the value is sent to IMS on input, and can be updated on output. For SEND, the parameter is IN only. For RECEIVE it is OUT only. IMS ignores the parameter when it is an empty or NULL value.

ims-tran-name

Specifies the name of an IMS transaction or command that is sent to IMS. If the IMS command is longer than eight characters, specify the first eight characters (including the "/" of the command). Specify the remaining characters of the command in the *ims-tran-name* parameter. If you use an empty or NULL value, you must specify the full transaction name or command in the *ims-data-in* parameter.

ims-data-in

Specifies the data that is sent to IMS. This parameter is required in each of the following cases:

- Input data is required for IMS
- No transaction name or command is passed in *ims-tran-name*
- The command is longer than eight characters

This parameter is ignored when for RECEIVE functions.

ims-data-out

Data returned after successful completion of the transaction. This parameter is required for SENDRECV and RECEIVE functions. The parameter is ignored for SEND functions.

otma-tpipe-name

Specifies an 8-byte user-defined communication session name that IMS uses for the input and output data for the transaction or the command in a SEND or a RECEIVE function. If the *otma_tpipe_name* parameter is used for a SEND function to generate an IMS output message, the same *otma_pipe_name* must be used to retrieve output data for the subsequent RECEIVE function.

otma-dru-name

Specifies the name of an IMS user-defined exit routine, OTMA destination resolution user exit routine, if it is used. This IMS exit routine can format part of the output prefix and can determine the output destination for an IMS ALT_PCB output. If an empty or null value is passed, IMS ignores this parameter.

user-data-in

This optional parameter contains any data that is to be included in the IMS message prefix, so that the data can be accessed by IMS OTMA user exit routines (DFSYIOE0 and DFSYDRU0) and can be tracked by IMS log records. IMS applications that run in dependent regions do not access this data. The specified user data is not included in the output message prefix. You can use this parameter to store input and output correlator tokens or other information. This parameter is ignored for RECEIIVE functions.

user-data-out

On output, this field contains the *user-data-in* in the IMS output prefix. IMS user exit routines (DFSYIOE0 and DFSYDRU0) can also create *user-data-out* for SENDRECV and RECEIVE functions. The parameter is not updated for SEND functions.

status-message

Indicates any error message that is returned from the transaction or command, OTMA, RRS, or DSNAIMS.

return-code

Indicates the return code that is returned for the transaction or command, OTMA, RRS, or DSNAIMS.

Examples

The following examples show how to call DSNAIMS.

Example 1: Sample parameters for executing an IMS command:

```
CALL SYSPROC.DSNAIMS("SENDRECV", "N", "IMS7GRP", "IMS7TMEM",
                    "IMSCCLNM", "", "", "", "", "", "",
                    "/LOG Hello World.", ims_data_out, "", "", "",
                    user_out, error_message, rc)
```

Example 2: Sample parameters for executing an IMS IVTNO transaction:

```
CALL SYSPROC.DSNAIMS("SENDRECV", "N", "IMS7GRP", "IMS7TMEM",
                    "IMSCCLNM", "", "", "", "", "", "",
                    "IVTNO DISPLAY LAST1", "", ims_data_out
                    "", "", "", user_out, error_message, rc)
```

Example 3: Sample parameters for send-only IMS transaction:

```
CALL SYSPROC.DSNAIMS("SEND", "N", "IMS7GRP", "IMS7TMEM",
                    "IMSCCLNM", "", "", "", "", "",
                    "IVTNO    DISPLAY LAST1    ", ims_data_out,
                    "DSNAPIPE", "", "", user_out, error_message, rc)
```

Example 4: Sample parameters for receive-only IMS transaction:

```
CALL SYSPROC.DSNAIMS("RECEIVE", "N", "IMS7GRP", "IMS7TMEM",
                    "IMSCCLNM", "", "", "", "", "",
                    "IVTNO    DISPLAY LAST1    ", ims_data_out,
                    "DSNAPIPE", "", "", user_out, error_message, rc)
```

Connecting to multiple IMS subsystems with DSNAIMS

By default DSNAIMS connects to only one IMS subsystem at a time. The first request to DSNAIMS determines to which IMS subsystem the stored procedure connects. DSNAIMS attempts to reconnect to IMS only in the following cases:

- IMS is restarted and the saved connection is no longer valid
- WLM loads another DSNAIMS task

To connect to multiple IMS subsystems simultaneously, perform the following steps:

1. Make a copy of the DB2-supplied job DSNTIJIM and customize it to your environment.
2. Change the procedure name from SYSPROCC.DSNAIMS to another name, such as DSNAIMSB.
3. Do not change the EXTERNAL NAME option. Leave it as DSNAIMS.
4. Run the new job to create a second instance of the stored procedure.
5. To ensure that you connect to the intended IMS target, consistently use the XFC group and member names that you associate with each stored procedure instance. For example:

```
CALL SYSPROC.DSNAIMS("SENDRECV", "N", "IMS7GRP", "IMS7TMEM", ...)
CALL SYSPROC.DSNAIMSB("SENDRECV", "N", "IMS8GRP", "IMS8TMEM", ...)
```

GUPI

DSNAIMS2 stored procedure

DSNAIMS2 is a stored procedure that allows DB2 applications to invoke IMS transactions and commands easily, without maintaining their own connections to IMS. DSNAIMS2 includes multi-segment input support for IMS transactions.

GUPI

DSNAIMS2 uses the IMS Open Transaction Manager Access (OTMA) API to connect to IMS and execute the transactions.

When you define the DSNAIMS2 stored procedure to your DB2 subsystem, you can use the name DSNAIMS in your application if you prefer. Customize DSNTIJI2 to define the stored procedure to your DB2 subsystem as DSNAIMS; however, the EXTERNAL NAME option must still be DSNAIMS2.

Environment

DSNAIMS2 runs in a WLM-established stored procedures address space. DSNAIMS2 requires DB2 with RRSF enabled and IMS version 7 or later with OTMA Callable Interface enabled.

To use a two-phase commit process, you must have IMS Version 8 with UQ70789 or later.

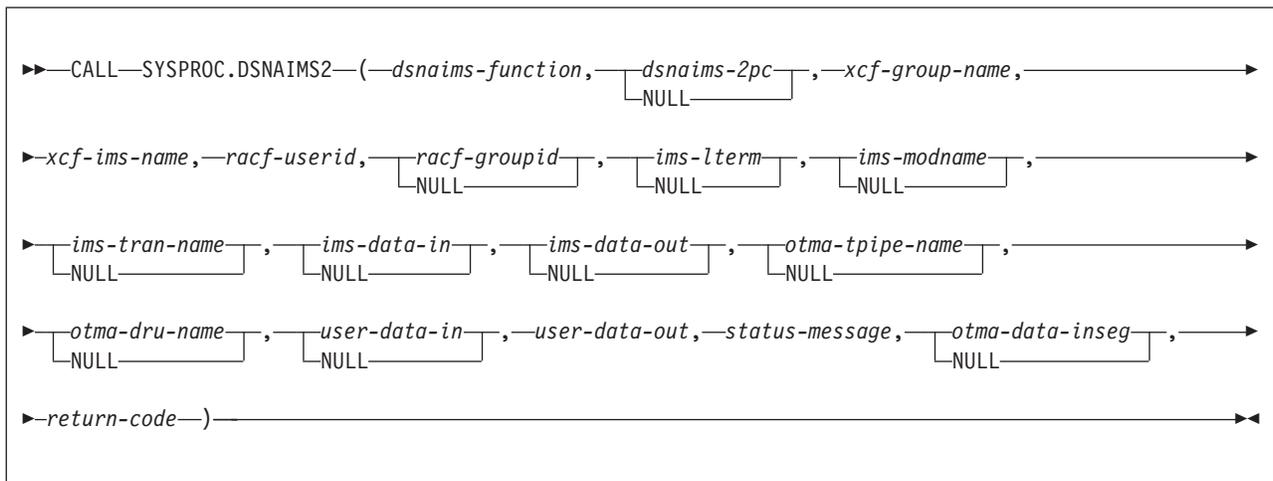
Authorization

To set up and run DSNAIMS2, you must be authorized to perform the following steps:

1. Use the job DSNTIJI2 to issue the CREATE PROCEDURE statement for DSNAIMS2 and to grant the execution of DSNAIMS2 to PUBLIC. DSNTIJI2 is provided in the SDSNSAMP data set. You need to customize DSNTIJI2 to fit the parameters of your system.
2. Ensure that OTMA C/I is initialized. See *IMS Open Transaction Manager Access Guide and Reference* for an explanation of the C/I initialization.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:



Option descriptions

dsnaims-function

A string that indicates whether the transaction is send-only, receive-only, or send-and-receive. Possible values are:

SENDRECV

Sends and receives IMS data. SENDRECV invokes an IMS transaction or command and returns the result to the caller. The transaction can be an IMS full function or a fast path. SENDRECV does not support multiple iterations of a conversational transaction

SEND

Sends IMS data. SEND invokes an IMS transaction or command, but does not receive IMS data. If result data exists, it can be retrieved with

the RECEIVE function. A send-only transaction cannot be an IMS fast path transaction or a conversations transaction.

RECEIVE

Receives IMS data. The data can be the result of a transaction or command initiated by the SEND function or an unsolicited output message from an IMS application. The RECEIVE function does not initiate an IMS transaction or command.

dsnaims-2pc

Specifies whether to use a two-phase commit process to perform the transaction syncpoint service. Possible values are Y or N. For N, commits and rollbacks that are issued by the IMS transaction do not affect commit and rollback processing in the DB2 application that invokes DSNAIMS2. Furthermore, IMS resources are not affected by commits and rollbacks that are issued by the calling DB2 application. If you specify Y, you must also specify SENDRECV. To use a two-phase commit process, you must set the IMS control region parameter (RRS) to Y.

This parameter is optional. The default is N.

xcf-group-name

Specifies the XCF group name that the IMS OTMA joins. You can obtain this name by viewing the GRNAME parameter in IMS PROCLIB member DFSPBxxx or by using the IMS command /DISPLAY OTMA.

xcf-ims-name

Specifies the XCF member name that IMS uses for the XCF group. If IMS is not using the XRF or RSR feature, you can obtain the XCF member name from the OTMANM parameter in IMS PROCLIB member DFSPBxxx. If IMS is using the XRF or RSR feature, you can obtain the XCF member name from the USERVAR parameter in IMS PROCLIB member DFSPBxxx.

racf-userid

Specifies the RACF user ID that is used for IMS to perform the transaction or command authorization checking. This parameter is required if DSNAIMS2 is running APF-authorized. If DSNAIMS2 is running unauthorized, this parameter is ignored and the EXTERNAL SECURITY setting for the DSNAIMS2 stored procedure definition determines the user ID that is used by IMS.

racf-groupid

Specifies the RACF group ID that is used for IMS to perform the transaction or command authorization checking. This field is used for stored procedures that are APF-authorized. It is ignored for other stored procedures.

ims-lterm

Specifies an IMS LTERM name that is used to override the LTERM name in the I/O program communication block of the IMS application program.

This field is used as an input and an output field:

- For SENDRECV, the value is sent to IMS on input and can be updated by IMS on output.
- For SEND, the parameter is IN only.
- For RECEIVE, the parameter is OUT only.

An empty or NULL value tells IMS to ignore the parameter.

ims-modname

Specifies the formatting map name that is used by the server to map output data streams, such as 3270 streams. Although this invocation does not have

IMS MFS support, the input MODNAME can be used as the map name to define the output data stream. This name is an 8-byte message output descriptor name that is placed in the I/O program communication block. When the message is inserted, IMS places this name in the message prefix with the map name in the program communication block of the IMS application program.

For SENDRECV, the value is sent to IMS on input, and can be updated on output. For SEND, the parameter is IN only. For RECEIVE it is OUT only. IMS ignores the parameter when it is an empty or NULL value.

ims-tran-name

Specifies the name of an IMS transaction or command that is sent to IMS. If the IMS command is longer than eight characters, specify the first eight characters (including the "/" of the command). Specify the remaining characters of the command in the *ims-tran-name* parameter. If you use an empty or NULL value, you must specify the full transaction name or command in the *ims-data-in* parameter.

ims-data-in

Specifies the data that is sent to IMS. This parameter is required in each of the following cases:

- Input data is required for IMS
- No transaction name or command is passed in *ims-tran-name*
- The command is longer than eight characters

This parameter is ignored when for RECEIVE functions.

ims-data-out

Data returned after successful completion of the transaction. This parameter is required for SENDRECV and RECEIVE functions. The parameter is ignored for SEND functions.

otma-tpipe-name

Specifies an 8-byte user-defined communication session name that IMS uses for the input and output data for the transaction or the command in a SEND or a RECEIVE function. If the *otma-tpipe-name* parameter is used for a SEND function to generate an IMS output message, the same *otma-tpipe-name* must be used to retrieve output data for the subsequent RECEIVE function.

otma-dru-name

Specifies the name of an IMS user-defined exit routine, OTMA destination resolution user exit routine, if it is used. This IMS exit routine can format part of the output prefix and can determine the output destination for an IMS ALT_PCB output. If an empty or null value is passed, IMS ignores this parameter.

user-data-in

This optional parameter contains any data that is to be included in the IMS message prefix, so that the data can be accessed by IMS OTMA user exit routines (DFSYIOE0 and DFSYDRU0) and can be tracked by IMS log records. IMS applications that run in dependent regions do not access this data. The specified user data is not included in the output message prefix. You can use this parameter to store input and output correlator tokens or other information. This parameter is ignored for RECEIVE functions.

user-data-out

On output, this field contains the *user-data-in* in the IMS output prefix. IMS

user exit routines (DFSYIOE0 and DFSYDRU0) can also create *user-data-out* for SENDRECV and RECEIVE functions. The parameter is not updated for SEND functions.

status-message

Indicates any error message that is returned from the transaction or command, OTMA, RRS, or DSNAIMS2.

otma-data-inseg

Specifies the number of segments followed by the lengths of the segments to be sent to IMS. All values should be separated by semicolons. This field is required to send multi-segment input to IMS. For single-segment transactions and commands, set the field to NULL, "0" or "0;".

return-code

Indicates the return code that is returned for the transaction or command, OTMA, RRS, or DSNAIMS2.

Examples

The following examples show how to call DSNAIMS2.

Example 1: Sample parameters for executing a multi-segment IMS transaction:

```
CALL SYSPROC.DSNAIMS2("SEND","N","IMS7GRP","IMS7TMEM",
    "IMSCLNM","","","","","","",
    "PART 1ST SEGMENT FROM CI 2ND SEGMENT FROM CI ",
    ims_data_out,"","","",user_out, error_message,
    "2;25;20",rc)
```

Example 2: Sample parameters for executing a single-segment IMS IVTNO transaction:

```
CALL SYSPROC.DSNAIMS2("SEND","N","IMS7GRP","IMS7TMEM",
    "IMSCLNM","","","","","IVTNO",
    "DISPLAY LAST1",ims_data_out,"","","",
    user_out, error_message,NULL,rc)
```

Connecting to multiple IMS subsystems with DSNAIMS2

By default DSNAIMS2 connects to only one IMS subsystem at a time. The first request to DSNAIMS2 determines to which IMS subsystem the stored procedure connects. DSNAIMS2 attempts to reconnect to IMS only in the following cases:

- IMS is restarted and the saved connection is no longer valid
- WLM loads another DSNAIMS2 task

To connect to multiple IMS subsystems simultaneously, perform the following steps:

1. Make a copy of the DB2-supplied job DSNTIJI2 and customize it to your environment.
2. Change the procedure name from SYSPROCC.DSNAIMS2 to another name, such as DSNAIMS2B.
3. Do not change the EXTERNAL NAME option. Leave it as DSNAIMS2.
4. Change the name of the stored procedure in the grant statement in job DSNTIJI2.
5. Run the new job to create a second instance of the stored procedure.

6. To ensure that you connect to the intended IMS target, consistently use the XFC group and member names that you associate with each stored procedure instance. For example:

```
CALL SYSPROC.DSNAIMS2("SENDRECV", "N", "IMS7GRP", "IMS7TMEM", ...)
CALL SYSPROC.DSNAIMS2B("SENDRECV", "N", "IMS8GRP", "IMS8TMEM", ...)
```

◀ GUPI

ADMIN_COMMAND_DB2 stored procedure

The SYSPROC.ADMIN_COMMAND_DB2 stored procedure executes one or more DB2 commands on a connected DB2 subsystem or on a DB2 data sharing group member and returns the command output messages.

◀ GUPI

Environment

ADMIN_COMMAND_DB2 must run in a WLM-established stored procedure address space.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNADMCD
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

To execute the DB2 command, you must use a privilege set that includes the authorization to execute the DB2 command, as described in the *DB2 Command Reference*.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
▶▶ CALL SYSPROC.ADMIN_COMMAND_DB2 ( ( DB2-command, command-length, parse-type,
DB2-member, commands-executed, -IFI-return-code, -IFI-reason-code, excess-bytes,
NULL,
group-IFI-reason-code, group-excess-bytes, return-code, message- ) )
```

Option descriptions

DB2-command

Specifies any DB2 command such as -DISPLAY THREAD(*), or multiple DB2

commands. With multiple DB2 commands, use '\0' to delimit the commands. The DB2 command is executed using the authorization ID of the user who invoked the stored procedure.

This is an input parameter of type VARCHAR(32704) and cannot be null.

command-length

Specifies the length of the DB2 command or commands. When multiple DB2 commands are specified in *DB2-command*, *command-length* is the sum of all of those commands, including the '\0' command delimiters.

This is an input parameter of type INTEGER and cannot be null.

parse-type

Identifies the type of output message parsing requested.

If you specify a parse type, ADMIN_COMMAND_DB2 parses the command output messages and provides the formatted result in a global temporary table. Possible values are:

- BP** Parse “-DISPLAY BUFFERPOOL” command output messages.
- DB** Parse “-DISPLAY DATABASE” command output messages and return database information.
- TS** Parse “-DISPLAY DATABASE(...) SPACENAM(...)” command output messages and return table spaces information.
- IX** Parse “-DISPLAY DATABASE(...) SPACENAM(...)” command output messages and return index spaces information.
- THD** Parse “-DISPLAY THREAD” command output messages.
- UT** Parse “-DISPLAY UTILITY” command output messages.
- GRP** Parse “-DISPLAY GROUP” command output messages.
- DDF** Parse “-DISPLAY DDF” command output messages.

Any other value

Do not parse any command output messages.

This is an input parameter of type VARCHAR(3) and cannot be null.

DB2-member

Specifies the name of a single data sharing group member on which an IFI request is to be executed

This is an input parameter of type VARCHAR(8).

commands-executed

Provides the number of commands that were executed

This is an output parameter of type INTEGER.

IFI-return-code

Provides the IFI return code

This is an output parameter of type INTEGER.

IFI-reason-code

Provides the IFI reason code

This is an output parameter of type INTEGER.

excess-bytes

Indicates the number of bytes that did not fit in the return area

This is an output parameter of type INTEGER.

group-IFI-reason-code

Provides the reason code for the situation in which an IFI call requests data from members of a data sharing group, and not all the data is returned from group members.

This is an output parameter of type INTEGER.

group-excess-bytes

Indicates the total length of data that was returned from other data sharing group members and did not fit in the return area

This is an output parameter of type INTEGER.

return-code

Provides the return code from the stored procedure. Possible values are:

- 0 The stored procedure did not encounter an SQL error during processing. Check the *IFI-return-code* value to determine whether the DB2 command issued using the instrumentation facility interface (IFI) was successful or not.
- 12 The stored procedure encountered an SQL error during processing. The *message* output parameter contains messages describing the SQL error.

This is an output parameter of type INTEGER.

message

Contains messages describing the SQL error encountered by the stored procedure. If no SQL error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_COMMAND_DB2:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
    /***** DB2 Host Variables *****/
    EXEC SQL BEGIN DECLARE SECTION;

    /* SYSPROC.ADMIN_COMMAND_DB2 parameters */
    char    command[32705];          /* DB2 command */
    short int ind_command;          /* Indicator variable */
    long int len_command;          /* DB2 command length */
    short int ind_len_command;     /* Indicator variable */
    char    parsetype[4];          /* Parse type required */
    short int ind_parsetype;       /* Indicator variable */
    char    mbrname[9];           /* DB2 data sharing group
                                   /* member name */

    short int ind_mbrname;        /* Indicator variable */
    long int excommands;          /* Number of commands exec. */
    short int ind_excommands;     /* Indicator variable */
}
```

```

long int    retifca;          /* IFI return code          */
short int   ind_retifca;     /* Indicator variable       */
long int    resifca;        /* IFI reason code         */
short int   ind_resifca;    /* Indicator variable       */
long int    xsbytes;        /* Excessive bytes         */
short int   ind_xsbytes;    /* Indicator variable       */
long int    gresifca;       /* IFI group reason code   */
short int   ind_gresifca;   /* Indicator variable       */
long int    gxsbytes;       /* Group excessive bytes   */
short int   ind_gxsbytes;   /* Indicator variable       */
long int    retcd;          /* Return code              */
short int   ind_retcd;     /* Indicator variable       */
char        errmsg[1332];   /* Error message            */
short int   ind_errmsg;     /* Indicator variable       */

/* Result Set Locators                                         */
volatile SQL TYPE IS RESULT_SET_LOCATOR * rs_loc1,
                                             rs_loc2;

/* First result set row                                         */
long int    rownum;         /* Sequence number of the   */
                                             /* table row                 */
char        text[81];      /* Command output           */

/* Second result set row                                         */
long int    ddfrownum;      /* DDF table sequence      */
char        ddfstat[7];    /* DDF status               */
char        ddfloc[19];    /* DDF location            */
char        ddf lunm[18];  /* DDF luname              */
char        ddfgenlu[18];  /* DDF generic lu          */
char        ddfv4ipaddr[18]; /* DDF IPv4 address        */
char        ddfv6ipaddr[40]; /* DDF IPv6 address        */
long int    ddftcpport;    /* DDF tcpport             */
long int    ddfresport;    /* DDF resport             */
char        ddfsqldom[46]; /* DDF sql domain          */
char        ddfrsyncdom[46]; /* DDF resync domain       */
EXEC SQL END DECLARE SECTION;

/*****
/* Assign values to input parameters to execute the DB2          */
/* command "-DISPLAY DDF"                                       */
/* Set the indicator variables to 0 for non-null input parameters */
/* Set the indicator variables to -1 for null input parameters   */
*****/
strcpy(command, "-DISPLAY DDF");
ind_command = 0;
lencommand = strlen(command);
ind_lencommand = 0;
strcpy(parsetype, "DDF");
ind_parsetype = 0;
ind_mbrname = -1;

/*****
/* Call stored procedure SYSPROC.ADMIN_COMMAND_DB2              */
*****/
EXEC SQL CALL SYSPROC.ADMIN_COMMAND_DB2
           (:command      :ind_command,
           :lencommand   :ind_lencommand,
           :parsetype    :ind_parsetype,
           :mbrname      :ind_mbrname,
           :excommands   :ind_excommands,
           :retifca      :ind_retifca,
           :resifca      :ind_resifca,
           :xsbytes      :ind_xsbytes,
           :gresifca     :ind_gresifca,
           :gxsbytes     :ind_gxsbytes,
           :retcd        :ind_retcd,

```

```

:errmsg      :ind_errmsg);

/*****
/* Retrieve result set(s) when the SQLCODE from the call is +466, */
/* which indicates that result sets were returned                */
/*****
if (SQLCODE == +466)      /* Result sets were returned */
{
  /* ESTABLISH A LINK BETWEEN EACH RESULT SET AND ITS LOCATOR    */
  EXEC SQL ASSOCIATE LOCATORS (:rs_loc1, :rs_loc2)
        WITH PROCEDURE SYSPROC.ADMIN_COMMAND_DB2;

  /* ASSOCIATE A CURSOR WITH EACH RESULT SET                      */
  EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :rs_loc1;
  EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :rs_loc2;

  /* PERFORM FETCHES USING C1 TO RETRIEVE ALL ROWS FROM THE     */
  /* FIRST RESULT SET                                           */
  EXEC SQL FETCH C1 INTO :rownum, :text;

  while(SQLCODE == 0)
  {
    EXEC SQL FETCH C1 INTO :rownum, :text;
  }

  /* PERFORM FETCHES USING C2 TO RETRIEVE THE -DISPLAY DDF      */
  /* PARSED OUTPUT FROM THE SECOND RESULT SET                   */
  EXEC SQL FETCH C2 INTO :ddfrownum, :ddfstat, :ddfloc,
                        :ddf1unm, :ddfgenlu,
                        :ddf4ipaddr, :ddf6ipaddr,
                        :ddftcport, :ddfresport,
                        :ddfsqldom, :ddfrsyncdom;
}

return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 841:

- *commands-executed*
- *IFI-return-code*
- *IFI-reason-code*
- *excess-bytes*
- *group-IFI-reason-code*
- *group-excess-bytes*
- *return-code*
- *message*

In addition to the preceding output, the stored procedure returns two result sets.

The first result set is returned in the created global temporary table SYSIBM.DB2_CMD_OUTPUT and contains the DB2 command output messages that were not parsed.

The following table shows the format of the first result set:

Table 185. Result set row for first ADMIN_COMMAND_DB2 result set

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
TEXT	CHAR(80)	DB2 command output message line

The format of the second result set varies, depending on the DB2 command issued and the *parse-type* value.

- Result set row for second ADMIN_COMMAND_DB2 result set (*parse-type* = "BP")
- Result set row for second ADMIN_COMMAND_DB2 result set (*parse-type* = "THD")
- Result set row for second ADMIN_COMMAND_DB2 result set (*parse-type* = "UT")
- Result set row for second ADMIN_COMMAND_DB2 result set (*parse-type* = "DB" or "TS" or "IX")
- Result set row for second ADMIN_COMMAND_DB2 result set (*parse-type* = "GRP")
- Result set row for second ADMIN_COMMAND_DB2 result set (*parse-type* = "DDF")

The following table shows the format of the result set returned in the created global temporary table SYSIBM.BUFFERPOOL_STATUS when *parse-type* = "BP":

Table 186. Result set row for second ADMIN_COMMAND_DB2 result set (parse-type = "BP")

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
BPNAME	CHAR(6)	Buffer pool name
VPSIZE	INTEGER	Buffer pool size
VPSEQT	INTEGER	Sequential steal threshold for the buffer pool
VPPSEQT	INTEGER	Parallel sequential threshold for the buffer pool
VPXPSEQT	INTEGER	Assisting parallel sequential threshold for the buffer pool
DWQT	INTEGER	Deferred write threshold for the buffer pool
PCT_VDWQT	INTEGER	Vertical deferred write threshold for the buffer pool (as a percentage of virtual buffer pool size)
ABS_VDWQT	INTEGER	Vertical deferred write threshold for the buffer pool (as absolute number of buffers)
PGSTEAL	CHAR(4)	Page-stealing algorithm that DB2 uses for the buffer pool

Table 186. Result set row for second ADMIN_COMMAND_DB2 result set (parse-type = "BP") (continued)

Column name	Data type	Contents
ID	INTEGER	Buffer pool internal identifier
USE_COUNT	INTEGER	Number of open table spaces or index spaces that reference this buffer pool
PGFIX	CHAR(3)	Specifies whether the buffer pool should be fixed in real storage when it is used

The following table shows the format of the result set returned in the created global temporary table SYSIBM.DB2_THREAD_STATUS when parse-type = "THD":

Table 187. Result set row for second ADMIN_COMMAND_DB2 result set (parse-type = "THD")

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
TYPE	INTEGER	Thread type: 0 Unknown 1 Active 2 Inactive 3 Indoubt 4 Postponed
NAME	CHAR(8)	Connection name used to establish the thread
STATUS	CHAR(11)	Status of the conversation or socket
ACTIVE	CHAR(1)	Indicates whether a thread is active or not. An asterisk means that the thread is active within DB2.
REQ	CHAR(5)	Current number of DB2 requests on the thread
ID	CHAR(12)	Recovery correlation ID associated with the thread
AUTHID	CHAR(8)	Authorization ID associated with the thread
PLAN	CHAR(8)	Plan name associated with the thread
ASID	CHAR(4)	Address space identifier
TOKEN	CHAR(6)	Unique thread identifier
COORDINATOR	CHAR(46)	Name of the two-phase commit coordinator

Table 187. Result set row for second ADMIN_COMMAND_DB2 result set (parse-type = "THD") (continued)

Column name	Data type	Contents
RESET	CHAR(5)	Indicates whether or not the thread needs to be reset to purge info from the indoubt thread report
URID	CHAR(12)	Unit of recovery identifier
LUWID	CHAR(35)	Logical unit of work ID of the thread
WORKSTATION	CHAR(18)	Client workstation name
USERID	CHAR(16)	Client user ID
APPLICATION	CHAR(32)	Client application name
ACCOUNTING	CHAR(247)	Client accounting information
LOCATION	VARCHAR(4050)	Location name of the remote system
DETAIL	VARCHAR(4050)	Additional thread information

The following table shows the format of the result set returned in the created global temporary table SYSIBM.UTILITY_JOB_STATUS when parse-type = "UT":

Table 188. Result set row for second ADMIN_COMMAND_DB2 result set (parse-type = "UT")

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
CSECT	CHAR(8)	Name of the command program CSECT that issued the message
USER	CHAR(8)	User ID of the person running the utility
MEMBER	CHAR(8)	Utility job is running on this member
UTILID	CHAR(16)	Utility job identifier
STATEMENT	INTEGER	Utility statement number
UTILITY	CHAR(20)	Utility name
PHASE	CHAR(20)	Utility restart from the beginning of this phase
COUNT	INTEGER	Number of pages or records processed in a utility phase
STATUS	CHAR(18)	Utility status
DETAIL	VARCHAR(4050)	Additional utility information
NUM_OBJ	INTEGER	Total number of objects in the list of objects the utility is processing
LAST_OBJ	INTEGER	Last object that started

The following table shows the format of the result set returned in the created global temporary table SYSIBM.DB_STATUS when *parse-type* = "DB" or "TS" or "IX":

Table 189. Result set row for second ADMIN_COMMAND_DB2 result set (parse-type = "DB" or "TS" or "IX")

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
DBNAME	CHAR(8)	Name of the database
SPACENAM	CHAR(8)	Name of the table space or index
TYPE	CHAR(2)	Status type: DB Database TS Table space IX Index
PART	SMALLINT	Individual partition or range of partition
STATUS	CHAR(18)	Status of the database, table space or index

The following table shows the format of the result set returned in the created global temporary table SYSIBM.DATA_SHARING_GROUP when *parse-type* = "GRP":

Table 190. Result set row for second ADMIN_COMMAND_DB2 result set (parse-type = "GRP")

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
DB2_MEMBER	CHAR(8)	Name of the DB2 group member
ID	INTEGER	ID of the DB2 group member
SUBSYS	CHAR(4)	Subsystem name of the DB2 group member
CMDPREF	CHAR(8)	Command prefix for the DB2 group member
STATUS	CHAR(8)	Status of the DB2 group member
DB2_LVL	CHAR(3)	DB2 version, release and modification level
SYSTEM_NAME	CHAR(8)	Name of the z/OS system where the member is running, or was last running in cases when the member status is QUIESCED or FAILED
IRLM_SUBSYS	CHAR(4)	Name of the IRLM subsystem to which the DB2 member is connected

Table 190. Result set row for second ADMIN_COMMAND_DB2 result set (parse-type = "GRP") (continued)

Column name	Data type	Contents
IRLMPROC	CHAR(8)	Procedure name of the connected IRLM

The following table shows the format of the result set returned in the created global temporary table SYSIBM.DDF_CONFIG when *parse-type* = "DDF":

Table 191. Result set row for second ADMIN_COMMAND_DB2 result set (parse-type = "DDF")

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
STATUS	CHAR(6)	Operational status of DDF
LOCATION	CHAR(18)	Location name of DDF
LUNAME	CHAR(17)	Fully qualified LUNAME of DDF
GENERICLU	CHAR(17)	Fully qualified generic LUNAME of DDF
IPV4ADDR	CHAR(17)	IPV4 address of DDF
IPV6ADDR	CHAR(39)	IPV6 address of DDF
TCPPOINT	INTEGER	SQL listener port used by DDF
RESPORT	INTEGER	Resync listener port used by DDF
SQL_DOMAIN	CHAR(45)	SQL domain used by DDF
RSYNC_DOMAIN	CHAR(45)	Resync domain used by DDF



ADMIN_COMMAND_DSN stored procedure

The SYSPROC.ADMIN_COMMAND_DSN stored procedure executes a BIND, REBIND, or FREE DSN subcommand and returns the output from the DSN subcommand execution.

Environment

ADMIN_COMMAND_DSN runs in a WLM-established stored procedures address space. TCB=1 is also required.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on the ADMIN_COMMAND_DSN stored procedure
- Ownership of the stored procedure
- SYSADM authority

To execute the DSN subcommand, you must use a privilege set that includes the authorization to execute the DSN subcommand as described in the *DB2 Command Reference*.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
▶▶—CALL—SYSPROC.ADMIN_COMMAND_DSN—(—DSN-subcommand,—message—)————▶▶
```

Option descriptions

DSN-subcommand

Specifies the DSN subcommand to be executed. If the DSN subcommand passed to the stored procedure is not BIND, REBIND, or FREE, an error message is returned. The DSN subcommand is performed using the authorization ID of the user who invoked the stored procedure.

This is an input parameter of type VARCHAR(32704) and cannot be null.

message

Contains messages if an error occurs during stored procedure execution.

A blank *message* does not mean that the DSN subcommand completed successfully. The calling application must read the result set to determine if the DSN subcommand was successful or not.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_COMMAND_DSN:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
    /***** DB2 Host Variables *****/
    EXEC SQL BEGIN DECLARE SECTION;

    /* SYSPROC.ADMIN_COMMAND_DSN parameters */
    char subcmd[32705]; /* BIND, REBIND or FREE DSN */
    /* subcommand */
    char errmsg[1332]; /* Error message */

    /* Result set locators */
    volatile SQL TYPE IS RESULT_SET_LOCATOR *rs_loc1;

    /* Result set row */
    long int rownum; /* Sequence number of the */
    /* table row */
    char text[256]; /* DSN subcommand output row */
}
```

```

EXEC SQL END DECLARE SECTION;

/*****
/* Set input parameter to execute a REBIND PLAN DSN subcommand */
/*****
strcpy(subcmd, "REBIND PLAN (DSNACCOB) FLAG(W)");

/*****
/* Call stored procedure SYSPROC.ADMIN_COMMAND_DSN */
/*****
EXEC SQL CALL SYSPROC.ADMIN_COMMAND_DSN (:subcmd, :errmsg);

/*****
/* Retrieve result set when the SQLCODE from the call is +446, */
/* which indicates that result sets were returned */
/*****
if (SQLCODE == +446) /* Result sets were returned */
{
/* Establish a link between the result set and its locator */
EXEC SQL ASSOCIATE LOCATORS (:rs_loc1)
WITH PROCEDURE SYSPROC.ADMIN_COMMAND_DSN;

/* Associate a cursor with the result set */
EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :rs_loc1;

/* Perform fetches using C1 to retrieve all rows from the */
/* result set */
EXEC SQL FETCH C1 INTO :rownum, :text;
while(SQLCODE==0)
{
EXEC SQL FETCH C1 INTO :rownum, :text;
}
}

return;
}

```

Output

This stored procedure returns an error message, *message*, if an error occurs.

The stored procedure returns one result set that contains the DSN sub-command output messages.

The following table shows the format of the result set returned in the created global temporary table SYSIBM.DSN_SUBCMD_OUTPUT:

Table 192. Result set row for ADMIN_COMMAND_DSN result set

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
TEXT	VARCHAR(255)	DSN subcommand output message line

ADMIN_COMMAND_UNIX stored procedure

The SYSPROC.ADMIN_COMMAND_UNIX stored procedure executes a z/OS UNIX System Services command and returns the output.

Environment

ADMIN_COMMAND_UNIX runs in a WLM-established stored procedures address space.

The load module for ADMIN_COMMAND_UNIX, DSNADMCU, must be program controlled if the BPX.DAEMON.HFSCCTL FACILITY class profile has not been set up. For information on how to define DSNADMCU to program control, see installation job DSNTIJRA.

Authorization

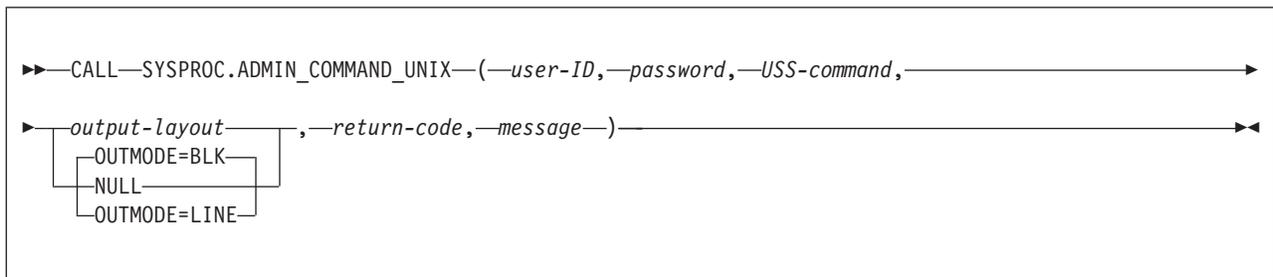
To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNADMCU
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The user specified in the *user-ID* input parameter of the SQL CALL statement must have the appropriate authority to execute the z/OS UNIX System Services command.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:



Option descriptions

user-ID

Specifies the user ID under which the z/OS UNIX System Services command is issued.

This is an input parameter of type VARCHAR(128) and cannot be null.

password

Specifies the password associated with the input parameter *user-ID*.

The value of *password* is passed to the stored procedure as part of payload, and is not encrypted. It is not stored in dynamic cache when parameter markers are used.

This is an input parameter of type VARCHAR(24) and cannot be null.

USS-command

Specifies the z/OS UNIX System Services command to be executed.

This is an input parameter of type VARCHAR(32704) and cannot be null.

output-layout

Specifies how the output from the z/OS UNIX System Services command is returned. The output from the z/OS UNIX System Services command is a multi-line message. Possible values are:

OUTMODE=LINE

Each line is returned as a row in the result set.

OUTMODE=BLK

The lines are blocked into 32677 blocks and each block is returned as a row in the result set.

If a null or empty string is provided, then the default option OUTMODE=BLK is used.

This is an input parameter of type VARCHAR(1024).

return-code

Provides the return code from the stored procedure. Possible values are:

- 0** The call completed successfully.
- 12** The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_COMMAND_UNIX:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
  /***** DB2 Host Variables *****/
  EXEC SQL BEGIN DECLARE SECTION;

  /* SYSPROC.ADMIN_COMMAND_UNIX parameters */
  char      userid[129];          /* User ID */
  short int ind_userid;          /* Indicator variable */
  char      password[25];        /* Password */
  short int ind_password;        /* Indicator variable */
  char      command[32705];      /* USS command */
  short int ind_command;         /* Indicator variable */
  char      layout[1025];        /* Command output layout */
  short int ind_layout;          /* Indicator variable */
```

```

long int   retcd;                /* Return code          */
short int  ind_retcd;           /* Indicator variable   */
char       errmsg[1332];       /* Error message        */
short int  ind_errmsg;         /* Indicator variable   */

/* Result set locators */
volatile SQL TYPE IS RESULT_SET_LOCATOR *rs_loc1;

/* Result set row */
long int   rownum;              /* Sequence number of the
/* table row */
char       text[32678];         /* A row in USS command output*/
EXEC SQL END DECLARE SECTION;

/*****
/* Assign values to input parameters to execute a USS command */
/* Set the indicator variables to 0 for non-null input parameters */
/* Set the indicator variables to -1 for null input parameters */
*****/
strcpy(userid, "USRT001");
ind_userid = 0;
strcpy(password, "NICETEST");
ind_password = 0;
strcpy(command, "ls");
ind_command = 0;
ind_layout = -1;

/*****
/* Call stored procedure SYSPROC.ADMIN_COMMAND_UNIX */
*****/
EXEC SQL CALL SYSPROC.ADMIN_COMMAND_UNIX
           (:userid   :ind_userid,
           :password  :ind_password,
           :command   :ind_command,
           :layout    :ind_layout,
           :retcd     :ind_retcd,
           :errmsg    :ind_errmsg);

/*****
/* Retrieve result set when the SQLCODE from the call is +446,
/* which indicates that result sets were returned */
*****/
if (SQLCODE == +446) /* Result sets were returned */
{
  /* Establish a link between the result set and its locator */
  EXEC SQL ASSOCIATE LOCATORS (:rs_loc1)
        WITH PROCEDURE SYSPROC.ADMIN_COMMAND_UNIX;

  /* Associate a cursor with the result set */
  EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :rs_loc1;

  /* Perform fetches using C1 to retrieve all rows from the
  /* result set */
  EXEC SQL FETCH C1 INTO :rownum, :text;
  while(SQLCODE==0)
  {
    EXEC SQL FETCH C1 INTO :rownum, :text;
  }
}

return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 853:

- *return-code*
- *message*

In addition to the preceding output, the stored procedure returns one result set that contains the z/OS UNIX System Services command output messages.

The following table shows the format of the result set returned in the created global temporary table SYSIBM.USS_CMD_OUTPUT:

Table 193. Result set row for ADMIN_COMMAND_UNIX result set

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
TEXT	VARCHAR(32677)	A block of text or a line from the output messages of a z/OS UNIX System Services command

ADMIN_DS_BROWSE stored procedure

The SYSPROC.ADMIN_DS_BROWSE stored procedure returns either text or binary records from a physical sequential (PS) data set, generation data set, or partitioned data set (PDS) or partitioned data set extended (PDSE) member. This stored procedure supports only data sets with LRECL=80 and RECFM=FB.

Environment

The load module for ADMIN_DS_BROWSE, DSNADMDB, must reside in an APF-authorized library. ADMIN_DS_BROWSE runs in a WLM-established stored procedures address space, and all libraries in this WLM proc STEPLIB DD concatenation must be APF-authorized.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNADMDB
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The ADMIN_DS_BROWSE caller also needs authorization from an external security system, such as RACF, in order to browse or view an z/OS data set resource.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
▶▶—CALL—SYSPROC.ADMIN_DS_BROWSE—(—data-type,—data-set-name,—member-name,—dump-option,—  
▶—return-code,—message—)
```

Option descriptions

data-type

Specifies the type of data to be browsed. Possible values are:

- 1 Text data
- 2 Binary data

This is an input parameter of type INTEGER and cannot be null.

data-set-name

Specifies the name of the data set, or of the library that contains the member to be browsed. Possible values are:

PS data set name

If reading from a PS data set, the *data-set-name* contains the name of the PS data set.

PDS or PDSE name

If reading from a member that belongs to this PDS or PDSE, the *data-set-name* contains the name of the PDS or PDSE.

GDS name

If reading from a generation data set, the *data-set-name* contains the name of the generation data set, such as USERGDG.FILE.G0001V00.

This is an input parameter of type CHAR(44) and cannot be null.

member-name

Specifies the name of the PDS or PDSE member, if reading from a PDS or PDSE member. Otherwise, a blank character.

This is an input parameter of type CHAR(8) and cannot be null.

dump-option

Specifies whether to use the DB2 standard dump facility to dump the information necessary for problem diagnosis when an SQL error occurred or when a call to the IBM routine IEFDB476 to get messages about an unsuccessful SVC 99 call failed.

Possible values are:

- Y Generate a dump.
- N Do not generate a dump.

This is an input parameter of type CHAR(1) and cannot be null.

return-code

Provides the return code from the stored procedure. Possible values are:

- 0 The call completed successfully.
- 12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 or by z/OS might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_DS_BROWSE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
/***** DB2 Host Variables *****/
EXEC SQL BEGIN DECLARE SECTION;

/* SYSPROC.ADMIN_DS_BROWSE parameters */
long int datatype; /* Data type */
char dsname[45]; /* Data set name */
char mbrname[9]; /* Library member name */
char dumpopt[2]; /* Dump option */
long int retcd; /* Return code */
char errmsg[1332]; /* Error message */

/* Result set locators */
volatile SQL TYPE IS RESULT_SET_LOCATOR *rs_loc1;

/* Result set row */
long int rownum; /* Sequence number of the */
/* table row */
char text_rec[81]; /* A data set record */
EXEC SQL END DECLARE SECTION;

/***** Assign values to input parameters to browse a library member *****/
datatype = 1;
strcpy(dsname, "USER.DATASET.PDS");
strcpy(mbrname, "MEMBER0A");
strcpy(dumpopt, "N");

/***** Call stored procedure SYSPROC.ADMIN_DS_BROWSE *****/
EXEC SQL CALL SYSPROC.ADMIN_DS_BROWSE
(:datatype, :dsname, :mbrname, :dumpopt,
:retcd, :errmsg);

/***** Retrieve result set when the SQLCODE from the call is +446, *****/
/* which indicates that result sets were returned */
}
```

```

/*****
if (SQLCODE == +466)          /* Result sets were returned */
{
  /* Establish a link between the result set and its locator */
  EXEC SQL ASSOCIATE LOCATORS (:rs_loc1)
        WITH PROCEDURE SYSPROC.ADMIN_DS_BROWSE;

  /* Associate a cursor with the result set */
  EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :rs_loc1;

  /* Perform fetches using C1 to retrieve all rows from the */
  /* result set */
  EXEC SQL FETCH C1 INTO :rownum, :text_rec;
  while(SQLCODE==0)
  {
    EXEC SQL FETCH C1 INTO :rownum, :text_rec;
  }
}

return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 857:

- *return-code*
- *message*

In addition to the preceding output, the stored procedure returns one result set that contains the text or binary records read.

The following table shows the format of the result set returned in the created global temporary table SYSIBM.TEXT_REC_OUTPUT containing text records read:

Table 194. Result set row for ADMIN_DS_BROWSE result set (text records)

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i> .
TEXT_REC	VARCHAR(80)	Record read (text format).

The following table shows the format of the result set returned in the created global temporary table SYSIBM.BIN_REC_OUTPUT containing binary records read:

Table 195. Result set row for ADMIN_DS_BROWSE result set (binary records)

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i> .
BINARY_REC	VARCHAR(80) FOR BIT DATA	Record read (binary format).

ADMIN_DS_DELETE stored procedure

The SYSPROC.ADMIN_DS_DELETE stored procedure deletes a physical sequential (PS) data set, a partitioned data set (PDS) or partitioned data set extended (PDSE), a generation data set (GDS), or a member of a PDS or PDSE.

Environment

The load module for ADMIN_DS_DELETE, DSNADMDD, must reside in an APF-authorized library. ADMIN_DS_DELETE runs in a WLM-established stored procedures address space, and all libraries in this WLM proc STEPLIB DD concatenation must be APF-authorized.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on the ADMIN_DS_DELETE stored procedure
- Ownership of the stored procedure
- SYSADM authority

The ADMIN_DS_DELETE caller also needs authorization from an external security system, such as RACF, in order to delete an z/OS data set resource.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
▶—CALL—SYSPROC.ADMIN_DS_DELETE—(—data-set-type,—data-set-name,—parent-data-set-name,—————▶  
▶—dump-option,—return-code,—message—)—————▶◀
```

Option descriptions

data-set-type

Specifies the type of data set to delete. Possible values are:

- 1 Partitioned data set (PDS)
- 2 Partitioned data set extended (PDSE)
- 3 Member of a PDS or PDSE
- 4 Physical sequential data set (PS)
- 6 Generation data set (GDS)

This is an input parameter of type INTEGER and cannot be null.

data-set-name

Specifies the name of the data set, library member, or GDS absolute generation number to be deleted. Possible values are:

PS, PDS, or PDSE name

If *data-set-type* is 1, 2, or 4, the *data-set-name* contains the name of the PS, PDS, or PDSE to be deleted.

PDS or PDSE member name

If *data-set-type* is 3, the *data-set-name* contains the name of the PDS or PDSE member to be deleted.

absolute generation number

If *data-set-type* is 6, the *data-set-name* contains the absolute generation number of the GDS to be deleted, such as G0001V00.

This is an input parameter of type CHAR(44) and cannot be null.

parent-data-set-name

Specifies the name of the library that contains the member to be deleted, or of the GDG that contains the GDS to be delete. Otherwise blank. Possible values are:

blank If *data-set-type* is 1, 2, or 4, the *parent-data-set-name* is left blank.

PDS or PDSE name

If *data-set-type* is 3, the *parent-data-set-name* contains the name of the PDS or PDSE whose member is to be deleted.

GDG name

If *data-set-type* is 6, the *parent-data-set-name* contains the name of the GDG that the GDS to be deleted belongs to.

This is an input parameter of type CHAR(44) and cannot be null.

dump-option

Specifies whether to use the DB2 standard dump facility to dump the information necessary for problem diagnosis when a call to the IBM routine IEFDB476 to get messages about an unsuccessful SVC 99 call failed.

Possible values are:

Y Generate a dump.

N Do not generate a dump.

This is an input parameter of type CHAR(1) and cannot be null.

return-code

Provides the return code from the stored procedure. Possible values are:

0 Data set, PDS member, PDSE member, or GDS was deleted successfully.

12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by z/OS might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_DS_DELETE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;
```

```

int main( int argc, char *argv[] )    /* Argument count and list    */
{
  /****** DB2 Host Variables *****/
  EXEC SQL BEGIN DECLARE SECTION;

  /* SYSPROC.ADMIN_DS_DELETE parameters */
  long int  dstype;                    /* Data set type                */
  char      dsname[45];                /* Data set name ,             */
                                          /* member name, or             */
                                          /* generation # (G0001V00)    */
  char      parentds[45];              /* PDS, PDSE, GDG or blank    */
  char      dumpopt[2];                /* Dump option                  */
  long int  retcd;                     /* Return code                  */
  char      errmsg[1332];              /* Error message                */
  EXEC SQL END DECLARE SECTION;

  /****** Assign values to input parameters to delete a data set *****/
  dstype = 4;
  strcpy(dsname, "USER.DATASET.PDS");
  strcpy(parentds, " ");
  strcpy(dumpopt, "N");

  /****** Call stored procedure SYSPROC.ADMIN_DS_DELETE *****/
  EXEC SQL CALL SYSPROC.ADMIN_DS_DELETE
              (:dstype, :dsname, :parentds, :dumpopt,
              :retcd, :errmsg);

  return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 860:

- *return-code*
- *message*

ADMIN_DS_LIST stored procedure

The SYSPROC.ADMIN_DS_LIST stored procedure returns a list of data set names, generation data group (GDG), partitioned data set (PDS) or partitioned data set extended (PDSE) members, or generation data sets of a GDG.

Environment

The load module for ADMIN_DS_LIST, DSNADMMDL, must reside in an APF-authorized library. ADMIN_DS_LIST runs in a WLM-established stored procedures address space, and all libraries in this WLM proc STEPLIB DD concatenation must be APF-authorized.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNADMMDL

- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The ADMIN_DS_LIST caller also needs authorization from an external security system, such as RACF, in order to perform the requested operation on an z/OS data set resource.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```

▶▶—CALL—SYSPROC.ADMIN_DS_LIST—(—data-set-name,—list-members,—list-generations,—
▶—max-results,—dump-option,—return-code,—message—)————▶▶

```

Option descriptions

data-set-name

Specifies the data set name. You can use masking characters. For example:
USER.*

If no masking characters are used, only one data set will be listed.

This is an input parameter of type CHAR(44) and cannot be null.

list-members

Specifies whether to list PDS or PDSE members. Possible values are:

Y List members. Only set to Y when *data-set-name* is a fully qualified PDS or PDSE.

N Do not list members.

This is an input parameter of type CHAR(1) and cannot be null.

list-generations

Specifies whether to list generation data sets. Possible values are:

Y List generation data sets. Only set to Y when *data-set-name* is a fully qualified GDG.

N Do not list generation data sets.

This is an input parameter of type CHAR(1) and cannot be null.

max-results

Specifies the maximum number of result set rows. This option is applicable only when both *list-members* and *list-generations* are 'N'.

This is an input parameter of type INTEGER and cannot be null.

dump-option

Specifies whether to use the DB2 standard dump facility to dump the information necessary for problem diagnosis when any of the following errors occur:

- SQL error.

- A call to the IBM routine IEFDB476 to get messages about an unsuccessful SVC 99 call failed.
- Load Catalog Search Interface module error.

Possible values are:

- Y Generate a dump.
- N Do not generate a dump.

This is an input parameter of type CHAR(1) and cannot be null.

return-code

Provides the return code from the stored procedure. Possible values are:

- 0 The call completed successfully.
- 12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 or by z/OS might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_DS_LIST:

```
#pragma csect(CODE,"SAMDLPGM")
#pragma csect(STATIC,"PGMDLSAM")
#pragma runopts(plist(os))

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
  /***** DB2 Host Variables *****/
  EXEC SQL BEGIN DECLARE SECTION;

  /* SYSPROC.ADMIN_DS_LIST parameters */
  char dsname[45]; /* Data set name or filter */
  char listmbr[2]; /* List library members */
  char listgds[2]; /* List GDS */
  long int maxresult; /* Maximum result set rows */
  char dumpopt[2]; /* Dump option */
  long int retcd; /* Return code */
  char errmsg[1332]; /* Error message */

  /* Result set locators */
  volatile SQL TYPE IS RESULT_SET_LOCATOR *rs_loc1;

  /* Result set row */
  char dsnamer[45]; /* Data set name,
                   /* library member name, or
                   /* absolute generation number */
}
```

```

long int    createyr;           /* Create year          */
long int    createday;        /* Create day           */
long int    type;             /* Data set type        */
char        volume[7];        /* Data set volume      */
long int    primaryext;       /* Size of first extent */
long int    secondext;        /* Size of secondary extent */
char        measure[10];      /* Extent unit of measurement */
long int    extinuse;         /* Current allocated extents */
char        dasduse[9];       /* DASD usage           */
char        harba[7];         /* High allocated RBA   */
char        hurba[7];         /* High used RBA        */
EXEC SQL END DECLARE SECTION;

char * ptr;
int i = 0;
/*****
/* Assign values to input parameters to list all members of
/* a library
*****/
strcpy(dsname, "USER.DATASET.PDS");
strcpy(listmbr, "Y");
strcpy(listgds, "N");
maxresult = 1;
strcpy(dumpopt, "N");

/*****
/* Call stored procedure SYSPROC.ADMIN_DS_LIST
*****/
EXEC SQL CALL SYSPROC.ADMIN_DS_LIST
              (:dsname, :listmbr, :listgds, :maxresult,
              :dumpopt, :retcd, :errmsg);

/*****
/* Retrieve result set when the SQLCODE from the call is +446,
/* which indicates that result sets were returned
*****/
if (SQLCODE == +446)          /* Result sets were returned */
{
  /* Establish a link between the result set and its locator
  EXEC SQL ASSOCIATE LOCATORS (:rs_loc1)
  WITH PROCEDURE SYSPROC.ADMIN_DS_LIST;

  /* Associate a cursor with the result set
  EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :rs_loc1;

  /* Perform fetches using C1 to retrieve all rows from the
  /* result set
  EXEC SQL FETCH C1 INTO :dsnamer, :createyr, :createday,
                        :type, :volume, :primaryext,
                        :secondext, :measure, :extinuse,
                        :dasduse, :harba, :hurba;

  while(SQLCODE==0)
  {
    EXEC SQL FETCH C1 INTO :dsnamer, :createyr, :createday,
                          :type, :volume, :primaryext,
                          :secondext, :measure, :extinuse,
                          :dasduse, :harba, :hurba;
  }
}
return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 863:

- *return-code*
- *message*

In addition to the preceding output, the stored procedure returns one result set that contains the list of data sets, GDGs, PDS or PDSE members, or generation data sets that were requested.

The following table shows the format of the result set returned in the created global temporary table SYSIBM.DSLIST:

Table 196. Result set row for ADMIN_DS_LIST result set

Column name	Data type	Contents
DSNAME	VARCHAR(44)	<ul style="list-style-type: none">• Data set name, if <i>list-members</i> is “N” and <i>list-generations</i> is “N”.• Member name, if <i>list-members</i> is “Y”.• Absolute generation number (of the form G0000V00) from a generation data set name, if <i>list-generations</i> is “Y”.
CREATE_YEAR	INTEGER	The year that the data set was created. Not applicable for member and VSAM cluster.
CREATE_DAY	INTEGER	The day of the year that the data set was created, as an integer in the range of 1 to 366 where 1 represents January 1). Not applicable for member and VSAM cluster.

Table 196. Result set row for ADMIN_DS_LIST result set (continued)

Column name	Data type	Contents
TYPE	INTEGER	Type of data set. Possible values are: 0 Unknown type of data set 1 PDS data set 2 PDSE data set 3 Member of PDS or PDSE 4 Physical sequential data set 5 Generation data group 6 Generation data set 8 VSAM cluster 9 VSAM data component 10 VSAM index component
VOLUME	CHAR(6)	Volume where data set resides. Not applicable for member and VSAM cluster.
PRIMARY_EXTENT	INTEGER	Size of first extent. Not applicable for member and VSAM cluster.
SECONDARY_EXTENT	INTEGER	Size of secondary extent. Not applicable for member and VSAM cluster.
MEASUREMENT_UNIT	CHAR(9)	Unit of measurement for first extent and secondary extent. Possible values are: • BLOCKS • BYTES • CYLINDERS • KB • MB • TRACKS Not applicable for member and VSAM cluster.
EXTENTS_IN_USE	INTEGER	Current allocated extents. Not applicable for member and VSAM cluster.
DASD_USAGE	CHAR(8) FOR BIT DATA	Disk usage. For VSAM data and VSAM index only.
HARBA	CHAR(6) FOR BIT DATA	High allocated RBA. For VSAM data and VSAM index only.
HURBA	CHAR(6) FOR BIT DATA	High used RBA. For VSAM data and VSAM index only.

When a data set spans more than one volume, one row is returned for each volume that contains a piece of the data set. The VOLUME, EXTENTS_IN_USE, DASD_USAGE, HARBA, and HURBA columns reflect information for the specified volume.

ADMIN_DS_RENAME stored procedure

The SYSPROC.ADMIN_DS_RENAME stored procedure renames a physical sequential (PS) data set, a partitioned data set (PDS) or partitioned data set extended (PDSE), or a member of a PDS or PDSE.

Environment

The load module for ADMIN_DS_RENAME, DSNADMDR, must reside in an APF-authorized library. ADMIN_DS_RENAME runs in a WLM-established stored procedures address space, and all libraries in this WLM proc STEPLIB DD concatenation must be APF-authorized.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on the ADMIN_DS_RENAME stored procedure
- Ownership of the stored procedure
- SYSADM authority

The ADMIN_DS_RENAME caller also needs authorization from an external security system, such as RACF, in order to rename an z/OS data set resource.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
►►—CALL—SYSPROC.ADMIN_DS_RENAME—(—data-set-type,—data-set-name,—parent-data-set-name,—————►  
►—new-data-set-name,—dump-option,—return-code,—message—)—————►◄◄
```

Option descriptions

data-set-type

Specifies the type of data set to rename. Possible values are:

- 1 Partitioned data set (PDS)
- 2 Partitioned data set extended (PDSE)
- 3 Member of a PDS or PDSE
- 4 Physical sequential data set (PS)

This is an input parameter of type INTEGER and cannot be null.

data-set-name

Specifies the data set or member to be renamed. Possible values are:

PS, PDS, or PDSE name

If *data-set-type* is 1, 2, or 4, the *data-set-name* contains the name of the PS, PDS, or PDSE to be renamed.

PDS or PDSE member name

If *data-set-type* is 3, the *data-set-name* contains the name of the PDS or PDSE member to be renamed.

This is an input parameter of type CHAR(44) and cannot be null.

parent-data-set-name

Specifies the name of the PDS or PDSE, if renaming a PDS or PDSE member. Otherwise, a blank character. Possible values are:

blank If *data-set-type* is 1, 2, or 4, the *parent-data-set-name* is left blank.

PDS or PDSE name

If *data-set-type* is 3, the *parent-data-set-name* contains the name of the PDS or PDSE whose member is to be renamed.

This is an input parameter of type CHAR(44) and cannot be null.

new-data-set-name

Specifies the new data set or member name. Possible values are:

new data set name

If *data-set-type* is 1, 2, or 4, the *new-data-set-name* contains the new data set name.

new member name

If *data-set-type* is 3, the *new-data-set-name* contains the new member name.

This is an input parameter of type CHAR(44) and cannot be null.

dump-option

Specifies whether to use the DB2 standard dump facility to dump the information necessary for problem diagnosis when any of the following errors occurred:

- A call to the IBM routine IEFDB476 to get messages about an unsuccessful SVC 99 call failed.
- Load IDCAMS program error.

Possible values are:

Y Generate a dump.

N Do not generate a dump.

This is an input parameter of type CHAR(1) and cannot be null.

return-code

Provides the return code from the stored procedure. Possible values are:

0 The data set, PDS member, or PDSE member was renamed successfully.

12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages based on *return-code* and *data-set-type* combinations.

<i>return-code</i>	<i>data-set-type</i>	Content
0	1, 2, or 4	Contains IDCAMS messages.
0	3	No message is returned.
Not 0	not applicable	Contains messages describing the error encountered by the stored procedure. The first messages are generated by the stored procedure and messages that are generated by z/OS might follow these first messages. The first messages can also be generated by z/OS.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_DS_RENAME:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
  /***** DB2 Host Variables *****/
  EXEC SQL BEGIN DECLARE SECTION;

  /* SYSPROC.ADMIN_DS_RENAME parameters */
  long int  dstype; /* Data set type */
  char      dsname[45]; /* Data set or member name */
  char      parentds[45]; /* Parent data set (PDS or
                          /* PDSE) name or blank */
  char      newdsname[45]; /* New data set or member name*/
  char      dumpopt[2]; /* Dump option */
  long int  retcd; /* Return code */
  char      ermmsg[1332]; /* Error message */
  EXEC SQL END DECLARE SECTION;

  /***** Assign values to input parameters to rename a library member *****/
  dstype = 3;
  strcpy(dsname, "MEMBER01");
  strcpy(parentds, "USER.DATASET.PDS");
  strcpy(newdsname, "MEMBER0A");
  strcpy(dumpopt, "N");

  /***** Call stored procedure SYSPROC.ADMIN_DS_RENAME *****/
  EXEC SQL CALL SYSPROC.ADMIN_DS_RENAME
              (:dstype, :dsname, :parentds, :newdsname,
```

```

                                :dumpopt, :retcd, :errmsg);

    return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 868:

- *return-code*
- *message*

ADMIN_DS_SEARCH stored procedure

The SYSPROC.ADMIN_DS_SEARCH stored procedure determines if a physical sequential (PS) data set, partitioned data set (PDS), partitioned data set extended (PDSE), generation data group (GDG), generation data set (GDS) is cataloged, or if a library member of a cataloged PDS or PDSE exists.

Environment

The load module for ADMIN_DS_SEARCH, DSNADMDE, must reside in an APF-authorized library. ADMIN_DS_SEARCH runs in a WLM-established stored procedures address space, and all libraries in this WLM proc STEPLIB DD concatenation must be APF-authorized.

Authorization

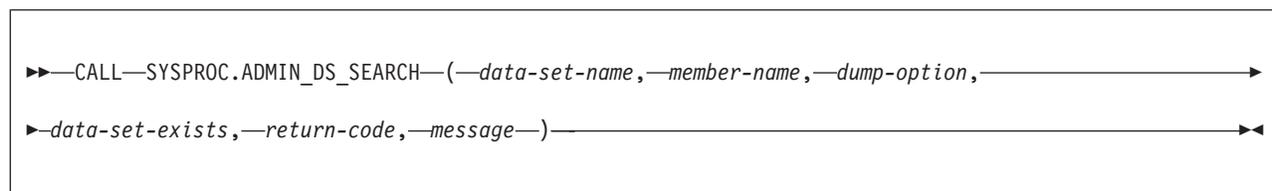
To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on the ADMIN_DS_SEARCH stored procedure
- Ownership of the stored procedure
- SYSADM authority

The ADMIN_DS_SEARCH caller also needs authorization from an external security system, such as RACF, in order to perform the requested operation on an z/OS data set resource.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:



Option descriptions

data-set-name

Specifies the name of a PS data set, PDS, PDSE, GDG or GDS.

This is an input parameter of type CHAR(44) and cannot be null.

member-name

Specifies the name of a PDS or PDSE member. Set this parameter to a blank character if you only want to check the existence of the PDS or PDSE.

This is an input parameter of type CHAR(8) and cannot be null.

dump-option

Specifies whether to use the DB2 standard dump facility to dump the information necessary for problem diagnosis when any of the following errors occurred:

- A call to the IBM routine IEFDB476 to get messages about an unsuccessful SVC 99 call failed.
- Load IDCAMS program error.

Possible values are:

- Y Generate a dump.
- N Do not generate a dump.

This is an input parameter of type CHAR(1) and cannot be null.

data-set-exists

Indicates whether a data set or library member exists or not. Possible values are:

- 1 Call did not complete successfully. Unable to determine if data set or member exists.
- 0 Data set or member was found
- 1 Data set not found
- 2 PDS or PDSE member not found

This is an output parameter of type INTEGER.

return-code

Provides the return code from the stored procedure. Possible values are:

- 0 The call completed successfully.
- 12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains IDCAMS messages if *return-code* is 0. Otherwise, contains messages describing the error encountered by the stored procedure. The first messages are generated by the stored procedure and messages that are generated by z/OS might follow these first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_DS_SEARCH:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;
```

```

int main( int argc, char *argv[] ) /* Argument count and list */
{
  /****** DB2 Host Variables *****/
  EXEC SQL BEGIN DECLARE SECTION;

  /* SYSPROC.ADMIN_DS_SEARCH parameters */
  char      dsname[45];          /* Data set name or GDG */
  char      mbrname[9];         /* Library member name */
  char      dumpopt[2];         /* Dump option */
  long int  exist;              /* Data set or library member */
                                   /* existence indicator */
  long int  retcd;              /* Return code */
  char      errmsg[1332];       /* Error message */
  EXEC SQL END DECLARE SECTION;

  /****** Assign values to input parameters to determine whether a
  /* library member exists or not *****/
  strcpy(dsname, "USER.DATASET.PDS");
  strcpy(mbrname, "MEMBER0A");
  strcpy(dumpopt, "N");

  /****** Call stored procedure SYSPROC.ADMIN_DS_SEARCH *****/
  EXEC SQL CALL SYSPROC.ADMIN_DS_SEARCH
              (:dsname, :mbrname, :dumpopt,
              :exist, :retcd, :errmsg);

  return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 871:

- *data-set-exists*
- *return-code*
- *message*

ADMIN_DS_WRITE stored procedure

The SYSPROC.ADMIN_DS_WRITE stored procedure writes either text or binary records passed in a global temporary table to either a physical sequential (PS) data set, partitioned data set (PDS) or partitioned data set extended (PDSE) member, or generation data set (GDS). It can either append or replace an existing PS data set, PDS or PDSE member, or GDS. It can create a new PS data set, PDS or PDSE data set or member, or a new GDS for an existing generation data group (GDG) as needed. This stored procedure supports only data sets with LRECL=80 and RECFM=FB.

Environment

The load module for ADMIN_DS_WRITE, DSNADMDW, must reside in an APF-authorized library. ADMIN_DS_WRITE runs in a WLM-established stored procedures address space, and all libraries in this WLM proc STEPLIB DD concatenation must be APF-authorized.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNADMWDW
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The ADMIN_DS_WRITE caller also needs authorization from an external security system, such as RACF, in order to write to an z/OS data set resource.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
▶▶—CALL—SYSPROC.ADMIN_DS_WRITE—(—data-type,—data-set-name,—member-name,—processing-option,—  
▶—dump-option,—return-code,—message—)—————▶▶
```

Option descriptions

This stored procedure takes the following input options:

data-type

Specifies the type of data to be saved. Possible values are:

- 1 Text data
- 2 Binary data

This is an input parameter of type INTEGER and cannot be null.

data-set-name

Specifies the name of the data set, GDG that contains the GDS, or library that contains the member, to be written to. Possible values are:

PS data set name

Name of the PS data set, if writing to a PS data set.

GDG name

Name of the GDG, if writing to a GDS within this GDG.

PDS or PDSE name

Name of the PDS or PDSE, if writing to a member that belongs to this library.

This is an input parameter of type CHAR(44) and cannot be null.

member-name

Specifies the relative generation number of the GDS, if writing to a GDS, or the name of the PDS or PDSE member, if writing to a PDS or PDSE member. Otherwise, a blank character. Possible values are:

GDS relative generation number

Relative generation number of a GDS, if writing to a GDS. For example: -1, 0, +1

PDS or PDSE member name

Name of the PDS or PDSE member, if writing to a library member.

blank In all other cases, blank.

This is an input parameter of type CHAR(8) and cannot be null.

processing-option

Specifies the type of operation. Possible values are:

R Replace

A Append

NM New member

ND New PS, PDS, PDSE, or GDS data set

This is an input parameter of type CHAR(2) and cannot be null.

dump-option

Specifies whether to use the DB2 standard dump facility to dump the information necessary for problem diagnosis when an SQL error has occurred or when a call to the IBM routine IEFDB476 to get messages about an unsuccessful SVC 99 call failed.

Possible values are:

Y Generate a dump.

N Do not generate a dump.

This is an input parameter of type CHAR(1) and cannot be null.

return-code

Provides the return code from the stored procedure. Possible values are:

0 The call completed successfully.

12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 or by z/OS might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Additional input

In addition to the input parameters, the stored procedure reads records to be written to a file from a created global temporary table. If the data to be written is text data, then the stored procedure reads records from SYSIBM.TEXT_REC_INPUT. If the data is binary data, then the stored procedure reads records from the created global temporary table SYSIBM.BIN_REC_INPUT.

The following table shows the format of the created global temporary table SYSIBM.TEXT_REC_INPUT containing text records to be saved:

Table 197. Additional input for text data for the ADMIN_DS_WRITE stored procedure

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i> .
TEXT_REC	CHAR(80)	Text record to be saved.

The following table shows the format of the created global temporary table SYSIBM.BIN_REC_INPUT containing binary records to be saved:

Table 198. Additional input for binary data for the ADMIN_DS_WRITE stored procedure

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i> .
BINARY_REC	VARCHAR(80) FOR BIT DATA	Binary record to be saved.

Example

The following C language sample shows how to invoke ADMIN_DS_WRITE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
    /***** DB2 Host Variables *****/
    EXEC SQL BEGIN DECLARE SECTION;

    /* SYSPROC.ADMIN_DS_WRITE parameters */
    long int datatype; /* Data type */
    char dsname[45]; /* Data set name or GDG */
    char mbrname[9]; /* Library member name,
                    /* generation # (-1, 0, +1),
                    /* or blank
    char procopt[3]; /* Processing option
    char dumpopt[2]; /* Dump option
    long int retcd; /* Return code
    char errmsg[1332]; /* Error message

    /* Temporary table SYSIBM.TEXT_REC_INPUT columns */
    long int rownum; /* Sequence number of the
                    /* table row
    char textrec[81]; /* Text record
    EXEC SQL END DECLARE SECTION;

    /*****
    /* Create the records to be saved
    /*****
    char dsrecord[12][50] = {
    "///IEBCOPY JOB ,CLASS=K,MSGCLASS=H,MSGLEVEL=(1,1)",
    "///STEP010 EXEC PGM=IEBCOPY",
    "///SYSPRINT DD SYSOUT=*",
    "///SYSUT3 DD SPACE=(TRK,(1,1)),UNIT=SYSDA",
```

```

//SYSUT4 DD SPACE=(TRK,(1,1)),UNIT=SYSDA",
"/*",
//DDI1 DD DSN=USER.DEV.LOADLIB1,DISP=SHR",
//DDO1 DD DSN=USER.DEV.LOADLIB2,DISP=SHR",
//SYSIN DD *,
" COPY OUTDD=DDO1,INDD=DDI1",
"/*",
"/*"
};
int i = 0;                                /* Loop counter */

/*****
/* Assign the values to input parameters to create a new      */
/* partitioned data set and member                             */
/*****
datatype = 1;
strcpy(dsname, "USER.DATASET.PDS");
strcpy(mbrname, "MEMBER01");
strcpy(procopt, "ND");
strcpy(dumpopt, "N");

/*****
/* Clear temporary table SYSIBM.TEXT_REC_INPUT                */
/*****
EXEC SQL DELETE FROM SYSIBM.TEXT_REC_INPUT;

/*****
/* Insert the records to be saved in the new library member   */
/* into the temporary table SYSIBM.TEXT_REC_INPUT             */
/*****
for (i = 0; i < 12; i++)
{
    rownum = i+1;
    strcpy(textrec, dsrecord[i]);
    EXEC SQL INSERT INTO SYSIBM.TEXT_REC_INPUT
        ( ROWNUM, TEXT_REC )
        VALUES (:rownum, :textrec);
};

/*****
/* Call stored procedure SYSPROC.ADMIN_DS_WRITE                */
/*****
EXEC SQL CALL SYSPROC.ADMIN_DS_WRITE
        (:datatype, :dsname, :mbrname, :procopt,
        :dumpopt, :retcd, :errmsg );

return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 874:

- *return-code*
- *message*

ADMIN_INFO_HOST stored procedure

The SYSPROC.ADMIN_INFO_HOST stored procedure returns the host name of a connected DB2 subsystem or the host name of every member of a data sharing group.

Environment

ADMIN_INFO_HOST runs in a WLM-established stored procedures address space.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNADMIH
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

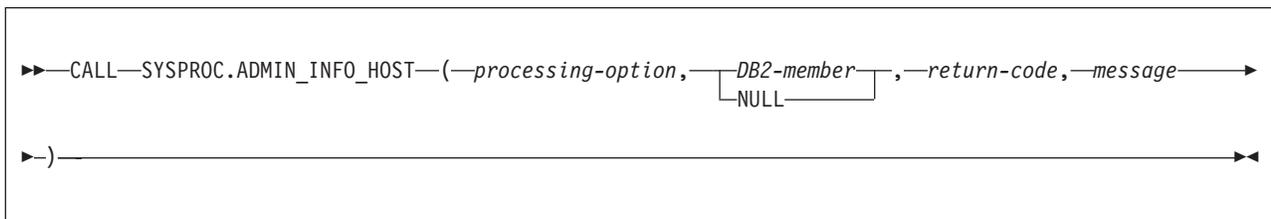
The ADMIN_INFO_HOST stored procedure internally calls the ADMIN_COMMAND_DB2 stored procedure to execute the following DB2 commands:

- -DISPLAY DDF
- -DISPLAY GROUP

The owner of the package or plan that contains the CALL ADMIN_INFO_HOST statement must also have the authorization required to execute the stored procedure ADMIN_COMMAND_DB2 and the specified DB2 commands. To determine the privilege or authority required to issue a DB2 command, see *DB2 Command Reference*.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:



Option descriptions

processing-option

Specifies processing option. Possible values are:

- 1 Return the host name of the connected DB2 subsystem or the host name of a specified DB2 data sharing group member.
For a data sharing group member, you must specify *DB2-member*.
- 2 Return the host name of every DB2 member of the same data sharing group.

This is an input parameter of type INTEGER and cannot be null.

DB2-member

Specifies the DB2 data sharing group member name.

This parameter must be null if *processing-option* is 2.

This is an input parameter of type CHAR(8).

return-code

Provides the return code from the stored procedure. Possible values are:

- 0 The call completed successfully.
- 4 Unable to list the host name of the connected DB2 subsystem or of every DB2 member of the same data sharing group due to one of the following reasons:
 - The IPADDR field returned when the -DISPLAY DDF command is executed on the connected DB2 subsystem or DB2 member contains the value -NONE
 - One of the DB2 members is down
- 12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_INFO_HOST:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
  /***** DB2 Host Variables *****/
  EXEC SQL BEGIN DECLARE SECTION;

  /* SYSPROC.ADMIN_INFO_HOST parameters */
  long int  procopt;          /* Processing option */
  short int ind_procopt;     /* Indicator variable */
  char      db2mbr[9];       /* Data sharing group member
                             /* name */
  short int ind_db2mbr;     /* Indicator variable */
  long int  retcd;          /* Return code */
  short int ind_retcd;      /* Indicator variable */
  char      errmsg[1332];    /* Error message */
  short int ind_errmsg;     /* Indicator variable */

  /* Result set locators */
  volatile SQL TYPE IS RESULT_SET_LOCATOR *rs_loc1;

  /* Result set row */
  long int  rownum;         /* Sequence number of the
                             /* table row */
  char      db2member[9];   /* DB2 data sharing group
                             /* member name */
}
```

```

char          hostname[256];          /* Host name of the connected */
                                          /* DB2 subsystem or DB2      */
                                          /* member name                */

EXEC SQL END DECLARE SECTION;

/*****
/* Assign values to input parameters to find the host name of      */
/* the connected DB2 subsystem                                     */
/* Set the indicator variables to 0 for non-null input parameters */
/* Set the indicator variables to -1 for null input parameters    */
*****/
procopt = 1;
ind_procopt = 0;
ind_db2mbr = -1;

/*****
/* Call stored procedure SYSPROC.ADMIN_INFO_HOST                    */
*****/
EXEC SQL CALL SYSPROC.ADMIN_INFO_HOST
           (:procopt   :ind_procopt,
           :db2mbr    :ind_db2mbr,
           :retcd     :ind_retcd,
           :errmsg    :ind_errmsg);

/*****
/* Retrieve result set when the SQLCODE from the call is +446,     */
/* which indicates that result sets were returned                 */
*****/
if (SQLCODE == +446)      /* Result sets were returned */
{
  /* Establish a link between the result set and its locator      */
  EXEC SQL ASSOCIATE LOCATORS (:rs_loc1)
        WITH PROCEDURE SYSPROC.ADMIN_INFO_HOST;

  /* Associate a cursor with the result set                        */
  EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :rs_loc1;

  /* Use C1 to fetch the only row from the result set            */
  EXEC SQL FETCH C1 INTO :rownum, :db2mbr, :hostname;
}

return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 878:

- *return-code*
- *message*

In addition to the preceding output, the stored procedure returns one result set that contains the host names.

The following table shows the format of the result set returned in the created global temporary table SYSIBM.SYSTEM_HOSTNAME:

Table 199. Result set row for ADMIN_INFO_HOST result set

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i> .

Table 199. Result set row for ADMIN_INFO_HOST result set (continued)

Column name	Data type	Contents
DB2_MEMBER	CHAR(8)	DB2 data sharing group member name.
HOSTNAME	VARCHAR(255)	Host name of the connected DB2 subsystem if the <i>processing-option</i> input parameter is 1 and the <i>DB2-member</i> input parameter is null. Otherwise, the host name of the DB2 member specified in the DB2_MEMBER column.

ADMIN_INFO_SSID stored procedure

The SYSPROC.ADMIN_INFO_SSID stored procedure returns the name of the connected DB2 subsystem.

GUPI

Environment

ADMIN_INFO_SSID must run in a WLM-established stored procedure address space.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on the ADMIN_INFO_SSID stored procedure
- Ownership of the stored procedure
- SYSADM authority

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
▶▶—CALL—SYSPROC.ADMIN_INFO_SSID—(—subsystem-ID,—return-code,—message—)————▶▶
```

Option descriptions

subsystem-ID

Identifies the subsystem ID of the connected DB2 subsystem

This is an output parameter of type VARCHAR(4).

return-code

Provides the return code from the stored procedure. Possible values are:

0 The call completed successfully.

12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_INFO_SSID:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
  /***** DB2 Host Variables *****/
  EXEC SQL BEGIN DECLARE SECTION;

  /* SYSPROC.ADMIN_INFO_SSID PARAMETERS */
  char      ssid[5];           /* DB2 subsystem identifier */
  long int  retcd;            /* Return code */
  char      errmsg[1332];     /* Error message */
  EXEC SQL END DECLARE SECTION;

  /***** Call stored procedure SYSPROC.ADMIN_INFO_SSID *****/
  EXEC SQL CALL SYSPROC.ADMIN_INFO_SSID
              (:ssid, :retcd, :errmsg);

  return(retcd);
}
```

Output

The output of this stored procedure is the following output parameters, which are described in “Option descriptions” on page 881:

- *subsystem-ID*
- *return-code*
- *message*



ADMIN_JOB_CANCEL stored procedure

The SYSPROC.ADMIN_JOB_CANCEL stored procedure purges or cancels a job.

Environment

The load module for ADMIN_JOB_CANCEL, DSNADMJP, must reside in an APF-authorized library. ADMIN_JOB_CANCEL runs in a WLM-established stored procedures address space, and all libraries in this WLM proc STEPLIB DD concatenation must be APF-authorized.

The load module for ADMIN_JOB_CANCEL, DSNADMJP, must be program controlled if the BPX.DAEMON.HFSCCTL FACILITY class profile has not been set up. For information on how to define DSNADMJP to program control, see installation job DSNTIJRA.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on the ADMIN_JOB_CANCEL stored procedure
- Ownership of the stored procedure
- SYSADM authority

The user specified in the *user-ID* input parameter of the SQL CALL statement also needs authorization from an external security system, such as RACF, in order to perform the requested operation.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
▶▶—CALL—SYSPROC.ADMIN_JOB_CANCEL—(—user-ID,—password,—processing-option,—job-ID,—————▶▶  
▶—return-code,—message—)—————▶▶
```

Option descriptions

user-ID

Specifies the user ID under which the job is canceled or purged.

This is an input parameter of type VARCHAR(128) and cannot be null.

password

Specifies the password associated with the input parameter *user-ID*.

The value of *password* is passed to the stored procedure as part of payload, and is not encrypted. It is not stored in dynamic cache when parameter markers are used.

This is an input parameter of type VARCHAR(24) and cannot be null.

processing-option

Identifies the type of command to invoke. Possible values are:

- 1 Cancel a job.
- 2 Purge a job.

This is an input parameter of type INTEGER and cannot be null.

job-ID

Specifies the job ID of the job to be canceled or purged. Acceptable formats are:

- *Jnnnnnnnn*
- *JOBnnnnnn*

where *n* is a digit between 0 and 9. For example: JOB01035

Both *Jnnnnnnnn* and *JOBnnnnnn* must be exactly 8 characters in length.

This is an input parameter of type CHAR(8) and cannot be null.

return-code

Provides the return code from the stored procedure. Possible values are:

- 0 The call completed successfully.
- 12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by z/OS might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_JOB_CANCEL:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
  /***** DB2 Host Variables *****/
  EXEC SQL BEGIN DECLARE SECTION;

  /* SYSPROC.ADMIN_JOB_CANCEL parameters */
  char      userid[129];          /* User ID */
  short int ind_userid;          /* Indicator variable */
  char      password[25];        /* Password */
  short int ind_password;        /* Indicator variable */
  long int  procopt;             /* Processing option */
  short int ind_procopt;         /* Indicator variable */
  char      jobid[9];            /* Job ID */
  short int ind_jobid;          /* Indicator variable */
  long int  retcd;               /* Return code */
  short int ind_retcd;           /* Indicator variable */
  char      errmsg[1332];        /* Error message */
  short int ind_errmsg;          /* Indicator variable */
  EXEC SQL END DECLARE SECTION;

  /***** Assign values to input parameters to purge a job */
  /* Set the indicator variables to 0 for non-null input parameters */
  /*****/
```

```

strcpy(userid, "USRT001");
ind_userid = 0;
strcpy(password, "NICETEST");
ind_password = 0;
procopt = 2;
ind_procopt = 0;
strcpy(jobid, "JOB00105");
ind_jobid = 0;

/*****
/* Call stored procedure SYSPROC.ADMIN_JOB_CANCEL          */
*****/
EXEC SQL CALL SYSPROC.ADMIN_JOB_CANCEL
              (:userid   :ind_userid,
              :password  :ind_password,
              :procopt   :ind_procopt,
              :jobid     :ind_jobid,
              :retcd     :ind_retcd,
              :errmsg    :ind_errmsg);

    return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 883:

- *return-code*
- *message*

ADMIN_JOB_FETCH stored procedure

The SYSPROC.ADMIN_JOB_FETCH stored procedure retrieves SYSOUT from JES spool and returns the SYSOUT.

Environment

The load module for ADMIN_JOB_FETCH, DSNADMJF, must reside in an APF-authorized library. ADMIN_JOB_FETCH runs in a WLM-established stored procedures address space, and all libraries in this WLM proc STEPLIB DD concatenation must be APF-authorized.

The load module for ADMIN_JOB_FETCH, DSNADMJF, must be program controlled if the BPX.DAEMON.HFSCCTL FACILITY class profile has not been set up. For information on how to define DSNADMJF to program control, see installation job DSNTIJRA.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNADMJF
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```
▶▶—CALL—SYSPROC.ADMIN_JOB_FETCH—(—user-ID,—password,—job-ID,—return-code,—message—)————▶▶
```

Option descriptions

user-ID

Specifies the user ID under which SYSOUT is retrieved.

This is an input parameter of type VARCHAR(128) and cannot be null.

password

Specifies the password associated with the input parameter *user-ID*.

The value of *password* is passed to the stored procedure as part of payload, and is not encrypted. It is not stored in dynamic cache when parameter markers are used.

This is an input parameter of type VARCHAR(24) and cannot be null.

job-ID

Specifies the JES2 or JES3 job ID whose SYSOUT data sets are to be retrieved.

This is an input parameter of type CHAR(8) and cannot be null.

return-code

Provides the return code from the stored procedure. Possible values are:

- 0** The call completed successfully.
- 12** The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_JOB_FETCH:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
    /***** DB2 Host Variables *****/
    EXEC SQL BEGIN DECLARE SECTION;
```

```

/* SYSPROC.ADMIN_JOB_FETCH parameters */
char      userid[129];          /* User ID */
short int ind_userid;          /* Indicator variable */
char      password[25];        /* Password */
short int ind_password;       /* Indicator variable */
char      jobid[9];           /* Job ID */
short int ind_jobid;          /* Indicator variable */
long int  retcd;              /* Return code */
short int ind_retcd;          /* Indicator variable */
char      errmsg[1332];       /* Error message */
short int ind_errmsg;         /* Indicator variable */

/* Result set locators */
volatile SQL TYPE IS RESULT_SET_LOCATOR *rs_loc1;

/* Result set row */
long int  rownum;              /* Sequence number of the */
/* table row */
char      text[4097];          /* A row in SYSOUT data set */
EXEC SQL END DECLARE SECTION;

/*****
/* Assign values to input parameters to fetch the SYSOUT of a job */
/* Set the indicator variables to 0 for non-null input parameters */
*****/
strcpy(userid, "USRT001");
ind_userid = 0;
strcpy(password, "NICETEST");
ind_password = 0;
strcpy(jobid, "JOB00100");
ind_jobid = 0;

/*****
/* Call stored procedure SYSPROC.ADMIN_JOB_FETCH */
*****/
EXEC SQL CALL SYSPROC.ADMIN_JOB_FETCH
           (:userid   :ind_userid,
            :password  :ind_password,
            :jobid    :ind_jobid,
            :retcd    :ind_retcd,
            :errmsg    :ind_errmsg);

/*****
/* Retrieve result set when the SQLCODE from the call is +446, */
/* which indicates that result sets were returned */
*****/
if (SQLCODE == +466) /* Result sets were returned */
{
  /* Establish a link between the result set and its locator */
  EXEC SQL ASSOCIATE LOCATORS (:rs_loc1)
        WITH PROCEDURE SYSPROC.ADMIN_JOB_FETCH;

  /* Associate a cursor with the result set */
  EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :rs_loc1;

  /* Perform fetches using C1 to retrieve all rows from the */
  /* result set */
  EXEC SQL FETCH C1 INTO :rownum, :text;
  while(SQLCODE==0)
  {
    EXEC SQL FETCH C1 INTO :rownum, :text;
  }
}

return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 886:

- *return-code*
- *message*

In addition to the preceding output, the stored procedure returns one result set that contains the data from the JES-managed SYSOUT data set that belong to the job ID specified in the input parameter *job-ID*.

The following table shows the format of the result set returned in the created global temporary table SYSIBM.JES_SYSOUT:

Table 200. Result set row for ADMIN_JOB_FETCH result set

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row
TEXT	VARCHAR(4096)	A record in the SYSOUT data set

ADMIN_JOB_QUERY stored procedure

The SYSPROC.ADMIN_JOB_QUERY stored procedure displays the status and completion information of a job.

Environment

The load module for ADMIN_JOB_QUERY, DSNADMJQ, must reside in an APF-authorized library. ADMIN_JOB_QUERY runs in a WLM-established stored procedures address space, and all libraries in this WLM proc STEPLIB DD concatenation must be APF-authorized.

The load module for ADMIN_JOB_QUERY, DSNADMJQ, must be program controlled if the BPX.DAEMON.HFCTL FACILITY class profile has not been set up. For information on how to define DSNADMJQ to program control, see installation job DSNTIJRA.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges:

- The EXECUTE privilege on the ADMIN_JOB_QUERY stored procedure
- Ownership of the stored procedure
- SYSADM authority

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

```

▶▶—CALL—SYSPROC.ADMIN_JOB_QUERY—(—user-ID,—password,—job-ID,—status,—max-RC,—
▶—completion-type,—system-abend-code,—user-abend-code,—return-code,—message—)————▶▶

```

Option descriptions

user-ID

Specifies the user ID under which the job is queried.

This is an input parameter of type VARCHAR(128) and cannot be null.

password

Specifies the password associated with the input parameter *user-ID*.

The value of *password* is passed to the stored procedure as part of payload, and is not encrypted. It is not stored in dynamic cache when parameter markers are used.

This is an input parameter of type VARCHAR(24) and cannot be null.

job-ID

Specifies the job ID of the job being queried. Acceptable formats are:

- Jnnnnnnnn
- JOBnnnnnn

where *n* is a digit between 0 and 9. For example: JOB01035

Both Jnnnnnnnn and JOBnnnnnn must be exactly 8 characters in length.

This is an input parameter of type CHAR(8) and cannot be null.

status

Identifies the current status of the job. Possible values are:

- 1 Job received, but not yet run (INPUT).
- 2 Job running (ACTIVE).
- 3 Job finished and has output to be printed or retrieved (OUTPUT).
- 4 Job not found.
- 5 Job in an unknown phase.

This is an output parameter of type INTEGER.

max-RC

Provides the job completion code.

This parameter is always null if querying in a JES3 z/OS Version 1.7 or earlier system. For JES3, this feature is only supported for z/OS Version 1.8 or higher.

This is an output parameter of type INTEGER.

completion-type

Identifies the job's completion type. Possible values are:

- 0 No completion information is available.
- 1 Job ended normally.
- 2 Job ended by completion code.
- 3 Job had a JCL error.

- 4 Job was canceled.
- 5 Job terminated abnormally.
- 6 Converter terminated abnormally while processing the job.
- 7 Job failed security checks.
- 8 Job failed in end-of-memory .

This parameter is always null if querying in a JES3 z/OS Version 1.7 or earlier system. For JES3, this feature is only supported for z/OS Version 1.8 or higher.

The *completion-type* information is the last six bits in the field STTRMXRC of the IAZSSST mapping macro. This information is returned via SSI 80. For additional information, see the discussion of the SSST macro in *z/OS MVS Data Areas*.

This is an output parameter of type INTEGER.

system-abend-code

Returns the system abend code if an abnormal termination occurs.

This parameter is always null if querying in a JES3 z/OS Version 1.7 or earlier system. For JES3, this feature is only supported for z/OS Version 1.8 or higher.

This is an output parameter of type INTEGER.

user-abend-code

Returns the user abend code if an abnormal termination occurs.

This parameter is always null if querying in a JES3 z/OS Version 1.7 or earlier system. For JES3, this feature is only supported for z/OS Version 1.8 or higher.

This is an output parameter of type INTEGER.

return-code

Provides the return code from the stored procedure. Possible values are:

- 0 The call completed successfully.
- 4 The job was not found, or the job status is unknown.
- 12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

This is an output parameter of type VARCHAR(1331).

Example

The following C language sample shows how to invoke ADMIN_JOB_QUERY:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
    /***** DB2 Host Variables *****/
```

```

EXEC SQL BEGIN DECLARE SECTION;

/* SYSPROC.ADMIN_JOB_QUERY parameters */
char      userid[129];          /* User ID */
short int ind_userid;          /* Indicator variable */
char      password[25];        /* Password */
short int ind_password;        /* Indicator variable */
char      jobid[9];            /* Job ID */
short int ind_jobid;           /* Indicator variable */
long int  stat;                /* Job status */
short int ind_stat;            /* Indicator variable */
long int  maxrc;               /* Job maxcc */
short int ind_maxrc;           /* Indicator variable */
long int  comptype;            /* Job completion type */
short int ind_comptype;        /* Indicator variable */
long int  sabndcd;             /* System abend code */
short int ind_sabndcd;         /* Indicator variable */
long int  uabndcd;             /* User abend code */
short int ind_uabndcd;         /* Indicator variable */
long int  retcd;               /* Return code */
short int ind_retcd;           /* Indicator variable */
char      errmsg[1332];        /* Error message */
short int ind_errmsg;          /* Indicator variable */
EXEC SQL END DECLARE SECTION;

/*****
/* Assign values to input parameters to query the status and
/* completion code of a job
/* Set the indicator variables to 0 for non-null input parameters */
*****/
strcpy(userid, "USRT001");
ind_userid = 0;
strcpy(password, "NICETEST");
ind_password = 0;
strcpy(jobid, "JOB00111");
ind_jobid = 0;

/*****
/* Call stored procedure SYSPROC.ADMIN_JOB_QUERY
*****/
EXEC SQL CALL SYSPROC.ADMIN_JOB_QUERY
           (:userid      :ind_userid,
            :password    :ind_password,
            :jobid       :ind_jobid,
            :stat        :ind_stat,
            :maxrc       :ind_maxrc,
            :comptype    :ind_comptype,
            :sabndcd     :ind_sabndcd,
            :uabndcd     :ind_uabndcd,
            :retcd       :ind_retcd,
            :errmsg       :ind_errmsg);

    return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 889:

- *status*
- *max-RC*
- *completion-type*
- *system-abend-code*
- *user-abend-code*

- *return-code*
- *message*

ADMIN_JOB_SUBMIT stored procedure

The SYSPROC.ADMIN_JOB_SUBMIT stored procedure submits a job to a JES2 or JES3 system.

Environment

ADMIN_JOB_SUBMIT runs in a WLM-established stored procedures address space.

The load module for ADMIN_JOB_SUBMIT, DSNADMJS, must be program controlled if the BPX.DAEMON.HFSCCTL FACILITY class profile has not been set up. For information on how to define DSNADMJS to program control, see installation job DSNTIJRA.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNADMJS
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

►►—CALL—SYSPROC.ADMIN_JOB_SUBMIT—(—*user-ID*,—*password*,—*job-ID*,—*return-code*,—*message*—)————►►

Option descriptions

user-ID

Specifies the user ID under which the job is submitted.

This is an input parameter of type VARCHAR(128) and cannot be null.

password

Specifies the password associated with the input parameter *user-ID*.

The value of *password* is passed to the stored procedure as part of payload, and is not encrypted. It is not stored in dynamic cache when parameter markers are used.

This is an input parameter of type VARCHAR(24) and cannot be null.

job-ID

Identifies the JES2 or JES3 job ID of the submitted job.

This is an output parameter of type CHAR(8).

return-code

Provides the return code from the stored procedure. Possible values are:

- 0 The call completed successfully.
- 12 The call did not complete successfully. The *message* output parameter contains messages describing the error.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Additional input

In addition to the input parameters, the stored procedure submits the job's JCL from the created global temporary table SYSIBM.JOB_JCL for execution.

The following table shows the format of the created global temporary table SYSIBM.JOB_JCL:

Table 201. Additional input for the ADMIN_JOB_SUBMIT stored procedure

Column name	Data type	Contents
ROWNUM	INTEGER	Sequence number of the table row, from 1 to <i>n</i>
STMT	VARCHAR(80)	A JCL statement

Example

The following C language sample shows how to invoke ADMIN_JOB_SUBMIT:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
  /***** DB2 Host Variables *****/
  EXEC SQL BEGIN DECLARE SECTION;

  /* SYSPROC.ADMIN_JOB_SUBMIT parameters */
  char      userid[129];          /* User ID */
  short int ind_userid;          /* Indicator variable */
  char      password[25];        /* Password */
  short int ind_password;        /* Indicator variable */
  char      jobid[9];            /* Job ID */
  short int ind_jobid;           /* Indicator variable */
  long int  retcd;               /* Return code */
  short int ind_retcd;           /* Indicator variable */
  char      errmsg[1332];        /* Error message */
  short int ind_errmsg;          /* Indicator variable */
```

```

/* Temporary table SYSIBM.JOB_JCL columns */
long int rownum; /* Sequence number of the */
/* table row */
char stmt[81]; /* JCL statement */
EXEC SQL END DECLARE SECTION;

/*****/
/* Create the JCL job to be submitted for execution */
/*****/
char jclstmt[12][50] = {
"//IEBCOPY JOB ,CLASS=K,MSGCLASS=H,MSGLEVEL=(1,1)",
"//STEP010 EXEC PGM=IEBCOPY",
"//SYSPRINT DD SYSOUT=*",
"//SYSUT3 DD SPACE=(TRK,(1,1)),UNIT=SYSDA",
"//SYSUT4 DD SPACE=(TRK,(1,1)),UNIT=SYSDA",
"//*",
"//DDI1 DD DSN=USER.DEV.LOADLIB1,DISP=SHR",
"//DDO1 DD DSN=USER.DEV.LOADLIB2,DISP=SHR",
"//SYSIN DD *",
" COPY OUTDD=DDO1,INDD=DDI1",
"/*",
"//*"
};
int i = 0; /* loop counter */

/*****/
/* Assign values to input parameters */
/* Set the indicator variables to 0 for non-null input parameters */
/*****/
strcpy(userid, "USRT001");
ind_userid = 0;
strcpy(password, "NICETEST");
ind_password = 0;

/*****/
/* Clear temporary table SYSIBM.JOB_JCL */
/*****/
EXEC SQL DELETE FROM SYSIBM.JOB_JCL;

/*****/
/* Insert the JCL job into the temporary table SYSIBM.JOB_JCL */
/*****/
for (i = 0; i < 12; i++)
{
rownum = i+1;
strcpy(stmt, jclstmt[i]);
EXEC SQL INSERT INTO SYSIBM.JOB_JCL
( ROWNUM, STMT)
VALUES (:rownum, :stmt);
};

/*****/
/* Call stored procedure SYSPROC.ADMIN_JOB_SUBMIT */
/*****/
EXEC SQL CALL SYSPROC.ADMIN_JOB_SUBMIT
(:userid :ind_userid,
:password :ind_password,
:jobid :ind_jobid,
:retcd :ind_retcd,
:errmsg :ind_errmsg);

return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 892:

- *job-ID*
- *return-code*
- *message*

ADMIN_UTL_SCHEDULE stored procedure

The SYSPROC.ADMIN_UTL_SCHEDULE stored procedure executes parallel utility execution.

Environment

ADMIN_UTL_SCHEDULE runs in a WLM-established stored procedures address space.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNADMUM
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

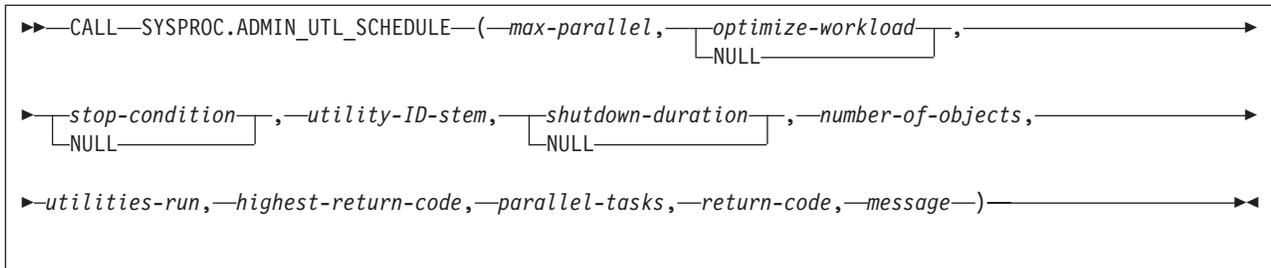
The ADMIN_UTL_SCHEDULE stored procedure internally calls the following stored procedures:

- ADMIN_COMMAND_DB2, to execute the DB2 DISPLAY UTILITY command
- ADMIN_INFO_SSID, to obtain the subsystem ID of the connected DB2 subsystem
- ADMIN_UTL_SORT, to sort objects into parallel execution units
- DSNUTILU, to run the requested utilities

The owner of the package or plan that contains the CALL ADMIN_UTL_SCHEDULE statement must also have the authorization required to execute these stored procedures and run the requested utilities. To determine the privilege or authority required to call DSNUTILU, see *DB2 Utility Guide and Reference*.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:



Option descriptions

max-parallel

Specifies the maximum number of parallel threads that may be started. The actual number may be lower than the requested number based on the optimizing sort result. Possible values are 1 to 99.

This is an input parameter of type SMALLINT and cannot be null.

optimize-workload

Specifies whether the parallel utility executions should be sorted to achieve shortest overall execution time. Possible values are:

NO or null

The workload is not to be sorted.

YES The workload is to be sorted.

This is an input parameter of type VARCHAR(8). The default value is NO.

stop-condition

Specifies the utility execution condition after which ADMIN_UTL_SCHEDULE will not continue starting new utility executions in parallel, but will wait until all currently running utilities have completed and will then return to the caller. Possible values are:

AUTHORIZ or null

No new utility executions will be started after one of the currently running utilities has encountered a return code from DSNUTILU of 12 or higher.

WARNING

No new utility executions will be started after one of the currently running utilities has encountered a return code from DSNUTILU of 4 or higher.

ERROR

No new utility executions will be started after one of the currently running utilities has encountered a return code from DSNUTILU of 8 or higher.

This is an input parameter of type VARCHAR(8). The default value is AUTHORIZ.

utility-ID-stem

Specifies the first part of the utility ID of a utility execution in a parallel thread. The complete utility ID is dynamically created in the form *utility-ID-stem* followed by *TT* followed by *NNNNNN*, where:

TT The zero-padded number of the subtask executing the utility

NNNNNN

A consecutive number of utilities executed in a subtask.

For example, `utilityidstem02000005` is the fifth utility execution that has been processed by the second subtask.

This is an input parameter of type `VARCHAR(8)` and cannot be null.

shutdown-duration

Specifies the number of seconds that `ADMIN_UTL_SCHEDULE` will wait for a utility execution to complete before a shutdown is initiated. When a shutdown is initiated, current utility executions can run to completion, and no new utility will be started. Possible values are:

null A shutdown will not be performed

1 to 999999999999999

A shutdown will be performed after this many seconds

This is an input parameter of type `FLOAT(8)`. The default value is null.

number-of-objects

As an input parameter, this specifies the number of utility executions and their sorting objects that were passed in the `SYSIBM.UTILITY_OBJECTS` table. Possible values are 1 to 999999.

As an output parameter, this specifies the number of objects that were passed in `SYSIBM.UTILITY_OBJECTS` table that are found in the DB2 catalog.

This is an input and output parameter of type `INTEGER` and cannot be null.

utilities-run

Indicates the number of actual utility executions.

This is an output parameter of type `INTEGER`.

highest-return-code

Indicates the highest return code from `DSNUTILU` for all utility executions.

This is an output parameter of type `INTEGER`.

parallel-tasks

Indicates the actual number of parallel tasks that were started to execute the utility in parallel.

This is an output parameter of type `SMALLINT`.

return-code

Provides the return code from the stored procedure. Possible values are:

0 All parallel utility executions ran successfully.

4 The statistics for one or more sorting objects have not been gathered in the catalog.

12 An `ADMIN_UTL_SCHEDULE` error occurred or all the objects passed in the `SYSIBM.UTILITY_OBJECTS` table are not found in the DB2 catalog. The *message* parameter contains details.

This is an output parameter of type `INTEGER`.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type `VARCHAR(1331)`.

Additional input

In addition to the input parameters, the stored procedure reads from the created global temporary tables SYSIBM.UTILITY_OBJECTS and SYSIBM.UTILITY_STMT.

The stored procedure reads objects for utility execution from SYSIBM.UTILITY_OBJECTS. The following table shows the format of the created global temporary table SYSIBM.UTILITY_OBJECTS:

Table 202. Format of the input objects

Column name	Data type	Contents
OBJECTID	INTEGER	A unique positive identifier for the object the utility execution is associated with. When you insert multiple rows, increment OBJECTID by 1, starting at 0 for every insert.
STMTID	INTEGER	A statement row in SYSIBM.UTILITY_STMT
TYPE	VARCHAR(10)	Object type: <ul style="list-style-type: none">• TABLESPACE• INDEXSPACE• TABLE• INDEX• STOGROUP
QUALIFIER	VARCHAR(128)	Qualifier (database or creator) of the object in NAME, empty or null for STOGROUP. If the qualifier is not provided and the type of the object is TABLESPACE or INDEXSPACE, then the default database is DSNDB04. If the object is of the type TABLE or INDEX, the schema is the current SQL authorization ID.
NAME	VARCHAR(128)	Unqualified name of the object. NAME cannot be null. If the object no longer exists, it will be ignored and the corresponding utility will not be executed.
PART	SMALLINT	Partition number of the object for which the utility will be invoked. Null or 0 if the object is not partitioned.
RESTART	VARCHAR(8)	Restart parameter of DSNUTILU

Table 202. Format of the input objects (continued)

Column name	Data type	Contents
UTILITY_NAME	VARCHAR(20)	<p>Utility name. UTILITY_NAME cannot be null.</p> <p>Recommendation: Sort objects for the same utility.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • CHECK DATA • CHECK INDEX • CHECK LOB • COPY • COPYTOCOPY • DIAGNOSE • LOAD • MERGECOPY • MODIFY RECOVERY • MODIFY STATISTICS • QUIESCE • REBUILD INDEX • RECOVER • REORG INDEX • REORG LOB • REORG TABLESPACE • REPAIR • REPORT RECOVERY • REPORT TABLESPACESET • RUNSTATS INDEX • RUNSTATS TABLESPACE • STOSPACE • UNLOAD

The stored procedure reads the corresponding utility statements from SYSIBM.UTILITY_STMT. The following table shows the format of the created global temporary table SYSIBM.UTILITY_STMT:

Table 203. Format of the utility statements

Column name	Data type	Contents
STMTID	INTEGER	A unique positive identifier for a single utility execution statement

Table 203. Format of the utility statements (continued)

Column name	Data type	Contents
STMTSEQ	INTEGER	If a utility statement exceeds 4000 characters, it can be split up and inserted into SYSIBM.UTILITY_STMT with the sequence starting at 0, and then being incremented with every insert. During the actual execution, the statement pieces are concatenated without any separation characters or blanks in between.
UTSTMT	VARCHAR(4000)	A utility statement or part of a utility statement. A placeholder &OBJECT. can be used to be replaced by the object name passed in SYSIBM.UTILITY_OBJECTS. A placeholder &THDINDEX. can be used to be replaced by the current thread index (01-99) of the utility being executed. You can use this when running REORG with SHRLEVEL CHANGE in parallel, so that you can specify a different mapping table for each thread of the utility execution.

Example

The following C language sample shows how to invoke ADMIN_UTL_SCHEDULE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
    /***** DB2 Host Variables *****/
    EXEC SQL BEGIN DECLARE SECTION;

    /* SYSPROC.ADMIN_UTL_SCHEDULE parameters */
    short int maxparallel; /* Max parallel */
    short int ind_maxparallel; /* Indicator variable */
    char optimizeworkload[9]; /* Optimize workload */
    short int ind_optimizeworkload; /* Indicator variable */
    char stoponcond[9]; /* Stop on condition */
    short int ind_stoponcond; /* Indicator variable */
    char utilityidstem[9]; /* Utility ID stem */
    short int ind_utilityidstem; /* Indicator variable */
    float shutdownduration; /* Shutdown duration */
    short int ind_shutdownduration; /* Indicator variable */
    long int numberofobjects; /* Number of objects */
}
```

```

short int ind_numberofobjects;      /* Indicator variable      */
long int utilitiesexec;             /* Utilities executed      */
short int ind_utilitiesexec;        /* Indicator variable      */
long int highestretcd;              /* DSNUTILU highest ret code */
short int ind_highestretcd;         /* Indicator variable      */
long int paralleltasks;             /* Parallel tasks          */
short int ind_paralleltasks;        /* Indicator variable      */
long int retcd;                     /* Return code             */
short int ind_retcd;                /* Indicator variable      */
char errmsg[1332];                  /* Error message           */
short int ind_errmsg;                /* Indicator variable      */

/* Temporary table SYSIBM.UTILITY_OBJECTS columns */
long int objectid;                  /* Object id                */
long int stmtid;                    /* Statement ID             */
char type[11];                      /* Object type (e.g. "INDEX") */
char qualifier[129];                /* Object qualifier        */
short int ind_qualifier;             /* Object qualifier ind. var. */
char name[129];                      /* Object name (qual. or unq.) */
short int part;                      /* Optional partition      */
short int ind_part;                  /* Partition indicator var  */
char restart[9];                     /* DSNUTILU restart parm   */
char utname[21];                     /* Utility name             */

/* Temporary table SYSIBM.UTILITY_STMT columns */
long int stmtid2;                    /* Statement ID             */
long int stmtseq;                    /* Utility stmt sequence    */
char utstmt[4001];                   /* Utility statement        */

/* Result set locators */
volatile SQL TYPE IS RESULT_SET_LOCATOR *rs_loc1;
volatile SQL TYPE IS RESULT_SET_LOCATOR *rs_loc2;

/* First result set row */
long int objectid1;                  /* Object id                */
long int textseq;                    /* Object utility output seq */
char text[255];                      /* Object utility output    */

/* Second result set row */
long int objectid2;                  /* Object id                */
long int utilretcd;                  /* DSNUTILU return code     */
EXEC SQL END DECLARE SECTION;

/*****
/* Set up the objects to be sorted */
/*****
long int objid_array[4] = {1, 2, 3, 4};
long int stmtid_array[4] = {1, 1, 1, 1};
char type_array[4][11] = {"TABLESPACE", "TABLESPACE",
                          "TABLESPACE", "TABLESPACE"};
char qual_array[4][129] = {"QUAL01", "QUAL01",
                           "QUAL01", "QUAL01"};
char name_array[4][129] = {"TBSP01", "TBSP02",
                           "TBSP03", "TBSP04"};
short int part_array[4] = {0, 0, 0, 0};
char restart_array[4][9] = {"NO", "NO",
                             "NO", "NO"};
char utname_array[4][21] = {"RUNSTATS TABLESPACE",
                            "RUNSTATS TABLESPACE",
                            "RUNSTATS TABLESPACE",
                            "RUNSTATS TABLESPACE"};

int i = 0;                            /* Loop counter */

/*****
/* Set up utility statement */
/*****

```

```

stmtid2 = 1;
stmtseq = 1;
strcpy(utstmt,
"RUNSTATS TABLESPACE &OBJECT. TABLE(ALL) SAMPLE 25 INDEX(ALL)");

/*****/
/* Assign values to input parameters */
/* Set the indicator variables to 0 for non-null input parameters */
/* Set the indicator variables to -1 for null input parameters */
/*****/
maxparallel = 2;
ind_maxparallel = 0;
strcpy(optimizeworkload, "YES");
ind_optimizeworkload = 0;
strcpy(stoportuncond, "AUTHORIZ");
ind_stoportuncond = 0;
strcpy(utilityidstem, "DSNADMUM");
ind_utilityidstem = 0;
numberofobjects = 4;
ind_numberofobjects = 0;
ind_shutdownduration = -1;

/*****/
/* Clear temporary table SYSIBM.UTILITY_OBJECTS */
/*****/
EXEC SQL DELETE FROM SYSIBM.UTILITY_OBJECTS;

/*****/
/* Insert the objects into the temporary table */
/* SYSIBM.UTILITY_OBJECTS */
/*****/
for (i = 0; i < 4; i++)
{
    objectid = objid_array[i];
    stmtid = stmtid_array[i];
    strcpy(type, type_array[i]);
    strcpy(qualifier, qual_array[i]);
    strcpy(name, name_array[i]);
    part = part_array[i];
    strcpy(restart, restart_array[i]);
    strcpy(utname, utname_array[i]);
    EXEC SQL INSERT INTO SYSIBM.UTILITY_OBJECTS
        (OBJECTID, STMTID, TYPE,
         QUALIFIER, NAME, PART,
         RESTART, UTILITY_NAME)
        VALUES (:objectid, :stmtid, :type,
                :qualifier, :name, :part,
                :restart, :utname);
};

/*****/
/* Clear temporary table SYSIBM.UTILITY_STMT */
/*****/
EXEC SQL DELETE FROM SYSIBM.UTILITY_STMT;

/*****/
/* Insert the utility statement into the temporary table */
/* SYSIBM.UTILITY_STMT */
/*****/
EXEC SQL INSERT INTO SYSIBM.UTILITY_STMT
    (STMTID, STMTSEQ, UTSTMT)
    VALUES (:stmtid2, :stmtseq, :utstmt);

/*****/
/* Call stored procedure SYSPROC.ADMIN_UTL_SCHEDULE */
/*****/
EXEC SQL CALL SYSPROC.ADMIN_UTL_SCHEDULE

```

```

(:maxparallel          :ind_maxparallel,
:optimizeworkload     :ind_optimizeworkload,
:stoponcond           :ind_stoponcond,
:utilityidstem        :ind_utilityidstem,
:shutdownduration     :ind_shutdownduration,
:numberofobjects      :ind_numberofobjects,
:utilitiesexec        :ind_utilitiesexec,
:highestretcd         :ind_highestretcd,
:paralleltasks        :ind_paralleltasks,
:retcd                :ind_retcd,
:errmsg               :ind_errmsg);

/*****
/* Retrieve result set when the SQLCODE from the call is +446,
/* which indicates that result sets were returned
*****/
if (SQLCODE == +466)          /* Result sets were returned */
{
  /* Establish a link between the result set and its locator
EXEC SQL ASSOCIATE LOCATORS (:rs_loc1, :rs_loc2)
      WITH PROCEDURE SYSPROC.ADMIN_UTL_SCHEDULE;

  /* Associate a cursor with the first result set
EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :rs_loc1;

  /* Associate a cursor with the second result set
EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :rs_loc2;

  /* Perform fetches using C1 to retrieve all rows from the
  /* first result set
EXEC SQL FETCH C1 INTO :objectid1, :textseq, :text;
while(SQLCODE==0)
{
  EXEC SQL FETCH C1 INTO :objectid1, :textseq, :text;
}

  /* Perform fetches using C2 to retrieve all rows from the
  /* second result set
EXEC SQL FETCH C2 INTO :objectid2, :utilretcd;
while(SQLCODE==0)
{
  EXEC SQL FETCH C2 INTO :objectid2, :utilretcd;
}
}

return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 896:

- *number-of-objects*
- *utilities-run*
- *highest-return-code*
- *parallel-tasks*
- *return-code*
- *message*

In addition to the preceding output, the stored procedure returns two results sets.

The first result set is returned in the created global temporary table SYSIBM.UTILITY_SYSPRINT and contains the output from the individual utility executions. The following table shows the format of the created global temporary table SYSIBM.UTILITY_SYSPRINT:

Table 204. Result set row for first ADMIN_UTL_SCHEDULE result set

Column name	Data type	Contents
OBJECTID	INTEGER	A unique positive identifier for the object the utility execution is associated with
TEXTSEQ	INTEGER	Sequence number of utility execution output statements for the object whose unique identifier is specified in the OBJECTID column
TEXT	VARCHAR(254)	A utility execution output statement

The second result set is returned in the created global temporary table SYSIBM.UTILITY_RETCODE and contains the return code for each of the individual DSNUTILU executions. The following table shows the format of the output created global temporary table SYSIBM.UTILITY_RETCODE:

Table 205. Result set row for second ADMIN_UTL_SCHEDULE result set

Column name	Data type	Contents
OBJECTID	INTEGER	A unique positive identifier for the object the utility execution is associated with
RETCODE	INTEGER	Return code from DSNUTILU for this utility execution

ADMIN_UTL_SORT stored procedure

The SYSPROC.ADMIN_UTL_SORT stored procedure sorts objects for parallel utility execution using JCL or the ADMIN_UTL_SCHEDULE stored procedure.

Environment

ADMIN_UTL_SORT runs in a WLM-established stored procedures address space.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

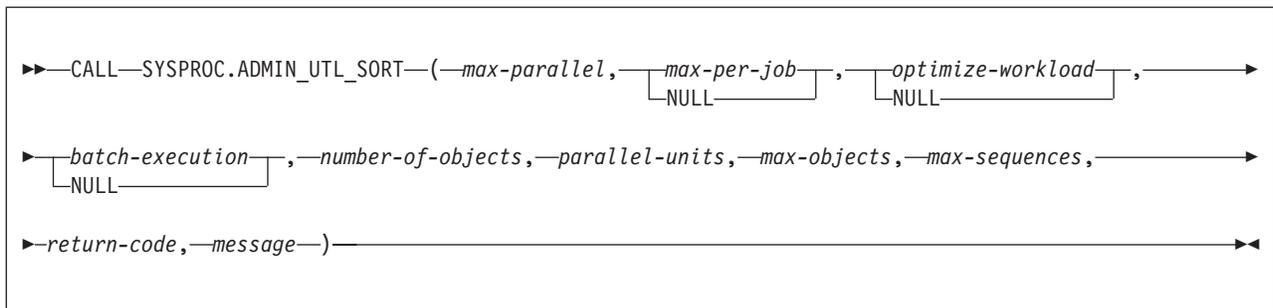
- The EXECUTE privilege on the package for DSNADMUS
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have SELECT authority on the following catalog tables:

- SYSIBM.SYSTABLEPART
- SYSIBM.SYSINDEXPART
- SYSIBM.SYSINDEXES
- SYSIBM.SYSTABLES

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:



Option descriptions

max-parallel

Specifies the maximum number of parallel units. The actual number may be lower than the requested number based on the optimizing sort result. Possible values are: 1 to 99.

This is an input parameter of type SMALLINT and cannot be null.

max-per-job

Specifies the maximum number of steps per job for batch execution. Possible values are:

1 to 255

Steps per job for batch execution

null Online execution

This is an input parameter of type SMALLINT. This parameter cannot be null if *batch-execution* is YES.

optimize-workload

Specifies whether the parallel units should be sorted to achieve shortest overall execution time. Possible values are:

NO or null

The workload is not to be sorted.

YES The workload is to be sorted.

This is an input parameter of type VARCHAR(8). The default value is NO.

batch-execution

Indicates whether the objects should be sorted for online or batch (JCL) execution.

NO or null

The workload is for online execution.

YES The workload is for batch execution.

This is an input parameter of type VARCHAR(8). The default value is NO.

number-of-objects

As an input parameter, this specifies the number of objects that were passed in SYSIBM.UTILITY_SORT_OBJ. Possible values are: 1 to 999999.

As an output parameter, this specifies the number of objects that were passed in SYSIBM.UTILITY_SORT_OBJ table that are found in the DB2 catalog.

This is an input and output parameter of type INTEGER and cannot be null.

parallel-units

Indicates the number of recommended parallel units.

This is an output parameter of type SMALLINT.

max-objects

Indicates the maximum number of objects in any parallel unit.

This is an output parameter of type INTEGER.

max-sequences

Indicates the number of jobs in any parallel unit.

This is an output parameter of type INTEGER.

return-code

Provides the return code from the stored procedure. Possible values are:

- 0 Sort ran successfully.
- 4 The statistics for one or more sorting objects have not been gathered in the catalog or the object no longer exists.
- 12 An ADMIN_UTL_SORT error occurred. The *message* parameter will contain details.

This is an output parameter of type INTEGER.

message

Contains messages describing the error encountered by the stored procedure. If no error occurred, then no message is returned.

The first messages in this area are generated by the stored procedure. Messages that are generated by DB2 might follow the first messages.

This is an output parameter of type VARCHAR(1331).

Additional input

In addition to the input parameters, this stored procedure reads the objects for sorting and the corresponding utility names from the created global temporary table SYSIBM.UTILITY_SORT_OBJ.

The following table shows the format of the created global temporary table SYSIBM.UTILITY_SORT_OBJ:

Table 206. Input for the ADMIN_UTL_SORT stored procedure

Column name	Data type	Contents
OBJECTID	INTEGER	A unique positive identifier for the object the utility execution is associated with. When you insert multiple rows, increment OBJECTID by 1, starting at 0 for every insert.
TYPE	VARCHAR(10)	Object type: <ul style="list-style-type: none"> • TABLESPACE • INDEXSPACE • TABLE • INDEX • STOGROUP
QUALIFIER	VARCHAR(128)	Qualifier (database or creator) of the object in NAME, empty or null for STOGROUP. If the qualifier is not provided and the type of the object is TABLESPACE or INDEXSPACE, then the default database is DSNDB04. If the object is of the type TABLE or INDEX, the schema is the current SQL authorization ID. If the object no longer exists, it will be ignored.
NAME	VARCHAR(128)	Unqualified name of the object. NAME cannot be null.
PART	SMALLINT	Partition number of the object for which the utility will be invoked. Null or 0 if the object is not partitioned.

Table 206. Input for the ADMIN_UTL_SORT stored procedure (continued)

Column name	Data type	Contents
UTILITY_NAME	VARCHAR(20)	<p>Utility name. UTILITY_NAME cannot be null.</p> <p>Recommendation: Sort objects for the same utility.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • CHECK DATA • CHECK INDEX • CHECK LOB • COPY • COPYTOCOPY • DIAGNOSE • LOAD • MERGECOPY • MODIFY RECOVERY • MODIFY STATISTICS • QUIESCE • REBUILD INDEX • RECOVER • REORG INDEX • REORG LOB • REORG TABLESPACE • REPAIR • REPORT RECOVERY • REPORT TABLESPACESET • RUNSTATS INDEX • RUNSTATS TABLESPACE • STOSPACE • UNLOAD

Example

The following C language sample shows how to invoke ADMIN_UTL_SORT:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

int main( int argc, char *argv[] ) /* Argument count and list */
{
    /***** DB2 Host Variables *****/
    EXEC SQL BEGIN DECLARE SECTION;

    /* SYSPROC.ADMIN_UTL_SORT parameters */
    short int maxparallel; /* Max parallel */
    short int ind_maxparallel; /* Indicator variable */
    short int maxperjob; /* Max per job */
    short int ind_maxperjob; /* Indicator variable */
    char optimizeworkload[9]; /* Optimize workload */
    short int ind_optimizeworkload; /* Indicator variable */
    char batchexecution[9]; /* Batch execution */
    short int ind_batchexecution; /* Indicator variable */
    long int numberofobjects; /* Number of objects */
    short int ind_numberofobjects; /* Indicator variable */
    short int parallelunits; /* Parallel units */
    short int ind_parallelunits; /* Indicator variable */
}
```

```

long int maxobjects;          /* Maximum objects per      */
                             /* parallel unit            */
short int ind_maxobjects;    /* Indicator variable       */
long int maxseqs;           /* Maximum jobs per unit   */
short int ind_maxseqs;      /* Indicator variable       */
long int retcd;             /* Return code              */
short int ind_retcd;        /* Indicator variable       */
char errmsg[1332];          /* Error message            */
short int ind_errmsg;       /* Indicator variable       */

/* Temporary table SYSIBM.UTILITY_SORT_OBJ columns */
long int objectid;          /* Object id                */
char type[11];              /* Object type (e.g. "INDEX") */
char qualifier[129];        /* Object qualifier         */
short int ind_qualifier;    /* Object qualifier ind. var. */
char name[129];             /* Object name (qual. or unq.) */
short int part;             /* Optional partition       */
short int ind_part;         /* Partition indicator var   */
char utname[21];           /* Utility name              */

/* Result set locators */
volatile SQL TYPE IS RESULT_SET_LOCATOR *rs_loc1;

/* Result set row */
long int resobjectid;       /* Object id                */
short int unit;             /* Execution unit value     */
long int unitseq;          /* Job seq within exec unit */
long int unitseqpos;       /* Pos within exec unit or  */
                             /* step within job          */
char exclusive[2];         /* Exclusive execution flag */
EXEC SQL END DECLARE SECTION;

/*****
/* Set up the objects to be sorted */
/*****
long int objid_array[4] = {0, 1, 2, 3};
char type_array[4][11] = {"TABLESPACE", "TABLESPACE",
                          "TABLESPACE", "TABLESPACE"};
char qual_array[4][129] = {"QUAL01", "QUAL01",
                          "QUAL01", "QUAL01"};
char name_array[4][129] = {"TBSP01", "TBSP02",
                          "TBSP03", "TBSP04"};
short int part_array[4] = {0, 0, 0, 0};
char utname_array[4][21] = {"RUNSTATS TABLESPACE",
                          "RUNSTATS TABLESPACE",
                          "RUNSTATS TABLESPACE",
                          "RUNSTATS TABLESPACE"};

int i = 0;                  /* Loop counter */

/*****
/* Assign values to input parameters */
/* Set the indicator variables to 0 for non-null input parameters */
/* Set the indicator variables to -1 for null input parameters */
/*****
maxparallel = 2;
ind_maxparallel = 0;
ind_maxperjob = -1;
strcpy(optimizeworkload, "YES");
ind_optimizeworkload = 0;
strcpy(batchexecution, "NO");
ind_batchexecution = 0;
numberofobjects = 4;
ind_numberofobjects = 0;

/*****
/* Clear temporary table SYSIBM.UTILITY_SORT_OBJ */
/*****

```

```

/*****/
EXEC SQL DELETE FROM SYSIBM.UTILITY_SORT_OBJ;

/*****/
/* Insert the objects into the temporary table */
/* SYSIBM.UTILITY_SORT_OBJ */
/*****/
for (i = 0; i < 4; i++)
{
    objectid = objid_array[i];
    strcpy(type, type_array[i]);
    strcpy(qualifier, qual_array[i]);
    strcpy(name, name_array[i]);
    part = part_array[i];
    strcpy(utname, utname_array[i]);
    EXEC SQL INSERT INTO SYSIBM.UTILITY_SORT_OBJ
        (OBJECTID, TYPE, QUALIFIER, NAME, PART,
         UTILITY_NAME)
        VALUES (:objectid, :type, :qualifier, :name, :part,
                :utname);
};

/*****/
/* Call stored procedure SYSPROC.ADMIN_UTL_SORT */
/*****/
EXEC SQL CALL SYSPROC.ADMIN_UTL_SORT
    (:maxparallel      :ind_maxparallel,
     :maxperjob        :ind_maxperjob,
     :optimizeworkload :ind_optimizeworkload,
     :batchexecution   :ind_batchexecution,
     :numberofobjects  :ind_numberofobjects,
     :parallelunits    :ind_parallelunits,
     :maxobjects       :ind_maxobjects,
     :maxseqs          :ind_maxseqs,
     :retcd            :ind_retcd,
     :errmsg           :ind_errmsg);

/*****/
/* Retrieve result set when the SQLCODE from the call is +446, */
/* which indicates that result sets were returned */
/*****/
if (SQLCODE == +466) /* Result sets were returned */
{
    /* Establish a link between the result set and its locator */
    EXEC SQL ASSOCIATE LOCATORS (:rs_loc1)
        WITH PROCEDURE SYSPROC.ADMIN_UTL_SORT;

    /* Associate a cursor with the result set */
    EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :rs_loc1;

    /* Perform fetches using C1 to retrieve all rows from the */
    /* result set */
    EXEC SQL FETCH C1 INTO :resobjectid, :unit,
        :unitseq, :unitseqpos, :exclusive;
    while(SQLCODE==0)
    {
        EXEC SQL FETCH C1 INTO :resobjectid, :unit,
            :unitseq, :unitseqpos, :exclusive;
    }
}

return(retcd);
}

```

Output

This stored procedure returns the following output parameters, which are described in “Option descriptions” on page 905:

- *number-of-objects*
- *parallel-units*
- *max-objects*
- *max-sequences*
- *return-code*
- *message*

In addition to the preceding output, the stored procedure returns one result set that contains the objects sorted into parallel execution units.

The following table shows the format of the result set returned in the created global temporary table SYSIBM.UTILITY_SORT_OUT:

Table 207. Result set row for ADMIN_UTL_SORT result set

Column name	Data type	Contents
OBJECTID	INTEGER	A unique positive identifier for the object
UNIT	SMALLINT	Number of parallel execution unit
UNIT_SEQ	INTEGER	Job sequence within parallel execution unit
UNIT_SEQ_POS	INTEGER	Step within job
EXCLUSIVE	CHAR(1)	Requires execution with nothing running in parallel

Common SQL API stored procedures

Common SQL API stored procedures implement a cross-database and cross-operating system SQL API that is portable across IBM data servers, including DB2 for Linux[®], UNIX, and Windows.

The Common SQL API is a solution-level API that supports common tooling across IBM data servers. This Common SQL API ensures that tooling does not break when a data server is upgraded, and it notifies the caller when an upgrade to tooling is available to capitalize on new data server functionality. Applications that support more than one IBM data server will benefit from using the Common SQL API, as it lowers the complexity of implementation. Such applications typically perform a variety of common administrative functions. For example, you can use these stored procedures to retrieve data server configuration information, return system information about the data server, and return the short message text for an SQLCODE.

These stored procedures use version-stable XML documents as parameters. These XML parameter documents adhere to a single, common document type definition (DTD). This DTD is flexible enough to represent hierarchical structures and binary data. The XML parameter documents can be parsed by using the Apache Commons Configuration component.

Each of the three XML parameter documents has a name and a version, and is typically associated with one stored procedure. The three types of XML parameter documents are:

- XML input document
- XML output document
- XML message document

The XML input document is passed as input to the stored procedure. The XML output document is returned as output, and the XML message document returns messages. If the structure, attributes, or types in an XML parameter document change, the version of the XML parameter document changes. The version of all three of these documents remains in sync when you call a stored procedure. For example, if you call the GET_SYSTEM_INFO stored procedure and specify the *major_version* parameter as 1 and the *minor_version* parameter as 1, the XML input, XML output, and XML message documents will be Version 1.1 documents.

Related information

 DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond

 Common DTD

 Apache Commons Configuration component

GET_CONFIG stored procedure

The GET_CONFIG stored procedure retrieves data server configuration information.

This data server configuration information includes:

- Data sharing group information
- DB2 subsystem status information
- DB2 subsystem parameters
- DB2 distributed access information
- Active log data set information
- The time of the last restart of DB2
- Resource limit facility information
- Connected DB2 subsystems information

Environment

The load module for the GET_CONFIG stored procedure, DSNADMGC, must reside in an APF-authorized library. The GET_CONFIG stored procedure runs in a WLM-established stored procedures address space, and all of the libraries that are specified in the STEPLIB DD statement must be APF-authorized.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have EXECUTE privilege on the GET_CONFIG stored procedure.

Syntax

```
CALL GET_CONFIG ( [major_version] , [minor_version] ,  
[requested_locale] , [xml_input] , [xml_filter] , [xml_output] , [xml_message] )
```

The schema is SYSPROC.

Option descriptions

major_version

An input and output parameter of type INTEGER that indicates the major document version. On input, this parameter indicates the major document version that you support for the XML documents that are passed as parameters in the stored procedure (*xml_input*, *xml_output*, and *xml_message*). The stored procedure processes all XML documents in the specified version, or returns an error (-20457) if the version is invalid.

On output, this parameter specifies the highest major document version that is supported by the procedure. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters. Currently, the highest and only major document version that is supported is 1.

This parameter is used in conjunction with the *minor_version* parameter. Therefore, you must specify both parameters together. For example, you must specify both as either NULL, or non-NULL.

minor_version

An input and output parameter of type INTEGER that indicates the minor document version. On input, this parameter specifies the minor document version that you support for the XML documents that are passed as parameters for this stored procedure (*xml_input*, *xml_output*, and *xml_message*). The stored procedure processes all XML documents in the specified version, or returns an error (-20457) if the version is invalid.

On output, this parameter indicates the highest minor document version that is supported for the highest supported major version. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters. Currently, the highest and only minor document version that is supported is 0 (zero).

This parameter is used in conjunction with the *major_version* parameter. Therefore, you must specify both parameters together. For example, you must specify both as either NULL, or non-NULL.

requested_locale

An input parameter of type VARCHAR(33) that specifies a locale. If the specified language is supported on the server, translated content is returned in the *xml_output* and *xml_message* parameters. Otherwise, content is returned in the default language. Only the language and possibly the territory information is used from the locale. The locale is not used to format numbers or influence the document encoding. For example, key names are not translated. The only translated portion of XML output and XML message documents are Display Name, Display Unit, and Hint. The value might be globalized where

applicable. You should always compare the requested language to the language that is used in the XML output document (see the Document Locale entry in the XML output document).

Currently, the only supported value for *requested_locale* is en_US.

xml_input

Currently, this stored procedure does not accept an XML input document. You must specify NULL for this parameter, or an error (+20458) is raised to indicate that the input is invalid.

xml_filter

An input parameter of type BLOB(4K) in UTF-8 that specifies a valid XPath query string. Use a filter when you want to retrieve a single value from an XML output document. For more information, see “XPath expressions for filtering output” on page 948.

The following example selects the value for the Data Server Product Version from the XML output document:

```
/plist/dict/key[.='Data Server Product Version']/following-sibling::string[1]
```

If the key is not followed by the specified sibling, an error is returned.

xml_output

An output parameter of type BLOB(2G) that returns a complete XML output document of type Data Server Configuration Output in UTF-8. If a filter is specified, this parameter returns a string value. If the stored procedure is unable to return a complete output document (for example, if a processing error occurs that results in an SQL warning or error), this parameter is set to NULL.

An XML output document contains the following key and value pairs followed by the data server configuration information. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
  <dict>
    <key>Document Type Name</key>
    <string>Data Server Configuration Output</string>
    <key>Document Type Major Version</key><integer>1</integer>
    <key>Document Type Minor Version</key><integer>0</integer>
    <key>Data Server Product Name</key><string>DSN</string>
    <key>Data Server Product Version</key><string>9.1.5</string>
    <key>Data Server Major Version</key><integer>9</integer>
    <key>Data Server Minor Version</key><integer>1</integer>
    <key>Data Server Platform</key><string>z/OS</string>
    <key>Document Locale</key><string>en_US</string>
    --- Data server configuration data here ---
  </dict>
</plist>
```

Entries in the XML output document are grouped by using nested dictionaries. Each entry in the XML output document describes a single piece of information. In general, an XML output document is comprised of the Display Name, the Value, and a Hint, as shown in the following example:

```
<key>SQL Domain</key>
<dict>
  <key>Display Name</key>
  <string>SQL Domain</string>
  <key>Value</key>
  <string>v33ec059.svl.ibm.com</string>
  <key>Hint</key>
  <string />
</dict>
```

For a sample XML output document, see Example 1 in the Examples section.

xml_message

An output parameter of type BLOB(64K) that returns a complete XML output document of type Data Server Message in UTF-8 that provides detailed information about an SQL warning condition. This document is returned when a call to the stored procedure results in an SQL warning, and the warning message indicates that additional information is returned in the XML message output document. If the warning message does not indicate that additional information is returned, then this parameter is set to NULL.

An XML message document contains the following key and value pairs followed by details about an SQL warning condition. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server Message</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Data Server Product Name</key><string>DSN</string>
  <key>Data Server Product Version</key><string>9.1.5</string>
  <key>Data Server Major Version</key><integer>9</integer>
  <key>Data Server Minor Version</key><integer>1</integer>
  <key>Data Server Platform</key><string>z/OS</string>
  <key>Document Locale</key><string>en_US</string>
  --- Details about an SQL warning condition are included here. ---
</dict>
</plist>
```

Similar to an XML output document, the details about an SQL warning condition will be encapsulated in a dictionary entry, which is comprised of the Display Name, the Value, and a Hint. For an example of an XML message document, see Example 2.

Examples

Example 1: The following example shows a fragment of an XML output document for the GET_CONFIG stored procedure. The two major sections that the XML output document always contains are "Common Data Sharing Group Information" and "DB2 Subsystem Specific Information." In this example, the ellipsis (. . .) represent a dictionary entry that is comprised of the Display Name, Value, and Hint, such as:

```
<dict>
  <key>Display Name</key>
  <string>DDF Status</string>
  <key>Value</key>
  <string>STARTD</string>
  <key>Hint</key>
  <string />
</dict>
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server Configuration Output</string>
  <key>Document Type Major Version</key>
  <integer>1</integer>
  <key>Document Type Minor Version</key>
  <integer>0</integer>
  <key>Data Server Product Name</key>
  <string>DSN</string>
```

```

<key>Data Server Product Version</key>
<string>9.1.5</string>
<key>Data Server Major Version</key>
<integer>9</integer>
<key>Data Server Minor Version</key>
<integer>1</integer>
<key>Data Server Platform</key>
<string>z/OS</string>
<key>Document Locale</key>
<string>en_US</string>

<key>Common Data Sharing Group Information</key>
<dict>
  <key>Display Name</key>
  <string>Common Data Sharing Group Information</string>
  <key>Data Sharing Group Name</key>
  ...
  <key>Data Sharing Group Level</key>
  ...
  <key>Data Sharing Group Mode</key>
  ...
  <key>Data Sharing Group Protocol Level</key>
  ...
  <key>Data Sharing Group Attach Name</key>
  ...
  <key>SCA Structure Size</key>
  ...
  <key>SCA Status</key>
  ...
  <key>SCA in Use</key>
  ...
  <key>LOCK1 Structure Size</key>
  ...
  <key>Number of Lock Entries</key>
  ...
  <key>Number of List Entries</key>
  ...
  <key>List Entries in Use</key>
  ...
  <key>Hint</key><string></string>
</dict>

<key>DB2 Subsystem Specific Information</key>
<dict>
  <key>Display Name</key>
  <string>DB2 Subsystem Specific Information</string>
  <key>V91A</key>
  <dict>
    <key>Display Name</key>
    <string>V91A</string>
    <key>DB2 Subsystem Status Information</key>
    <dict>
      <key>Display Name</key>
      <string>DB2 Subsystem Status Information</string>
      <key>DB2 Member Identifier</key>
      ...
      <key>DB2 Member Name</key>
      ...
      <key>DB2 Command Prefix</key>
      ...
      <key>DB2 Status</key>
      ...
      <key>DB2 System Level</key>
      ...
      <key>System Name</key>
      ...
      <key>IRLM Subsystem Name</key>
    
```

```

...
<key>IRLM Procedure Name</key>
...
<key>Parallel Coordinator</key>
...
<key>Parallel Assistant</key>
...
<key>Hint</key><string></string>
</dict>

<key>DB2 Subsystem Parameters</key>
<dict>
  <key>Display Name</key>
  <string>DB2 Subsystem Parameters</string>
  <key>DSNHDECP</key>
  <dict>
    <key>Display Name</key>
    <string>DSNHDECP</string>
    <key>AGCCSID</key>
    <dict>
      <key>Display Name</key>
      <string>AGCCSID</string>
      <key>Installation Panel Name</key>
      ...
      <key>Installation Panel Field Name</key>
      ...
      <key>Location on Installation Panel</key>
      ...
      <key>Subsystem Parameter Value</key>
      ...
      <key>Hint</key><string></string>
    </dict>
  </dict>

  --- This is only a fragment of the
        DSNHDECP parameters that are returned
        by the GET_CONFIG stored procedure. ---

  <key>Hint</key><string></string>
</dict>

  --- This is only a fragment of the
        DB2 subsystem parameters that are returned
        by the GET_CONFIG stored procedure. ---

  <key>Hint</key><string></string>
</dict>

<key>DB2 Distributed Access Information</key>
<dict>
  <key>Display Name</key>
  <string>DB2 Distributed Access Information</string>
  <key>DDF Status</key>
  ...
  <key>Location Name</key>
  ...
  <key>LU Name</key>
  ...
  <key>Generic LU Name</key>
  ...
  <key>TCP/IP Port</key>
  ...
  <key>Resynchronization Port</key>
  ...
  <key>IP Address</key>
  ...
  <key>SQL Domain</key>
  ...

```

```

<key>Resynchronization Domain</key>
...
<key>DT - DDF Thread Value</key>
...
<key>CONDBAT - Maximum Inbound Connections</key>
...
<key>MDBAT - Maximum Concurrent Active DBATs</key>
...
<key>ADBAT - Active DBATs</key>
...
<key>QUEDBAT - Times that ADBAT Reached MDBAT Limit</key>
...
<key>INADBAT - Inactive DBATs (Type 1)</key>
...
<key>CONQUED - Queued Connections</key>
...
<key>DSCDBAT - Pooled DBATs</key>
...
<key>INACONN - Inactive Connections (Type 2)</key>
...
<key>Hint</key><string></string>
</dict>

<key>Active Log Data Set Information</key>
<dict>
  <key>Display Name</key>
  <string>Active Log Data Set Information</string>
  <key>Active Log Copy 01</key>
  <dict>
    <key>Display Name</key>
    <string>Active Log Copy 01</string>
    <key>Data Set Name</key>
    ...
    <key>Data Set Volumes</key>
    <dict>
      <key>Display Name</key>
      <string>Data Set Volumes</string>
      <key>Value</key>
      <array>
        <string>CATLGJ</string>
      </array>
      <key>Hint</key><string></string>
    </dict>
    <key>Hint</key><string></string>
  </dict>
  <key>Active Log Copy 02</key>
  <dict>
    --- The format of this dictionary entry is
        the same as that of Active Log Copy 01. ---
  </dict>
  <key>Hint</key><string></string>
</dict>

<key>Time of Last DB2 Restart</key>
...

<key>Resource Limit Facility Information</key>
<dict>
  <key>Display Name</key>
  <string>Resource Limit Facility Information</string>
  <key>RLF Table Names</key>
  <dict>
    <key>Display Name</key>
    <string>RLF Table Names</string>
    <key>Value</key>
    <array>
      <string>SYSADM.DSNRLST01</string>
    </array>
  </dict>

```

```

        </array>
        <key>Hint</key><string></string>
    </dict>
    <key>Hint</key><string></string>
</dict>

    <key>Connected DB2 Subsystem</key>
    ...
    <key>Hint</key><string></string>
</dict>
<key>Hint</key><string></string>
</dict>
<key>Hint</key><string></string>
</dict>
</plist>

```

Example 2: The following example shows a sample XML message document for the GET_CONFIG stored procedure.

```

<?xml version="1.0" encoding="UTF-8" ?>
<plist version="1.0">
<dict>
    <key>Document Type Name</key><string>Data Server Message</string>
    <key>Document Type Major Version</key><integer>1</integer>
    <key>Document Type Minor Version</key><integer>0</integer>
    <key>Data Server Product Name</key><string>DSN</string>
    <key>Data Server Product Version</key><string>9.1.5</string>
    <key>Data Server Major Version</key><integer>9</integer>
    <key>Data Server Minor Version</key><integer>1</integer>
    <key>Data Server Platform</key><string>z/OS</string>
    <key>Document Locale</key><string>en_US</string>
    <key>Short Message Text</key>
    <dict>
        <key>Display Name</key><string>Short Message Text</string>
        <key>Value</key>
        <string>DSNA630I DSNADMGC A PARAMETER FORMAT OR CONTENT ERROR WAS FOUND.
            The XML input document must be empty or NULL.</string>
        <key>Hint</key><string />
    </dict>
    </dict>
</dict>
</plist>

```

Example 3: This example shows a simple and static Java program that calls the GET_CONFIG stored procedure with an XPath that queries the value of the data server's IP address. The XPath is statically created as a string object by the program, and then converted to a BLOB to serve as input for the *xml_filter* parameter. After the stored procedure is called, the *xml_output* parameter contains only a single string and no XML document. This output is materialized as a file called *xml_output.xml* that is in the same directory where the *GetConfDriver* class resides.

```

//*****
// Licensed Materials - Property of IBM
// 5635-DB2
// (C) COPYRIGHT 1982, 2006 IBM Corp. All Rights Reserved.
//
// STATUS = Version 9
//*****
// Source file name: GetConfDriver.java
//
// Sample: How to call SYSPROC.GET_CONFIG with a valid XPath to extract the
// IP Address.
//
//The user runs the program by issuing:
//java GetConfDriver <alias or //server/database> <userid> <password>
//

```

```

//The arguments are:
//<alias> - DB2 subsystem alias for type 2 or //server/database for type 4
// connectivity
//<userid> - user ID to connect as
//<password> - password to connect with
//*****
import java.io.*;
import java.sql.*;
public class GetConfDriver
{

    public static void main (String[] args)
    {
        Connection con = null;
        CallableStatement cstmt = null;
        String driver = "com.ibm.db2.jcc.DB2Driver";
        String url = "jdbc:db2:";
        String userid = null;
        String password = null;

        // Parse arguments
        if (args.length != 3)
        {
            System.err.println("Usage: GetConfDriver <alias or //server/database>
<userid> <password>");
            System.err.println("where <alias or //server/database> is DB2
subsystem alias or //server/database for type 4 connectivity");
            System.err.println(" <userid> is user ID to connect as");
            System.err.println(" <password> is password to connect with");
            return;
        }
        url += args[0];
        userid = args[1];
        password = args[2];

        try {

            byte[] xml_input;
            String str_xmlfilter = new String(
                "/plist/dict/key[.='DB2 Subsystem Specific Information']/following-
sibling::dict[1]" +
                "/key[.='V91A']/following-sibling::dict[1]" +
                "/key[.='DB2 Distributed Access Information']/following-sibling::dict[1]" +
                "/key[.='IP Address']/following-sibling::dict[1]" +
                "/key[.='Value']/following-sibling::string[1]");

            /* Convert XML_FILTER to byte array to pass as BLOB */
            byte[] xml_filter = str_xmlfilter.getBytes("UTF-8");

            // Load the DB2 Universal JDBC Driver
            Class.forName(driver);

            // Connect to database
            con = DriverManager.getConnection(url, userid, password);
            con.setAutoCommit(false);

            cstmt = con.prepareCall("CALL SYSPROC.GET_CONFIG(?,?,?,?);");

            // Major / Minor Version / Requested Locale
            cstmt.setInt(1, 1);
            cstmt.setInt(2, 0);
            cstmt.setString(3, "en_US");
            // No Input document
            cstmt.setObject(4, null, Types.BLOB);
            cstmt.setObject(5, xml_filter, Types.BLOB);
        }
    }
}

```

```

// Output Params
cstmt.registerOutParameter(1, Types.INTEGER);
cstmt.registerOutParameter(2, Types.INTEGER);
cstmt.registerOutParameter(6, Types.BLOB);
cstmt.registerOutParameter(7, Types.BLOB);

cstmt.execute();
con.commit();

SQLWarning ctstmt_warning = cstmt.getWarnings();
if (ctstmt_warning != null) {
    System.out.println("SQL Warning: " + ctstmt_warning.getMessage());
}
else {
    System.out.println("SQL Warning: None\r\n");
}

System.out.println("Major Version returned " + cstmt.getInt(1) );
System.out.println("Minor Version returned " + cstmt.getInt(2) );

/* get output BLOBs */
Blob b_out = cstmt.getBlob(6);

if(b_out != null)
{
    int out_length = (int)b_out.length();
    byte[] bxml_output = new byte[out_length];

    /* open an inputstream on BLOB data */
    InputStream instr_out = b_out.getBinaryStream();

    /* copy from inputstream into byte array */
    int out_len = instr_out.read(bxml_output, 0, out_length);

    /* write byte array into FileOutputStream */
    FileOutputStream fxml_out = new FileOutputStream("xml_output.xml");

    /* write byte array content into FileOutputStream */
    fxml_out.write(bxml_output, 0, out_length );

    //Close streams
    instr_out.close();
    fxml_out.close();
}

Blob b_msg = cstmt.getBlob(7);
if(b_msg != null)
{
    int msg_length = (int)b_msg.length();
    byte[] bxml_message = new byte[msg_length];

    /* open an inputstream on BLOB data */
    InputStream instr_msg = b_msg.getBinaryStream();

    /* copy from inputstream into byte array */
    int msg_len = instr_msg.read(bxml_message, 0, msg_length);

    /* write byte array content into FileOutputStream */
    FileOutputStream fxml_msg = new FileOutputStream(new File
("xml_message.xml"));
    fxml_msg.write(bxml_message, 0, msg_length);

    //Close streams
    instr_msg.close();
    fxml_msg.close();
}
}

```

```

catch (SQLException sqle) {
    System.out.println("Error during CALL "
        + " SQLSTATE = " + sqle.getSQLState()
        + " SQLCODE = " + sqle.getErrorCode()
        + " : " + sqle.getMessage());
}
catch (Exception e) {
    System.out.println("Internal Error " + e.toString());
}
finally
{
    if(cstmt != null)
        try { cstmt.close(); } catch ( SQLException sqle)
    { sqle.printStackTrace(); }
    if(con != null)
        try { con.close(); } catch ( SQLException sqle)
    { sqle.printStackTrace(); }
}
}
}

```

GET_MESSAGE stored procedure

The GET_MESSAGE stored procedure returns the short message text for an SQLCODE.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have EXECUTE privilege on the GET_MESSAGE stored procedure.

Syntax

The following syntax diagram shows the SQL CALL statement for invoking this stored procedure:

<pre> ▶▶—CALL—GET_MESSAGE—(—major_version—,—minor_version—,——————▶ ▶—requested_locale—,—xml_input—,—xml_filter—,—xml_output—,—xml_message—)————▶▶ </pre>
--

The schema is SYSPROC.

Option descriptions

major_version

An input and output parameter of type INTEGER that indicates the major document version. On input, this parameter indicates the major document version that you support for the XML documents that are passed as parameters in the stored procedure (*xml_input*, *xml_output*, and *xml_message*). The stored procedure processes all XML documents in the specified version, or returns an error (-20457) if the version is invalid.

On output, this parameter specifies the highest major document version that is supported by the stored procedure. To determine the highest supported

document version, specify NULL for this input parameter and all other required parameters. Currently, the highest and only major document version that is supported is 1.

If the XML document in the *xml_input* parameter specifies the Document Type Major Version key, the value for that key must be equal to the value provided in the *major_version* parameter, or an error (+20458) is raised.

This parameter is used in conjunction with the *minor_version* parameter. Therefore, you must specify both parameters together. For example, you must specify both as either NULL, or non-NULL.

minor_version

An input and output parameter of type INTEGER that indicates the minor document version. On input, this parameter specifies the minor document version that you support for the XML documents that are passed as parameters for this stored procedure (*xml_input*, *xml_output*, and *xml_message*). The stored procedure processes all XML documents in the specified version, or returns an error (-20457) if the version is invalid.

On output, this parameter indicates the highest minor document version that is supported for the highest supported major version. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters. Currently, the highest and only minor document version that is supported is 0 (zero).

If the XML document in the *xml_input* parameter specifies the Document Type Minor Version key, the value for that key must be equal to the value provided in the *minor_version* parameter, or an error (+20458) is raised.

This parameter is used in conjunction with the *major_version* parameter. Therefore, you must specify both parameters together. For example, you must specify both as either NULL, or non-NULL.

requested_locale

An input parameter of type VARCHAR(33) that specifies a locale. If the specified language is supported on the server, translated content is returned in the *xml_output* and *xml_message* parameters. Otherwise, content is returned in the default language. Only the language and possibly the territory information is used from the locale. The locale is not used to format numbers or influence the document encoding. For example, key names are not translated. The only translated portion of the XML output and XML message documents are Display Name, Display Unit, and Hint. The value might be globalized where applicable. You should always compare the requested language to the language that is used in the XML output document (see the Document Locale entry in the XML output document).

Currently, the only supported value for *requested_locale* is en_US.

xml_input

An input parameter of type BLOB(2G) that specifies an XML input document of type Data Server Message Input in UTF-8 that contains input values for the stored procedure.

The general structure of an XML input document is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key><string>Data Server Message Input</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Document Locale</key><string>en_US</string>
```

```

<key>Complete</key><false/>
<key>Required Parameters</key>
<dict>
  <key>SQLCODE</key>
  <dict>
    <key>Value</key><integer>sqlcode</integer>
  </dict>
</dict>
<key>Optional Parameters</key>
<dict>
  <key>Message Tokens</key>
  <dict>
    <key>Value</key>
    <array>
      <string>token1 in SQLCA</string>
      <string>token2 in SQLCA</string>
    </array>
  </dict>
</dict>
</dict>
</plist>

```

You can omit the Complete key entry. Omitting the Complete key is the same as setting the value to false, which means that the stored procedure will not run in *Complete mode*. If the Complete key is included and you set the value to true, the stored procedure will run in Complete mode, and all other entries in the XML input document will be ignored.

If the stored procedure runs in Complete mode, a complete input document is returned by the *xml_output* parameter of the stored procedure. The returned XML document is a full XML input document that includes a Document Type and sections for all possible required and optional parameters. The returned XML input document also includes entries for Display Names, Hints, and the Document Locale. Although these entries are not required (and will be ignored) in the XML input document, they are usually needed when rendering the document in a client application. All entries in the returned XML input document can be rendered and changed in ways that are independent of the operating system or data server. Subsequently, the modified XML input document can be passed in the *xml_input* parameter in a new call to the same stored procedure. This enables you to programmatically create valid *xml_input* documents.

The following example shows the minimal XML input document that is required for the stored procedure to run in Complete mode. If you specify XML elements in addition to the ones shown, those entries are ignored.

```

<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Complete</key><true/>
</dict>
</plist>

```

For an example of an XML input document that is returned by the *xml_output* parameter when the stored procedure is running in Complete mode, see Example 1 in the Examples section.

If the stored procedure is not running in Complete mode, you must specify the Document Type Name key, the required parameters, and any optional parameters that you want to specify. Specifying the Document Type Major Version and Document Type Minor Version keys are optional. If you specify the Document Type Major Version and Document Type Minor Version keys, the values must be the same as the values that you specified in the *major_version* and *minor_version* parameters. You must either specify both or

omit both of the Document Type Major Version and Document Type Minor Version keys. Specifying the Document Locale key is optional. If you specify the Document Locale key, the value is ignored.

For an example of an XML input document that will not run in Complete mode, see Example 2 in the Examples section.

xml_filter

An input parameter of type BLOB(4K) in UTF-8 that specifies a valid XPath query string. Use a filter when you want to retrieve a single value from an XML output document. For more information, see “XPath expressions for filtering output” on page 948.

The following example selects the value for the short message text from the XML output document:

```
/plist/dict/key[.='Short Message Text']/following-sibling::dict[1]/key[.='Value']/following-sibling::string[1]
```

If the key is not followed by the specified sibling, an error is returned.

xml_output

An output parameter of type BLOB(2G) that returns a complete XML output document of type Data Server Message Output in UTF-8. If a filter is specified, this parameter returns a string value. If the stored procedure is unable to return a complete output document (for example, if a processing error occurs that results in an SQL warning or error), this parameter is set to NULL.

The XML output document contains the following key and value pairs followed by the short message text for an SQLCODE:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server Message Output</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Data Server Product Name</key><string>DSN</string>
  <key>Data Server Product Version</key><string>9.1.5</string>
  <key>Data Server Major Version</key><integer>9</integer>
  <key>Data Server Minor Version</key><integer>1</integer>
  <key>Data Server Platform</key><string>z/OS</string>
  <key>Document Locale</key><string>en_US</string>
  --- Message text for an SQLCODE here ---
</dict>
</plist>
```

The short message text for an SQLCODE will be encapsulated in a dictionary entry, which is comprised of the Display Name, the Value, and a Hint.

For an example of an XML output document, see Example 3 in the Examples section.

When the stored procedure runs in Complete mode, this parameter returns an XML document that you can modify and pass back to the stored procedure as the *xml_input* parameter. This approach provides a programmatic way to create valid XML input documents.

xml_message

An output parameter of type BLOB(64K) that returns a complete XML output document of type Data Server Message in UTF-8 that provides detailed information about an SQL warning condition. This document is returned when a call to the procedure results in an SQL warning, and the warning message

indicates that additional information is returned in the XML message output document. If the warning message does not indicate that additional information is returned, then this parameter is set to NULL.

An XML message document contains the following key and value pairs followed by details about an SQL warning condition. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server Message</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Data Server Product Name</key><string>DSN</string>
  <key>Data Server Product Version</key><string>9.1.5</string>
  <key>Data Server Major Version</key><integer>9</integer>
  <key>Data Server Minor Version</key><integer>1</integer>
  <key>Data Server Platform</key><string>z/OS</string>
  <key>Document Locale</key><string>en_US</string>
  --- Details about an SQL warning condition are included here. ---
</dict>
</plist>
```

Similar to an XML output document, the details about an SQL warning condition will be encapsulated in a dictionary entry, which is comprised of the Display Name, the Value, and a Hint. For an example of an XML message document, see Example 4.

Example

Example 1: The following example shows an XML input document that is returned by the *xml_output* parameter when the stored procedure is running in Complete mode.

```
<?xml version="1.0" encoding="UTF-8" ?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server Message Input</string>
  <key>Document Type Major Version</key>
  <integer>1</integer>
  <key>Document Type Minor Version</key>
  <integer>0</integer>
  <key>Document Locale</key>
  <string>en_US</string>
  <key>Required Parameters</key>
  <dict>
    <key>Display Name</key>
    <string>Required Parameters</string>
    <key>SQLCODE</key>
    <dict>
      <key>Display Name</key>
      <string>SQLCODE</string>
      <key>Value</key>
      <integer />
      <key>Hint</key>
      <string />
    </dict>
    <key>Hint</key>
    <string />
  </dict>
  <key>Optional Parameters</key>
  <dict>
    <key>Display Name</key>
    <string>Optional Parameters</string>
```

```

    <key>Message Tokens</key>
    <dict>
      <key>Display Name</key>
      <string>Message Tokens</string>
      <key>Value</key>
      <array>
        <string />
      </array>
      <key>Hint</key>
      <string />
    </dict>
  <key>Hint</key>
  <string />
</dict>
</dict>
</plist>

```

Example 2: The following example shows the format of a sample XML input document for the GET_MESSAGE stored procedure that contains an SQLCODE.

```

<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server Message Input</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Document Locale</key><string>en_US</string>
  <key>Required Parameters</key>
  <dict>
    <key>SQLCODE</key>
    <dict>
      <key>Value</key><integer>-104</integer>
    </dict>
  </dict>
  <key>Optional Parameters</key>
  <dict>
    <key>Message Tokens</key>
    <dict>
      <key>Value</key>
      <array>
        <string>X</string>
        <string>( . LIKE AS</string>
      </array>
    </dict>
  </dict>
</dict>
</plist>

```

Example 3: The following example shows the format of a sample XML output document for the GET_MESSAGE stored procedure.

```

<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server Message Output</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Data Server Product Name</key><string>DSN</string>
  <key>Data Server Product Version</key><string>9.1.5</string>
  <key>Data Server Major Version</key><integer>9</integer>
  <key>Data Server Minor Version</key><integer>1</integer>
  <key>Data Server Platform</key><string>z/OS</string>
  <key>Document Locale</key><string>en_US</string>

  <key>Short Message Text</key>

```

```

    <dict>
      <key>Display Name</key><string>Short Message Text</string>
      <key>Hint</key><string />
    </dict>
  </dict>
</plist>

```

Example 4: The following example shows a sample XML message document for the GET_MESSAGE stored procedure.

```

<?xml version="1.0" encoding="UTF-8" ?>
<plist version="1.0">
  <dict>
    <key>Document Type Name</key><string>Data Server Message</string>
    <key>Document Type Major Version</key><integer>1</integer>
    <key>Document Type Minor Version</key><integer>0</integer>
    <key>Data Server Product Name</key><string>DSN</string>
    <key>Data Server Product Version</key><string>9.1.5</string>
    <key>Data Server Major Version</key><integer>9</integer>
    <key>Data Server Minor Version</key><integer>1</integer>
    <key>Data Server Platform</key><string>z/OS</string>
    <key>Document Locale</key><string>en_US</string>
    <key>Short Message Text</key>
    <dict>
      <key>Display Name</key><string>Short Message Text</string>
      <key>Value</key>
      <string>DSNA630I DSNADMGM A PARAMETER FORMAT OR CONTENT ERROR WAS FOUND.
        The value for key 'Document Type Minor Version' is '2'. It does
        not match the value '0', which was specified for parameter 2 of
        the stored procedure. Both values must be equal.</string>
      <key>Hint</key><string />
    </dict>
  </dict>
</plist>

```

Example 5: This example shows a simple and static Java program that calls the GET_MESSAGE stored procedure with an XML input document and an XPath that queries the short message text of an SQLCODE.

The XML input document is initially saved as a file called xml_input.xml that is in the same directory where the GetMessageDriver class resides. This sample program uses the following xml_input.xml file:

```

<?xml version="1.0" encoding="UTF-8" ?>
<plist version="1.0">
  <dict>
    <key>Document Type Name</key>
    <string>Data Server Message Input</string>
    <key>Document Type Major Version</key>
    <integer>1</integer>
    <key>Document Type Minor Version</key>
    <integer>0</integer>
    <key>Document Locale</key>
    <string>en_US</string>
    <key>Complete</key>
    <false />
    <key>Required Parameters</key>
    <dict>
      <key>SQLCODE</key>
      <dict>
        <key>Value</key>
        <integer>-204</integer>
      </dict>
    </dict>
  </dict>
  <key>Optional Parameters</key>

```

```

<dict>
  <key>Message Tokens</key>
  <dict>
    <key>Value</key>
    <array>
      <string>SYSIBM.DDF_CONFIG</string>
    </array>
  </dict>
</dict>
</dict>
</plist>

```

The XPath is statically created as a string object by the program and then converted to a BLOB to serve as input for the *xml_filter* parameter. After the stored procedure is called, the *xml_output* parameter contains only a single string and no XML document. This output is materialized as a file called *xml_output.xml* that is in the same directory where the *GetMessageDriver* class resides.

Sample invocation of the GET_MESSAGE stored procedure with a valid XML input document and a valid XPath:

```

//*****
// Licensed Materials - Property of IBM
// 5635-DB2
// (C) COPYRIGHT 1982, 2006 IBM Corp. All Rights Reserved.
//
// STATUS = Version 9
//*****
// Source file name: GetSystemDriver.java
//
// Sample: How to call SYSPROC.GET_SYSTEM_INFO with a valid XML input document
// and a valid XPath to extract the operating system name and release.
//
// The user runs the program by issuing:
// java GetSystemDriver <alias or //server/database> <userid> <password>
//
// The arguments are:
// <alias> - DB2 subsystem alias for type 2 or //server/database for type 4
// connectivity
// <userid> - user ID to connect as
// <password> - password to connect with
//*****
import java.io.*;
import java.sql.*;

public class GetSystemDriver
{

    public static void main (String[] args)
    {
        Connection con = null;
        CallableStatement cstmt = null;
        String driver = "com.ibm.db2.jcc.DB2Driver";
        String url = "jdbc:db2:";
        String userid = null;
        String password = null;

        // Parse arguments
        if (args.length != 3)
        {
            System.err.println("Usage: GetSystemDriver <alias or //server/database>
<userid> <password>");
            System.err.println("where <alias or //server/database> is DB2 subsystem
alias or //server/database for type 4 connectivity");
            System.err.println(" <userid> is user ID to connect as");
            System.err.println(" <password> is password to connect with");

```

```

        return;
    }
    url += args[0];
    userid = args[1];
    password = args[2];

    try {

        String str_xmlfilter = new String(
            "/plist/dict/key[.='Operating System Information']/following-sibling::
dict[1]" +
            "/key[.='Name and Release']/following-sibling::dict[1]" +
            "/key[.='Value']/following-sibling::string[1]");

        // Convert XML_FILTER to byte array to pass as BLOB
        byte[] xml_filter = str_xmlfilter.getBytes("UTF-8");

        // Read XML_INPUT from file
        File fptr = new File("xml_input.xml");

        int file_length = (int)fptr.length();
        byte[] xml_input = new byte[file_length];

        FileInputStream instream = new FileInputStream(fptr);
        int tot_bytes = instream.read(xml_input,0, xml_input.length);
        if (tot_bytes == -1) {
            System.out.println("Error during file read");
            return;
        }
        instream.close();

        // Load the DB2 Universal JDBC Driver
        Class.forName(driver);

        // Connect to database
        con = DriverManager.getConnection(url, userid, password);
        con.setAutoCommit(false);

        cstmt = con.prepareStatement("CALL SYSPROC.GET_SYSTEM_INFO(?,?,?,?,?,?)");

        // Major / Minor Version / Requested Locale
        cstmt.setInt(1, 1);
        cstmt.setInt(2, 1);
        cstmt.setString(3, "en_US");

        // Input documents
        cstmt.setObject(4, xml_input, Types.BLOB);
        cstmt.setObject(5, xml_filter, Types.BLOB);

        // Output Params
        cstmt.registerOutParameter(1, Types.INTEGER);
        cstmt.registerOutParameter(2, Types.INTEGER);
        cstmt.registerOutParameter(6, Types.BLOB);
        cstmt.registerOutParameter(7, Types.BLOB);

        cstmt.execute();
        con.commit();

        SQLWarning ctstmt_warning = cstmt.getWarnings();
        if (ctstmt_warning != null) {
            System.out.println("SQL Warning: " + ctstmt_warning.getMessage());
        }
        else {
            System.out.println("SQL Warning: None\r\n");
        }

        System.out.println("Major Version returned " + cstmt.getInt(1) );
    }
}

```

```

System.out.println("Minor Version returned " + cstmt.getInt(2) );

// Get output BLOBs
Blob b_out = cstmt.getBlob(6);

if(b_out != null)
{
    int out_length = (int)b_out.length();
    byte[] bxml_output = new byte[out_length];

    // Open an inputstream on BLOB data
    InputStream instr_out = b_out.getBinaryStream();

    // Copy from inputstream into byte array
    int out_len = instr_out.read(bxml_output, 0, out_length);

    // Write byte array content into FileOutputStream
    FileOutputStream fxml_out = new FileOutputStream("xml_output.xml");
    fxml_out.write(bxml_output, 0, out_length );

    //Close streams
    instr_out.close();
    fxml_out.close();
}

Blob b_msg = cstmt.getBlob(7);

if(b_msg != null)
{
    int msg_length = (int)b_msg.length();
    byte[] bxml_message = new byte[msg_length];

    // Open an inputstream on BLOB data
    InputStream instr_msg = b_msg.getBinaryStream();

    // Copy from inputstream into byte array
    int msg_len = instr_msg.read(bxml_message, 0, msg_length);

    // Write byte array content into FileOutputStream
    FileOutputStream fxml_msg = new FileOutputStream(new File
("xml_message.xml"));
    fxml_msg.write(bxml_message, 0, msg_length);

    //Close streams
    instr_msg.close();
    fxml_msg.close();
}

catch (SQLException sqle) {
    System.out.println("Error during CALL "
        + " SQLSTATE = " + sqle.getSQLState()
        + " SQLCODE = " + sqle.getErrorCode()
        + " : " + sqle.getMessage());
}

catch (Exception e) {
    System.out.println("Internal Error " + e.toString());
}

finally
{
    if(cstmt != null)
        try { cstmt.close(); } catch ( SQLException sqle)
    { sqle.printStackTrace(); }
    if(con != null)
        try { con.close(); } catch ( SQLException sqle)

```

```

    { sqlc.printStackTrace(); }
  }
}

```

GET_SYSTEM_INFO stored procedure

The GET_SYSTEM_INFO stored procedure returns system information about the data server.

This system information includes:

- Operating system information
- Product information
- DB2 MEPL
- SYSMOD APPLY status
- Workload Manager (WLM) classification rules that apply to DB2 Workload for subsystem types DB2 and DDF

Environment

The load module for the GET_SYSTEM_INFO stored procedure, DSNADMGS, must reside in an APF-authorized library. The GET_SYSTEM_INFO stored procedure runs in a WLM-established stored procedures address space, and all of the libraries that are specified in the STEPLIB DD statement must be APF-authorized. TCB=1 is also required.

Authorization

To execute the CALL statement, the owner of the package or plan that contains the CALL statement must have EXECUTE privilege on the GET_SYSTEM_INFO stored procedure.

In addition, because the GET_SYSTEM_INFO stored procedure queries the SMPCSI data set for the status of the SYSMODs, the authorization ID that is associated with the stored procedure address space where the GET_SYSTEM_INFO stored procedure is running must have at least RACF read authority to the SMPCSI data set.

Syntax

```

CALL GET_SYSTEM_INFO (—major_version—, —minor_version—, —
requested_locale—, —xml_input—, —xml_filter—, —xml_output—, —xml_message—)

```

The schema is SYSPROC.

Option descriptions

major_version

An input and output parameter of type INTEGER that indicates the major document version. On input, this parameter indicates the major document version that you support for the XML documents passed as parameters in the

stored procedure (*xml_input*, *xml_output*, and *xml_message*). The stored procedure processes all XML documents in the specified version, or returns an error (-20457) if the version is invalid.

On output, this parameter specifies the highest major document version that is supported by the stored procedure. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters. Currently, the highest and the only major document version that is supported is 1.

If the XML document in the *xml_input* parameter specifies a Document Type Major Version key, the value for that key must be equal to the value that is provided in the *major_version* parameter, or an error (+20458) is raised.

This parameter is used in conjunction with the *minor_version* parameter. Therefore, you must specify both parameters together. For example, you must specify both as either NULL, or non-NULL.

minor_version

An input and output parameter of type INTEGER that indicates the minor document version. On input, this parameter specifies the minor document version that you support for the XML documents passed as parameters for this stored procedure (*xml_input*, *xml_output*, and *xml_message*). The stored procedure processes all XML documents in the specified version, or returns an error (-20457) if the version is invalid.

On output, this parameter indicates the highest minor document version that is supported for the highest supported major version. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters. The highest minor document version that is supported is 1. Minor document version 0 (zero) is also supported.

If the XML document in the *xml_input* parameter specifies a Document Type Minor Version key, the value for that key must be equal to the value that is provided in the *minor_version* parameter, or an error (+20458) is raised.

This parameter is used in conjunction with the *major_version* parameter. Therefore, you must specify both parameters together. For example, you must specify both as either NULL, or non-NULL.

requested_locale

An input parameter of type VARCHAR(33) that specifies a locale. If the specified language is supported on the server, translated content is returned in the *xml_output* and *xml_message* parameters. Otherwise, content is returned in the default language. Only the language and possibly the territory information is used from the locale. The locale is not used to format numbers or influence the document encoding. For example, key names are not translated. The only translated portion of the XML output and XML message documents are Display Name, Display Unit, and Hint. The value might be globalized where applicable. You should always compare the requested language to the language that is used in the XML output document (see the Document Locale entry in the XML output document).

Currently, the only supported value for *requested_locale* is en_US.

xml_input

An input parameter of type BLOB(2G) that specifies an XML input document of type Data Server System Input in UTF-8 that contains input values for the stored procedure.

This XML input document is optional. If the XML input document is not passed to the stored procedure, the stored procedure returns the following information by default:

- Operating system information
- Product information
- DB2 MEPL
- Workload Manager (WLM) classification rules for DB2 Workload

This stored procedure supports two types of XML input documents, Version 1.0 or Version 1.1.

For Version 1.0, the general structure of an XML input document is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key><string>Data Server System Input</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Document Locale</key><string>en_US</string>
  <key>Complete</key><false/>
  <key>Optional Parameters</key>
  <dict>
    <key>SMPCSI Data Set</key>
    <dict>
      <key>Value</key><string>SMPCSI data set name</string>
    </dict>
  </dict>
  <key>SYSMOD</key>
  <dict>
    <key>Value</key>
    <array>
      <string>SYSMOD number</string>
      <string>SYSMOD number</string>
    </array>
  </dict>
</dict>
</plist>
```

For Version 1.1, the general structure of an XML input document is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key><string>Data Server System Input</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>1</integer>
  <key>Document Locale</key><string>en_US</string>
  <key>Complete</key><false/>
  <key>Optional Parameters</key>
  <dict>
    <key>Include</key>
    <dict>
      <key>Value</key>
      <array>
        <string>Operating System Information</string>
        <string>Product Information</string>
        <string>DB2 MEPL</string>
        <string>Workload Manager (WLM) Classification Rules for
          DB2 Workload</string>
      </array>
    </dict>
  </dict>
  <key>SMPCSI Data Set</key>
  <dict>
    <key>Value</key><string>SMPCSI data set name</string>
  </dict>
</plist>
```

```

    <key>SYSMOD</key>
    <dict>
      <key>Value</key>
      <array>
        <string>SYSMOD number</string>
        <string>SYSMOD number</string>
      </array>
    </dict>
  </dict>
</dict>
</plist>

```

You can omit the Complete key entry. Omitting the Complete key is the same as setting the value to false, which means that the stored procedure will not run in *Complete mode*. If the Complete key is included and you set the value to true, the stored procedure will run in Complete mode, and all other entries in the XML input document will be ignored.

If the stored procedure runs in Complete mode, a complete input document is returned by the *xml_output* parameter of the stored procedure. The returned XML document is a full XML input document that includes a Document Type and a section for all possible optional parameters. The returned XML input document also includes entries for Display Names, Hints, and the Document Locale. Although these entries are not required (and will be ignored) in the XML input document, they are usually needed when rendering the document in a client application. All entries in the returned XML input document can be rendered and changed in ways that are independent of the operating system and data server. Subsequently, the modified XML input document can be passed in the *xml_input* parameter in a new call to the same stored procedure. This enables you to programmatically create valid *xml_input* documents.

The following example shows the minimal XML input document that is required for the stored procedure to run in Complete mode. If you specify XML elements in addition to the ones shown, those entries are ignored.

```

<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
  <dict>
    <key>Complete</key><true/>
  </dict>
</plist>

```

For examples of Version 1.0 and Version 1.1 XML input documents that are returned by the *xml_output* parameter when the stored procedure is running in Complete mode, see Example 1 and Example 2 respectively.

Version 1.0: To use Version 1.0 of the XML input document you must specify the *major_version* parameter as 1 and the *minor_version* parameter as 0 (zero). You must also specify the Document Type Name key, the SMPCSI data set, and the list of SYSMODs. Specifying the Document Type Major Version and Document Type Minor Version keys is optional. If you specify the Document Type Major Version and Document Type Minor Version keys, the values must be the same as the values that you specified for the *major_version* and *minor_version* parameters. You either must specify both or omit both of the Document Type Major Version and Document Type Minor Version keys. Specifying the Document Locale key is optional. If you specify the Document Locale key, the value is ignored.

When a Version 1.0 XML input document is passed to the stored procedure, the stored procedure returns the following information in a Version 1.0 XML output document:

- Operating system information

- Product information
- DB2 MEPL
- SYSMOD status (APPLY status for the SYSMODs that are listed in the XML input document)
- Workload Manager (WLM) classification rules for DB2 Workload

For an example of a complete Version 1.0 XML input document for the GET_SYSTEM_INFO stored procedure, see Example 3 in the Examples section.

Version 1.1: A Version 1.1 XML input document supports the **Include** parameter, in addition to the **SMPCSI Data Set** and **SYSMOD** parameters that are supported by a Version 1.0 XML input document.

You can use the Version 1.1 XML input document in the following ways:

- To specify which items to include in the XML output document by specifying these items in the Include array
- To specify the SMPCSI data set and list of SYSMODs so that the stored procedure returns their APPLY status

To use Version 1.1 of the XML input document, you must specify the *major_version* parameter as 1 and the *minor_version* parameter as 1. You must also specify the Document Type Name key, and at least one of the following parameters:

- **Include**
- **SMPCSI Data Set** and **SYSMOD**

Specifying the Document Type Major Version and Document Type Minor Version keys is optional. If you specify the Document Type Major Version and Document Type Minor Version keys, the values must be the same as the values that you specified for the *major_version* and *minor_version* parameters. You either must specify both or omit both of the Document Type Major Version and Document Type Minor Version keys. Specifying the Document Locale key is optional. If you specify the Document Locale key, the value is ignored.

If you pass a Version 1.1 XML input document to the stored procedure, the stored procedure returns the information in a Version 1.1 XML output document.

If you pass a Version 1.1 XML input document to the stored procedure and specify the **Include**, **SMPCSI Data Set**, and **SYSMOD** parameters, the stored procedure will return the items that you specified in the Include array, and the SYSMOD status of the SYSMODs that you specified in the SYSMOD array.

If you pass a Version 1.1 XML input document to the stored procedure and specify the **Include** parameter only, the stored procedure will return only the items that you specified in the Include array.

If you pass a Version 1.1 XML input document to the stored procedure and specify only the **SMPCSI Data Set** and **SYSMOD** parameters, the stored procedure returns the following information in a Version 1.1 XML output document:

- Operating system information
- Product information
- DB2 MEPL
- SYSMOD status (APPLY status for the SYSMODs that are listed in the XML input document)
- Workload Manager (WLM) classification rules for DB2 Workload

For an example of a complete Version 1.1 XML input document for the GET_SYSTEM_INFO stored procedure, see Example 4 in the Examples section.

xml_filter

An input parameter of type BLOB(4K) in UTF-8 that specifies a valid XPath query string. Use a filter when you want to retrieve a single value from an XML output document. For more information, see “XPath expressions for filtering output” on page 948.

The following example selects the value for the Data Server Product Version from the XML output document:

```
/plist/dict/key[.='Data Server Product Version']/following-sibling::string[1]
```

If the key is not followed by the specified sibling, an error is returned.

xml_output

An output parameter of type BLOB(2G) that returns a complete XML output document of type Data Server System Output in UTF-8. If a filter is specified, this parameter returns a string value. If the stored procedure is unable to return a complete output document (for example, if a processing error occurs that results in an SQL warning or error), this parameter is set to NULL.

The *xml_output* parameter can return either a Version 1.0 or Version 1.1 XML output document depending on the *major_version* and *minor_version* parameters that you specify. A Version 1.0 XML output document is returned if you specify the *major_version* as 1 and *minor_version* as 0 (zero). A Version 1.1 XML output document is returned if you specify the *major_version* as 1 and *minor_version* as 1. For more information about the content differences between the Version 1.0 and Version 1.1 XML output documents, see the option description for the *xml_input* parameter.

A complete XML output document provides the following system information:

- Operating system information
- Product information
- DB2 MEPL
- The APPLY status of SYSMODs
- Workload Manager (WLM) classification rules for DB2 Workload for subsystem types DB2 and DDF

The following example shows the general format of a Version 1.0 XML output document. The format of a Version 1.1 XML output document is the same, except that the value of the Document Type Minor Version key is 1. An XML output document contains the following key and value pairs followed by the data server configuration system information:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server System Output</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Data Server Product Name</key><string>DSN</string>
  <key>Data Server Product Version</key><string>9.1.5</string>
  <key>Data Server Major Version</key><integer>9</integer>
  <key>Data Server Minor Version</key><integer>1</integer>
  <key>Data Server Platform</key><string>z/OS</string>
  <key>Document Locale</key><string>en_US</string>
  --- Data server system data here ---
</dict>
</plist>
```

Entries in the XML output document are grouped by using nested dictionaries. Each entry in the XML output document describes a single piece of information. In general, the XML output document is comprised of the Display Name, the Value, and a Hint, as shown in the following example:

```
<key>DFSMS Release</key>
<dict>
  <key>Display Name</key><string>DFSMS Release</string>
  <key>Value</key><string>z/OS 1.7.0</string>
  <key>Hint</key><string />
</dict>
```

For an example of an XML output document, see Example 5 in the Examples section.

xml_message

An output parameter of type BLOB(64K) that returns a complete XML output document of type Data Server Message in UTF-8 that provides detailed information about a SQL warning condition. This document is returned when a call to the stored procedure results in a SQL warning, and the warning message indicates that additional information is returned in the XML message output document. If the warning message does not indicate that additional information is returned, then this parameter is set to NULL.

The *xml_message* parameter can return either a Version 1.0 or Version 1.1 XML message document, but the format of these documents is similar. An XML message document contains the following key and value pairs followed by details about an SQL warning condition. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server Message</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Data Server Product Name</key><string>DSN</string>
  <key>Data Server Product Version</key><string>9.1.5</string>
  <key>Data Server Major Version</key><integer>9</integer>
  <key>Data Server Minor Version</key><integer>1</integer>
  <key>Data Server Platform</key><string>z/OS</string>
  <key>Document Locale</key><string>en_US</string>
  --- Details about an SQL warning condition are included here. ---
</dict>
</plist>
```

Similar to an XML output document, the details about an SQL warning condition will be encapsulated in a dictionary entry, which is comprised of the Display Name, the Value, and a Hint. For an example of an XML message document, see Example 6.

Examples

Example 1: The following example shows a Version 1.0 XML input document that is returned by the *xml_output* parameter when the stored procedure is running in Complete mode.

```
<?xml version="1.0" encoding="UTF-8" ?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key><string>Data Server System Input</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Document Locale</key><string>en_US</string>
  <key>Optional Parameters</key>
```

```

<dict>
  <key>Display Name</key><string>Optional Parameters</string>
  <key>SMPCSI Data Set</key>
  <dict>
    <key>Display Name</key><string>SMPCSI Data Set</string>
    <key>Value</key><string />
    <key>Hint</key><string />
  </dict>
  <key>SYSMOD</key>
  <dict>
    <key>Display Name</key><string>SYSMOD</string>
    <key>Value</key>
    <array>
      <string />
    </array>
    <key>Hint</key><string />
  </dict>
  <key>Hint</key><string />
</dict>
</plist>

```

Example 2: The following example shows a Version 1.1 XML input document that is returned by the *xml_output* parameter when the stored procedure is running in Complete mode.

```

<?xml version="1.0" encoding="UTF-8" ?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key><string>Data Server System Input</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>1</integer>
  <key>Document Locale</key><string>en_US</string>
  <key>Optional Parameters</key>
  <dict>
    <key>Display Name</key><string>Optional Parameters</string>
    <key>Include</key>
    <dict>
      <key>Display Name</key><string>Include</string>
      <key>Value</key>
      <array>
        <string>Operating System Information</string>
        <string>Product Information</string>
        <string>DB2 MEPL</string>
        <string>Workload Manager (WLM) Classification Rules for
          DB2 Workload</string>
      </array>
      <key>Hint</key><string />
    </dict>
  <key>SMPCSI Data Set</key>
  <dict>
    <key>Display Name</key><string>SMPCSI Data Set</string>
    <key>Value</key><string />
    <key>Hint</key><string />
  </dict>
  <key>SYSMOD</key>
  <dict>
    <key>Display Name</key><string>SYSMOD</string>
    <key>Value</key>
    <array>
      <string />
    </array>
    <key>Hint</key><string />
  </dict>
</plist>

```

```

        <key>Hint</key><string />
    </dict>
</dict>
</plist>

```

Example 3: The following example shows a complete sample of a Version 1.0 XML input document for the GET_SYSTEM_INFO stored procedure.

```

<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
    <key>Document Type Name</key>
    <string>Data Server System Input</string>
    <key>Document Type Major Version</key><integer>1</integer>
    <key>Document Type Minor Version</key><integer>0</integer>
    <key>Document Locale</key><string>en_US</string>
    <key>Optional Parameters</key>
    <dict>
        <key>SMPCSI Data Set</key>
        <dict>
            <key>Value</key><string>IXM180.GLOBAL.CSI</string>
        </dict>
        <key>SYSMOD</key>
        <dict>
            <key>Value</key>
            <array>
                <string>UK20028</string>
                <string>UK20030</string>
            </array>
        </dict>
    </dict>
</dict>
</plist>

```

You must specify the SMPCSI data set and one or more SYSMODs. SYSMOD status information will be returned for only the SYSMODs that are listed in the Optional Parameters section, provided that the SMPCSI data set that you specify is valid.

Example 4: The following example shows a complete sample of a Version 1.1 XML input document for the GET_SYSTEM_INFO stored procedure.

```

<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
    <key>Document Type Name</key><string>Data Server System Input</string>
    <key>Document Type Major Version</key><integer>1</integer>
    <key>Document Type Minor Version</key><integer>1</integer>
    <key>Document Locale</key><string>en_US</string>
    <key>Optional Parameters</key>
    <dict>
        <key>Include</key>
        <dict>
            <key>Value</key>
            <array>
                <string>Operating System Information</string>
                <string>Product Information</string>
                <string>DB2 MEPL</string>
                <string>Workload Manager (WLM) Classification Rules for
                    DB2 Workload</string>
            </array>
        </dict>
        <key>SMPCSI Data Set</key>
        <dict>
            <key>Value</key><string>IXM180.GLOBAL.CSI</string>
        </dict>
    </dict>
</dict>
</plist>

```

```

        <key>SYSMOD</key>
        <dict>
            <key>Value</key>
            <array>
                <string>UK24596</string>
                <string>UK24709</string>
            </array>
        </dict>
    </dict>
</dict>
</plist>

```

Example 5: The following example shows a fragment of an XML output document for the GET_SYSTEM_INFO stored procedure. In this example, the ellipsis (. . .) represent a dictionary entry that is comprised of the Display Name, Value, and Hint, such as:

```

<dict>
    <key>Display Name</key>
    <string>Name</string>
    <key>Value</key>
    <string>JES2</string>
    <key>Hint</key>
    <string />
</dict>
<?xml version="1.0" encoding="UTF-8" ?>
<plist version="1.0">
<dict>
    <key>Document Type Name</key>
    <string>Data Server System Output</string>
    <key>Document Type Major Version</key>
    <integer>1</integer>
    <key>Document Type Minor Version</key>
    <integer>1</integer>
    <key>Data Server Product Name</key>
    <string>DSN</string>
    <key>Data Server Product Version</key>
    <string>9.1.5</string>
    <key>Data Server Major Version</key>
    <integer>9</integer>
    <key>Data Server Minor Version</key>
    <integer>1</integer>
    <key>Data Server Platform</key>
    <string>z/OS</string>
    <key>Document Locale</key>
    <string>en_US</string>
    <key>Operating System Information</key>
<dict>
    <key>Display Name</key><string>Operating System Information</string>
    <key>Name and Release</key>
    ...
    <key>CPU</key>
<dict>
    <key>Display Name</key><string>CPU</string>
    <key>Model</key>
    ...
    <key>Number of Online CPUs</key>
    ...
    <key>Online CPUs</key>
<dict>
    <key>Display Name</key><string>Online CPUs</string>
    <key>CPU ID 01</key>
<dict>
    <key>Display Name</key><string>CPU ID 01</string>
    <key>Serial Number</key>
    ...

```

```

        <key>Hint</key><string />
      </dict>
    <key>Hint</key><string />
  </dict>
<key>Hint</key><string />
</dict>
<key>Real Storage Size</key>
<dict>
  <key>Display Name</key><string>Real Storage Size</string>
  <key>Value</key><integer>256</integer>
  <key>Display Unit</key><string>MB</string>
  <key>Hint</key><string />
</dict>
<key>Hint</key><string />
</dict>

<key>Product Information</key>
<dict>
  <key>Display Name</key><string>Product Information</string>
  <key>Primary Job Entry Subsystem</key>
  <dict>
    <key>Display Name</key><string>Primary Job Entry Subsystem</string>
    <key>Name</key>
    ...
    <key>Release</key>
    ...
    <key>Node Name</key>
    ...
    <key>Held Output Class</key>
    ...
    <key>Hint</key><string />
  </dict>
  <key>Security Software</key>
  <dict>
    <key>Display Name</key><string>Security Software</string>
    <key>Name</key>
    ...
    <key>FMID</key>
    ...
    <key>Hint</key><string />
  </dict>
  <key>DFSMS Release</key>
  ...
  <key>TSO Release</key>
  ...
  <key>VTAM Release</key>
  ...
  <key>Hint</key><string />
</dict>

<key>DB2 MEPL</key>
<dict>
  <key>Display Name</key><string>DB2 MEPL</string>
  <key>DSNUTILB</key>
  <dict>
    <key>Display Name</key><string>DSNUTILB</string>
    <key>DSNAA</key>
    <dict>
      <key>Display Name</key><string>DSNAA</string>
      <key>PTF Level</key>
      ...
      <key>PTF Apply Date</key>
      ...
    <key>Hint</key><string />
    </dict>
  </dict>

```

--- This is only a fragment of the utility modules that

are returned by the GET_SYSTEM_INFO stored procedure. ---

```
<key>Hint</key><string></string>
</dict>
```

```
--- This is only a fragment of the
DB2 MEPL information that is returned by
the GET_SYSTEM_INFO stored procedure. ---
```

```
</dict>
```

```
<key>SYSMOD Status</key>
```

```
<dict>
```

```
<key>Display Name</key><string>SYSMOD Status</string>
```

```
<key>AA15195</key>
```

```
<dict>
```

```
<key>Display Name</key><string>AA15195</string>
```

```
<key>Apply</key>
```

```
...
```

```
<key>Apply Date</key>
```

```
...
```

```
<key>Hint</key><string />
```

```
</dict>
```

```
--- This is only a fragment of the SYSMOD
status information that is returned by
the GET_SYSTEM_INFO stored procedure. ---
```

```
</dict>
```

```
<key>Workload Manager (WLM) Classification Rules for DB2 Workload</key>
```

```
<dict>
```

```
<key>Display Name</key>
```

```
<string>Workload Manager (WLM) Classification Rules for DB2 Workload</string>
```

```
<key>DB2</key>
```

```
<dict>
```

```
<key>Display Name</key><string>DB2</string>
```

```
<key>Hint</key><string />
```

```
</dict>
```

```
<key>DDF</key>
```

```
<dict>
```

```
<key>Display Name</key><string>DDF</string>
```

```
<key>1.1.1</key>
```

```
<dict>
```

```
<key>Display Name</key><string>1.1.1</string>
```

```
<key>Nesting Level</key>
```

```
...
```

```
<key>Qualifier Type</key>
```

```
...
```

```
<key>Qualifier Type Full Name</key>
```

```
...
```

```
<key>Qualifier Name</key>
```

```
...
```

```
<key>Start Position</key>
```

```
...
```

```
<key>Service Class</key>
```

```
...
```

```
<key>Report Class</key>
```

```
...
```

```
<key>Hint</key><string />
```

```
</dict>
```

```
<key>2.1.1</key>
```

```
<dict>
```

```
--- This dictionary entry describes the second classification
rule, and its format is the same as that of 1.1.1 above,
which describes the first classification rule. ---
```

```
</dict>
```

```

        <key>Hint</key><string />
    </dict>
    <key>Hint</key><string />
</dict>
</dict>
</plist>

```

Example 6: The following example shows a sample XML message document for the GET_SYSTEM_INFO stored procedure.

```

<?xml version="1.0" encoding="UTF-8" ?>
<plist version="1.0">
  <dict>
    <key>Document Type Name</key><string>Data Server Message</string>
    <key>Document Type Major Version</key><integer>1</integer>
    <key>Document Type Minor Version</key><integer>1</integer>
    <key>Data Server Product Name</key><string>DSN</string>
    <key>Data Server Product Version</key><string>9.1.5</string>
    <key>Data Server Major Version</key><integer>9</integer>
    <key>Data Server Minor Version</key><integer>1</integer>
    <key>Data Server Platform</key><string>z/OS</string>
    <key>Document Locale</key><string>en_US</string>
    <key>Short Message Text</key>
    <dict>
      <key>Display Name</key><string>Short Message Text</string>
      <key>Value</key>
      <string>DSNA647I DSNADMGS INVOCATION OF GIMAPI FAILED. Error processing
        command: QUERY . RC=12 CC=50504. GIM54701W ALLOCATION FAILED FOR
        SMPCSI - IKJ56228I DATA SET IXM180.GLOBAL.CSI NOT IN CATALOG OR
        CATALOG CAN NOT BE ACCESSED. GIM44232I GIMMPVIA - DYNAMIC
        ALLOCATION FAILED FOR THE GLOBAL ZONE, DATA SET IXM180.GLOBAL.CSI.
        GIM50504S ** OPEN PROCESSING FAILED FOR THE GLOBAL ZONE.</string>
      <key>Hint</key><string />
    </dict>
  </dict>
</plist>

```

Example 7: This example shows a simple and static Java program that calls the GET_SYSTEM_INFO stored procedure with an XML input document and an XPath that queries the value of the operating system name and release.

The XML input document is initially saved as a file called xml_input.xml that is in the same directory where the GetSystemDriver class resides. This sample program uses the following xml_input.xml file:

```

<?xml version="1.0" encoding="UTF-8" ?>
<plist version="1.0">
  <dict>
    <key>Document Type Name</key>
    <string>Data Server System Input</string>
    <key>Document Type Major Version</key>
    <integer>1</integer>
    <key>Document Type Minor Version</key>
    <integer>1</integer>
    <key>Document Locale</key>
    <string>en_US</string>
    <key>Optional Parameters</key>
    <dict>
      <key>Include</key>
      <dict>
        <key>Value</key>
        <array>
          <string>Operating System Information</string>
        </array>
      </dict>
    </dict>
  </dict>
</plist>

```

```

        </dict>
    </dict>
</dict>
</plist>

```

The XPath is statically created as a string object by the program and then converted to a BLOB to serve as input for the *xml_filter* parameter. After the stored procedure is called, the *xml_output* parameter contains only a single string and no XML document. This output is materialized as a file called *xml_output.xml* that is in the same directory where the *GetSystemDriver* class resides.

Sample invocation of the GET_SYSTEM_INFO stored procedure with a valid XML input document and a valid XPath:

```

//*****
// Licensed Materials - Property of IBM
// 5635-DB2
// (C) COPYRIGHT 1982, 2006 IBM Corp. All Rights Reserved.
//
// STATUS = Version 9
//*****
// Source file name: GetSystemDriver.java
//
// Sample: How to call SYSPROC.GET_SYSTEM_INFO with a valid XML input document
// and a valid XPath to extract the operating system name and release.
//
// The user runs the program by issuing:
// java GetSystemDriver <alias or //server/database> <userid> <password>
//
// The arguments are:
// <alias> - DB2 subsystem alias for type 2 or //server/database for type 4
// connectivity
// <userid> - user ID to connect as
// <password> - password to connect with
//*****
import java.io.*;
import java.sql.*;

public class GetSystemDriver
{
    public static void main (String[] args)
    {
        Connection con = null;
        CallableStatement cstmt = null;
        String driver = "com.ibm.db2.jcc.DB2Driver";
        String url = "jdbc:db2:";
        String userid = null;
        String password = null;

        // Parse arguments
        if (args.length != 3)
        {
            System.err.println("Usage: GetSystemDriver <alias or //server/database>
<userid> <password>");
            System.err.println("where <alias or //server/database> is DB2 subsystem
alias or //server/database for type 4 connectivity");
            System.err.println(" <userid> is user ID to connect as");
            System.err.println(" <password> is password to connect with");
            return;
        }
        url += args[0];
        userid = args[1];
        password = args[2];

        try {

```

```

        String str_xmlfilter = new String(
            "/plist/dict/key[.='Operating System Information']/following-sibling::
dict[1]" +
            "/key[.='Name and Release']/following-sibling::dict[1]" +
            "/key[.='Value']/following-sibling::string[1]");

        // Convert XML_FILTER to byte array to pass as BLOB
        byte[] xml_filter = str_xmlfilter.getBytes("UTF-8");

        // Read XML_INPUT from file
        File fptr = new File("xml_input.xml");

        int file_length = (int)fptr.length();
        byte[] xml_input = new byte[file_length];

        FileInputStream instream = new FileInputStream(fptr);
        int tot_bytes = instream.read(xml_input,0, xml_input.length);
        if (tot_bytes == -1) {
            System.out.println("Error during file read");
            return;
        }
        instream.close();

        // Load the DB2 Universal JDBC Driver
        Class.forName(driver);

        // Connect to database
        con = DriverManager.getConnection(url, userid, password);
        con.setAutoCommit(false);

        cstmt = con.prepareStatement("CALL SYSPROC.GET_SYSTEM_INFO(?,?,?,?,?,?)");

        // Major / Minor Version / Requested Locale
        cstmt.setInt(1, 1);
        cstmt.setInt(2, 1);
        cstmt.setString(3, "en_US");

        // Input documents
        cstmt.setObject(4, xml_input, Types.BLOB);
        cstmt.setObject(5, xml_filter, Types.BLOB);

        // Output Params
        cstmt.registerOutParameter(1, Types.INTEGER);
        cstmt.registerOutParameter(2, Types.INTEGER);
        cstmt.registerOutParameter(6, Types.BLOB);
        cstmt.registerOutParameter(7, Types.BLOB);

        cstmt.execute();
        con.commit();

        SQLWarning ctstmt_warning = cstmt.getWarnings();
        if (ctstmt_warning != null) {
            System.out.println("SQL Warning: " + ctstmt_warning.getMessage());
        }
        else {
            System.out.println("SQL Warning: None\r\n");
        }

        System.out.println("Major Version returned " + cstmt.getInt(1) );
        System.out.println("Minor Version returned " + cstmt.getInt(2) );

        // Get output BLOBs
        Blob b_out = cstmt.getBlob(6);

        if(b_out != null)
        {

```

```

        int out_length = (int)b_out.length();
        byte[] bxml_output = new byte[out_length];

        // Open an inputstream on BLOB data
        InputStream instr_out = b_out.getBinaryStream();

        // Copy from inputstream into byte array
        int out_len = instr_out.read(bxml_output, 0, out_length);

        // Write byte array content into FileOutputStream
        FileOutputStream fxml_out = new FileOutputStream("xml_output.xml");
        fxml_out.write(bxml_output, 0, out_length );

        //Close streams
        instr_out.close();
        fxml_out.close();
    }

    Blob b_msg = cstmt.getBlob(7);

    if(b_msg != null)
    {
        int msg_length = (int)b_msg.length();
        byte[] bxml_message = new byte[msg_length];

        // Open an inputstream on BLOB data
        InputStream instr_msg = b_msg.getBinaryStream();

        // Copy from inputstream into byte array
        int msg_len = instr_msg.read(bxml_message, 0, msg_length);

        // Write byte array content into FileOutputStream
        FileOutputStream fxml_msg = new FileOutputStream(new File("xml_message.
xml"));
        fxml_msg.write(bxml_message, 0, msg_length);

        //Close streams
        instr_msg.close();
        fxml_msg.close();
    }
}

catch (SQLException sqle) {
    System.out.println("Error during CALL "
        + " SQLSTATE = " + sqle.getSQLState()
        + " SQLCODE = " + sqle.getErrorCode()
        + " : " + sqle.getMessage());
}

catch (Exception e) {
    System.out.println("Internal Error " + e.toString());
}

finally
{
    if(cstmt != null)
        try { cstmt.close(); } catch ( SQLException sqle)
    { sqle.printStackTrace(); }
    if(con != null)
        try { con.close(); } catch ( SQLException sqle)
    { sqle.printStackTrace(); }
}
}
}

```

XPath expressions for filtering output

You can use an XPath expression to filter the XML output that is returned by the GET_CONFIG, GET_MESSAGE, and GET_SYSTEM_INFO stored procedures.

To filter the output, specify a valid XPath query string in the *xml_filter* parameter of the stored procedure.

The following restrictions apply to the XPath expression that you specify:

- The XPath expression must reference a single value.
- The XPath expression must always be absolute from the root node. For example, the following path expressions are allowed: `/`, `nodename`, `.`, and `...`. The following expressions are not allowed: `//` and `@`
- The only predicates allowed are `[path='value']` and `[n]`.
- The only axis allowed is `following-sibling`.
- The XPath expression must end with one of the following, and, if necessary, be appended with the predicate `[1]`: `following-sibling::string`, `following-sibling::data`, `following-sibling::date`, `following-sibling::real`, or `following-sibling::integer`.
- Unless the axis is found at the end of the XPath expression, it must be followed by a `::dict`, `::string`, `::data`, `::date`, `::real`, or `::integer`, and if necessary, be appended with the predicate `[1]`.
- The only supported XPath operator is `=`.
- The XPath expression cannot contain a function, namespace, processing instruction, or comment.

Tip: If the stored procedure operates in complete mode, do not apply filtering, or an SQLCODE (+20458) is raised.

Example: The following XPath expression selects the value for the Data Server Product Version key from an XML output document:

```
/plist/dict/key[.='Data Server Product Version']/following-sibling::string[1]
```

The stored procedure returns the string 9.1.5 in the *xml_output* parameter if the value of the Data Server Product Version is 9.1.5. Therefore, the stored procedure call returns a single value rather than an XML document.

Information resources for DB2 for z/OS and related products

Many information resources are available to help you use DB2 for z/OS and many related products. A large amount of technical information about IBM products is now available online in information centers or on library Web sites.

Disclaimer: Any Web addresses that are included here are accurate at the time this information is being published. However, Web addresses sometimes change. If you visit a Web address that is listed here but that is no longer valid, you can try to find the current Web address for the product information that you are looking for at either of the following sites:

- <http://www.ibm.com/support/publications/us/library/index.shtml>, which lists the IBM information centers that are available for various IBM products
- <http://www.elink.ibm.com/public/applications/publications/cgibin/pbi.cgi>, which is the IBM Publications Center, where you can download online PDF books or order printed books for various IBM products

DB2 for z/OS product information

The primary place to find and use information about DB2 for z/OS is the Information Management Software for z/OS Solutions Information Center (<http://publib.boulder.ibm.com/infocenter/imzic>), which also contains information about IMS, QMF, and many DB2 and IMS Tools products. The majority of the DB2 for z/OS information in this information center is also available in the books that are identified in the following table. You can access these books at the DB2 for z/OS library Web site (<http://www.ibm.com/software/data/db2/zos/library.html>) or at the IBM Publications Center (<http://www.elink.ibm.com/public/applications/publications/cgibin/pbi.cgi>).

Table 208. DB2 Version 9.1 for z/OS book titles

Title	Publication number	Available in information center	Available in PDF	Available in BookManager® format	Available in printed book
<i>DB2 Version 9.1 for z/OS Administration Guide</i>	SC18-9840	X	X	X	X
<i>DB2 Version 9.1 for z/OS Application Programming & SQL Guide</i>	SC18-9841	X	X	X	X
<i>DB2 Version 9.1 for z/OS Application Programming Guide and Reference for Java</i>	SC18-9842	X	X	X	X
<i>DB2 Version 9.1 for z/OS Codes</i>	GC18-9843	X	X	X	X
<i>DB2 Version 9.1 for z/OS Command Reference</i>	SC18-9844	X	X	X	X
<i>DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration</i>	SC18-9845	X	X	X	X
<i>DB2 Version 9.1 for z/OS Diagnosis Guide and Reference</i> ¹	LY37-3218		X	X	X
<i>DB2 Version 9.1 for z/OS Diagnostic Quick Reference</i>	LY37-3219				X

Table 208. DB2 Version 9.1 for z/OS book titles (continued)

Title	Publication number	Available in information center	Available in PDF	Available in BookManager® format	Available in printed book
<i>DB2 Version 9.1 for z/OS Installation Guide</i>	GC18-9846	X	X	X	X
<i>DB2 Version 9.1 for z/OS Introduction to DB2</i>	SC18-9847	X	X	X	X
<i>DB2 Version 9.1 for z/OS Licensed Program Specifications</i>	GC18-9848		X		X
<i>DB2 Version 9.1 for z/OS Messages</i>	GC18-9849	X	X	X	X
<i>DB2 Version 9.1 for z/OS ODBC Guide and Reference</i>	SC18-9850	X	X	X	X
<i>DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide</i>	SC18-9851	X	X	X	X
<i>DB2 Version 9.1 for z/OS Optimization Service Center</i>		X			
<i>DB2 Version 9.1 for z/OS Program Directory</i>	GI10-8737		X		X
<i>DB2 Version 9.1 for z/OS RACF Access Control Module Guide</i>	SC18-9852		X	X	
<i>DB2 Version 9.1 for z/OS Reference for Remote DRDA Requesters and Servers</i>	SC18-9853	X	X	X	
<i>DB2 Version 9.1 for z/OS Reference Summary</i>	SX26-3854				
<i>DB2 Version 9.1 for z/OS SQL Reference</i>	SC18-9854	X	X	X	X
<i>DB2 Version 9.1 for z/OS Utility Guide and Reference</i>	SC18-9855	X	X	X	X
<i>DB2 Version 9.1 for z/OS What's New?</i>	GC18-9856	X	X	X	X
<i>DB2 Version 9.1 for z/OS XML Extender Administration and Programming</i>	SC18-9857	X	X	X	X
<i>DB2 Version 9.1 for z/OS XML Guide</i>	SC18-9858	X	X	X	X

Note:

1. *DB2 Version 9.1 for z/OS Diagnosis Guide and Reference* is available in PDF and BookManager formats on the DB2 Version 9.1 for z/OS Licensed Collection kit, LK3T-7195. You can order this License Collection kit on the IBM Publications Center site (<http://www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi>). This book is also available in online format in DB2 data set DSN910.SDSNIVPD(DSNDR).

Information resources for related products

In the following table, related product names are listed in alphabetic order, and the associated Web addresses of product information centers or library Web pages are indicated.

Table 209. Related product information resource locations

Related product	Information resources
C/C++ for z/OS	Library Web site: http://www.ibm.com/software/awdtools/czos/library/ This product is now called z/OS XL C/C++.

Table 209. Related product information resource locations (continued)

Related product	Information resources
CICS Transaction Server for z/OS	Information center: http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp
COBOL	Information center: http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp This product is now called Enterprise COBOL for z/OS.
DB2 Connect	Information center: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp This resource is for DB2 Connect 9.
DB2 Database for Linux, UNIX, and Windows	Information center: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp This resource is for DB2 9 for Linux, UNIX, and Windows.
DB2 Query Management Facility	Information center: http://publib.boulder.ibm.com/infocenter/imzic
DB2 Server for VSE & VM	One of the following locations: <ul style="list-style-type: none"> • For VSE: http://www.ibm.com/support/docview.wss?rs=66&uid=swg27003758 • For VM: http://www.ibm.com/support/docview.wss?rs=66&uid=swg27003759
DB2 Tools	One of the following locations: <ul style="list-style-type: none"> • Information center: http://publib.boulder.ibm.com/infocenter/imzic • Library Web site: http://www.ibm.com/software/data/db2imstools/library.html <p>These resources include information about the following products and others:</p> <ul style="list-style-type: none"> • DB2 Administration Tool • DB2 Automation Tool • DB2 DataPropagator (also known as WebSphere Replication Server for z/OS) • DB2 Log Analysis Tool • DB2 Object Restore Tool • DB2 Query Management Facility • DB2 SQL Performance Analyzer
DB2 [®] Universal Database [™] for iSeries [®]	Information center: http://www.ibm.com/systems/i/infocenter/
Debug Tool for z/OS	Information center: http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp
Enterprise COBOL for z/OS	Information center: http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp
Enterprise PL/I for z/OS	Information center: http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp
IMS	Information center: http://publib.boulder.ibm.com/infocenter/imzic
IMS Tools	One of the following locations: <ul style="list-style-type: none"> • Information center: http://publib.boulder.ibm.com/infocenter/imzic • Library Web site: http://www.ibm.com/software/data/db2imstools/library.html <p>These resources have information about the following products and others:</p> <ul style="list-style-type: none"> • IMS Batch Terminal Simulator for z/OS • IMS Connect • IMS HALDB Conversion and Maintenance Aid • IMS High Performance Utility products • IMS DataPropagator • IMS Online Reorganization Facility • IMS Performance Analyzer

Table 209. Related product information resource locations (continued)

Related product	Information resources
PL/I	Information center: http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp This product is now called Enterprise PL/I for z/OS.
System z [®]	http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp
Tivoli OMEGAMONXE for DB2 Performance Expert on z/OS	Information center: http://publib.boulder.ibm.com/infocenter/tivihelp/v15r1/index.jsp?topic=/com.ibm.ko2pe.doc/ko2welcome.htm In earlier releases, this product was called DB2 Performance Expert for z/OS.
WebSphere Application Server	Information center: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp
WebSphere Message Broker with Rules and Formatter Extension	Information center: http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp The product is also known as WebSphere MQ Integrator Broker.
WebSphere MQ	Information center: http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp The resource includes information about MQSeries [®] .
WebSphere Replication Server for z/OS	Either of the following locations: <ul style="list-style-type: none"> Information center: http://publib.boulder.ibm.com/infocenter/imzic Library Web site: http://www.ibm.com/software/data/db2imstools/library.html This product is also known as DB2 DataPropagator.
z/Architecture [®]	Library Center site: http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

Table 209. Related product information resource locations (continued)

Related product	Information resources
z/OS	<p data-bbox="505 258 1365 281">Library Center site: http://www.ibm.com/servers/eserver/zseries/zos/bkserv/</p> <p data-bbox="505 310 1453 333">This resource includes information about the following z/OS elements and components:</p> <ul data-bbox="505 348 951 1444" style="list-style-type: none"> • Character Data Representation Architecture • Device Support Facilities • DFSORT • Fortran • High Level Assembler • NetView • SMP/E for z/OS • SNA • TCP/IP • TotalStorage[®] Enterprise Storage Server • VTAM • z/OS C/C++ • z/OS Communications Server • z/OS DCE • z/OS DFSMS • z/OS DFSMS Access Method Services • z/OS DFSMSdss • z/OS DFSMSHsm • z/OS DFSMSdfp • z/OS ICSF • z/OS ISPF • z/OS JES3 • z/OS Language Environment • z/OS Managed System Infrastructure • z/OS MVS • z/OS MVS JCL • z/OS Parallel Sysplex[®] • z/OS RMF[™] • z/OS Security Server • z/OS UNIX System Services
z/OS XL C/C++	<p data-bbox="505 1465 1114 1488">http://www.ibm.com/software/awdtools/czos/library/</p>

The following information resources from IBM are not necessarily specific to a single product:

- The DB2 for z/OS Information Roadmap; available at: <http://www.ibm.com/software/data/db2/zos/roadmap.html>
- DB2 Redbooks[®] and Redbooks about related products; available at: <http://www.ibm.com/redbooks>
- IBM Educational resources:
 - Information about IBM educational offerings is available on the Web at: <http://www.ibm.com/software/sw-training/>

- A collection of glossaries of IBM terms in multiple languages is available on the IBM Terminology Web site at: <http://www.ibm.com/software/globalization/terminology/index.jsp>
- National Language Support information; available at the IBM Publications Center at: <http://www.elink.ibm.com/public/applications/publications/cgibin/pbi.cgi>
- *SQL Reference for Cross-Platform Development*; available at the following developerWorks® site: <http://www.ibm.com/developerworks/db2/library/techarticle/0206sqlref/0206sqlref.html>

The following information resources are not published by IBM but can be useful to users of DB2 for z/OS and related products:

- Database design topics:
 - *DB2 for z/OS and OS/390 Development for Performance Volume I*, by Gabrielle Wiorkowski, Gabrielle & Associates, ISBN 0-96684-605-2
 - *DB2 for z/OS and OS/390 Development for Performance Volume II*, by Gabrielle Wiorkowski, Gabrielle & Associates, ISBN 0-96684-606-0
 - *Handbook of Relational Database Design*, by C. Fleming and B. Von Halle, Addison Wesley, ISBN 0-20111-434-8
- Distributed Relational Database Architecture™ (DRDA) specifications; <http://www.opengroup.org>
- Domain Name System: *DNS and BIND*, Third Edition, Paul Albitz and Cricket Liu, O'Reilly, ISBN 0-59600-158-4
- Microsoft® Open Database Connectivity (ODBC) information; <http://msdn.microsoft.com/library/>
- Unicode information; <http://www.unicode.org>

How to obtain DB2 information

You can access the official information about the DB2 product in a number of ways.

- “DB2 on the Web”
- “DB2 product information”
- “DB2 education” on page 956
- “How to order the DB2 library” on page 956

DB2 on the Web

Stay current with the latest information about DB2 by visiting the DB2 home page on the Web:

www.ibm.com/software/db2zos

On the DB2 home page, you can find links to a wide variety of information resources about DB2. You can read news items that keep you informed about the latest enhancements to the product. Product announcements, press releases, fact sheets, and technical articles help you plan and implement your database management strategy.

DB2 product information

The official DB2 for z/OS information is available in various formats and delivery methods. IBM provides mid-version updates to the information in the information center and in softcopy updates that are available on the Web and on CD-ROM.

Information Management Software for z/OS Solutions Information Center

DB2 product information is viewable in the information center, which is the primary delivery vehicle for information about DB2 for z/OS, IMS, QMF, and related tools. This information center enables you to search across related product information in multiple languages for data management solutions for the z/OS environment and print individual topics or sets of related topics. You can also access, download, and print PDFs of the publications that are associated with the information center topics. Product technical information is provided in a format that offers more options and tools for accessing, integrating, and customizing information resources. The information center is based on Eclipse open source technology.

The Information Management Software for z/OS Solutions Information Center is viewable at the following Web site:

<http://publib.boulder.ibm.com/infocenter/imzic>

CD-ROMs and DVD

Books for DB2 are available on a CD-ROM that is included with your product shipment:

- DB2 V9.1 for z/OS Licensed Library Collection, LK3T-7195, in English

The CD-ROM contains the collection of books for DB2 V9.1 for z/OS in PDF and BookManager formats. Periodically, IBM refreshes the books on subsequent editions of this CD-ROM.

The books for DB2 for z/OS are also available on the following CD-ROM and DVD collection kits, which contain online books for many IBM products:

- IBM z/OS Software Products Collection , SK3T-4270, in English
- IBM z/OS Software Products DVD Collection , SK3T-4271, in English

PDF format

Many of the DB2 books are available in PDF (Portable Document Format) for viewing or printing from CD-ROM or the DB2 home page on the Web or from the information center. Download the PDF books to your intranet for distribution throughout your enterprise.

BookManager format

You can use online books on CD-ROM to read, search across books, print portions of the text, and make notes in these BookManager books. Using the IBM Softcopy Reader, appropriate IBM Library Readers, or the BookManager Read product, you can view these books in the z/OS, Windows, and VM environments. You can also view and search many of the DB2 BookManager books on the Web.

DB2 education

IBM Education and Training offers a wide variety of classroom courses to help you quickly and efficiently gain DB2 expertise. IBM schedules classes in cities all over the world. You can find class information, by country, at the IBM Learning Services Web site:

www.ibm.com/services/learning

IBM also offers classes at your location, at a time that suits your needs. IBM can customize courses to meet your exact requirements. For more information, including the current local schedule, contact your IBM representative.

How to order the DB2 library

To order books, visit the IBM Publication Center on the Web:

www.elink.ibm.com/public/applications/publications/cgibin/pbi.cgi

From the IBM Publication Center, you can go to the Publication Notification System (PNS). PNS users receive electronic notifications of updated publications in their profiles. You have the option of ordering the updates by using the publications direct ordering application or any other IBM publication ordering channel. The PNS application does not send automatic shipments of publications. You will receive updated publications and a bill for them if you respond to the electronic notification.

You can also order DB2 publications and CD-ROMs from your IBM representative or the IBM branch office that serves your locality. If your location is within the United States or Canada, you can place your order by calling one of the toll-free numbers:

- In the U.S., call 1-800-879-2755.
- In Canada, call 1-800-426-4968.

To order additional copies of licensed publications, specify the SOFTWARE option. To order additional publications or CD-ROMs, specify the PUBLICATIONS option.

Be prepared to give your customer number, the product number, and either the feature codes or order numbers that you want.

How to use the DB2 library

Titles of books in the library begin with DB2 Version 9.1 for z/OS. However, references from one book in the library to another are shortened and do not include the product name, version, and release. Instead, they point directly to the section that holds the information.

If you are new to DB2 for z/OS, *Introduction to DB2 for z/OS* provides a comprehensive introduction to DB2 Version 9.1 for z/OS. Topics included in this book explain the basic concepts that are associated with relational database management systems in general, and with DB2 for z/OS in particular.

The most rewarding task associated with a database management system is asking questions of it and getting answers, the task called *end use*. Other tasks are also necessary—defining the parameters of the system, putting the data in place, and so on. The tasks that are associated with DB2 are grouped into the following major categories.

Installation

If you are involved with DB2 only to install the system, *DB2 Installation Guide* might be all you need.

If you will be using data sharing capabilities you also need *DB2 Data Sharing: Planning and Administration*, which describes installation considerations for data sharing.

End use

End users issue SQL statements to retrieve data. They can also insert, update, or delete data, with SQL statements. They might need an introduction to SQL, detailed instructions for using SPUFI, and an alphabetized reference to the types of SQL statements. This information is found in *DB2 Application Programming and SQL Guide*, and *DB2 SQL Reference*.

End users can also issue SQL statements through the DB2 Query Management Facility (QMF) or some other program, and the library for that licensed program might provide all the instruction or reference material they need. For a list of the titles in the DB2 QMF library, see the bibliography at the end of this book.

Application programming

Some users access DB2 without knowing it, using programs that contain SQL statements. DB2 application programmers write those programs. Because they write SQL statements, they need the same resources that end users do.

Application programmers also need instructions for many other topics:

- How to transfer data between DB2 and a host program—written in Java, C, or COBOL, for example
- How to prepare to compile a program that embeds SQL statements
- How to process data from two systems simultaneously, for example, DB2 and IMS or DB2 and CICS

- How to write distributed applications across operating systems
- How to write applications that use Open Database Connectivity (ODBC) to access DB2 servers
- How to write applications that use JDBC and SQLJ with the Java programming language to access DB2 servers
- How to write applications to store XML data on DB2 servers and retrieve XML data from DB2 servers.

The material needed for writing a host program containing SQL is in *DB2 Application Programming and SQL Guide*.

The material needed for writing applications that use JDBC and SQLJ to access DB2 servers is in *DB2 Application Programming Guide and Reference for Java*. The material needed for writing applications that use DB2 CLI or ODBC to access DB2 servers is in *DB2 ODBC Guide and Reference*. The material needed for working with XML data in DB2 is in *DB2 XML Guide*. For handling errors, see *DB2 Messages* and *DB2 Codes*.

If you will be working in a distributed environment, you will need *DB2 Reference for Remote DRDA Requesters and Servers*.

Information about writing applications across operating systems can be found in *IBM DB2 SQL Reference for Cross-Platform Development*.

System and database administration

Administration covers almost everything else. *DB2 Administration Guide* divides some of those tasks among the following sections:

- **DB2 concepts:** Introduces DB2 structures, the DB2 environment, and high availability.
- **Designing a database:** Discusses the decisions that must be made when designing a database and tells how to implement the design by creating and altering DB2 objects, loading data, and adjusting to changes.
- **Security and auditing:** Describes ways of controlling access to the DB2 system and to data within DB2, to audit aspects of DB2 usage, and to answer other security and auditing concerns.
- **Operation and recovery:** Describes the steps in normal day-to-day operation and discusses the steps one should take to prepare for recovery in the event of some failure.

DB2 Performance Monitoring and Tuning Guide explains how to monitor the performance of the DB2 system and its parts. It also lists things that can be done to make some parts run faster.

If you will be using the RACF access control module for DB2 authorization checking, you will need *DB2 RACF Access Control Module Guide*.

If you are involved with DB2 only to design the database, or plan operational procedures, you need *DB2 Administration Guide*. If you also want to carry out your own plans by creating DB2 objects, granting privileges, running utility jobs, and so on, you also need:

- *DB2 SQL Reference*, which describes the SQL statements you use to create, alter, and drop objects and grant and revoke privileges
- *DB2 Utility Guide and Reference*, which explains how to run utilities

- *DB2 Command Reference*, which explains how to run commands

If you will be using data sharing, you need *DB2 Data Sharing: Planning and Administration*, which describes how to plan for and implement data sharing.

Additional information about system and database administration can be found in *DB2 Messages* and *DB2 Codes*, which list messages and codes issued by DB2, with explanations and suggested responses.

Diagnosis

Diagnostics detect and describe errors in the DB2 program. They might also recommend or apply a remedy. The documentation for this task is in *DB2 Diagnosis Guide and Reference*, *DB2 Messages*, and *DB2 Codes*.

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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This information is intended to help you to plan for and administer DB2 Version 9.1 for z/OS. This information also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by DB2 Version 9.1 for z/OS.

General-use Programming Interface and Associated Guidance Information

General-use Programming Interfaces allow the customer to write programs that obtain the services of DB2 Version 9.1 for z/OS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

 General-use Programming Interface and Associated Guidance Information...


Product-sensitive Programming Interface and Associated Guidance Information

Product-sensitive Programming Interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive Programming Interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

 Product-sensitive Programming Interface and Associated Guidance Information...


Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)[®] are trademarks or registered marks 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 trademarks that were owned by IBM at the time this information was published. A complete and current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of other companies, and have been used at least once in the DB2 for z/OS library:

- Microsoft, Windows, Windows NT[®], and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Intel[®], Intel logo, Intel Inside[®], Intel Inside logo, Intel Centrino[®], Intel Centrino logo, Celeron[®], Intel Xeon[®], Intel SpeedStep[®], Itanium[®], and Pentium[®] are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Adobe[®], the Adobe logo, Postscript, and the Postscript log are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

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

Glossary

abend See abnormal end of task.

abend reason code

A 4-byte hexadecimal code that uniquely identifies a problem with DB2.

abnormal end of task (abend)

Termination of a task, job, or subsystem because of an error condition that recovery facilities cannot resolve during execution.

access method services

The facility that is used to define, alter, delete, print, and reproduce VSAM key-sequenced data sets.

access path

The path that is used to locate data that is specified in SQL statements. An access path can be indexed or sequential.

active log

The portion of the DB2 log to which log records are written as they are generated. The active log always contains the most recent log records. See also archive log.

address space

A range of virtual storage pages that is identified by a number (ASID) and a collection of segment and page tables that map the virtual pages to real pages of the computer's memory.

address space connection

The result of connecting an allied address space to DB2. See also allied address space and task control block.

address space identifier (ASID)

A unique system-assigned identifier for an address space.

AFTER trigger

A trigger that is specified to be activated after a defined trigger event (an insert, update, or delete operation on the table that is specified in a trigger definition). Contrast with BEFORE trigger and INSTEAD OF trigger.

agent In DB2, the structure that associates all processes that are involved in a DB2 unit of work. See also allied agent and system agent.

aggregate function

An operation that derives its result by using values from one or more rows. Contrast with scalar function.

alias

An alternative name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 subsystem. An alias can be qualified with a schema qualifier and can thereby be referenced by other users. Contrast with synonym.

allied address space

An area of storage that is external to DB2 and that is connected to DB2. An allied address space can request DB2 services. See also address space.

allied agent

An agent that represents work requests that originate in allied address spaces. See also system agent.

allied thread

A thread that originates at the local DB2 subsystem and that can access data at a remote DB2 subsystem.

allocated cursor

A cursor that is defined for a stored procedure result set by using the SQL ALLOCATE CURSOR statement.

ambiguous cursor

A database cursor for which DB2 cannot determine whether it is used for update or read-only purposes.

APAR See authorized program analysis report.

APF See authorized program facility.

API See application programming interface.

APPL A VTAM network definition statement that is used to define DB2 to VTAM as an application program that uses SNA LU 6.2 protocols.

application

A program or set of programs that performs a task; for example, a payroll application.

application plan

The control structure that is produced during the bind process. DB2 uses the

application plan to process SQL statements that it encounters during statement execution.

application process

The unit to which resources and locks are allocated. An application process involves the execution of one or more programs.

application programming interface (API)

A functional interface that is supplied by the operating system or by a separately orderable licensed program that allows an application program that is written in a high-level language to use specific data or functions of the operating system or licensed program.

application requester

The component on a remote system that generates DRDA requests for data on behalf of an application.

application server

The target of a request from a remote application. In the DB2 environment, the application server function is provided by the distributed data facility and is used to access DB2 data from remote applications.

archive log

The portion of the DB2 log that contains log records that have been copied from the active log. See also active log.

ASCII An encoding scheme that is used to represent strings in many environments, typically on PCs and workstations. Contrast with EBCDIC and Unicode.

ASID See address space identifier.

attachment facility

An interface between DB2 and TSO, IMS, CICS, or batch address spaces. An attachment facility allows application programs to access DB2.

attribute

A characteristic of an entity. For example, in database design, the phone number of an employee is an attribute of that employee.

authorization ID

A string that can be verified for connection to DB2 and to which a set of privileges is allowed. An authorization ID can represent an individual or an organizational group.

authorized program analysis report (APAR)

A report of a problem that is caused by a suspected defect in a current release of an IBM supplied program.

authorized program facility (APF)

A facility that allows an installation to identify system or user programs that can use sensitive system functions.

automatic bind

(More correctly *automatic rebind*.) A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.

automatic query rewrite

A process that examines an SQL statement that refers to one or more base tables or materialized query tables, and, if appropriate, rewrites the query so that it performs better.

auxiliary index

An index on an auxiliary table in which each index entry refers to a LOB or XML document.

auxiliary table

A table that contains columns outside the actual table in which they are defined. Auxiliary tables can contain either LOB or XML data.

backout

The process of undoing uncommitted changes that an application process made. A backout is often performed in the event of a failure on the part of an application process, or as a result of a deadlock situation.

backward log recovery

The final phase of restart processing during which DB2 scans the log in a backward direction to apply UNDO log records for all aborted changes.

base table

A table that is created by the SQL CREATE TABLE statement and that holds persistent data. Contrast with clone table, materialized query table, result table, temporary table, and transition table.

base table space

A table space that contains base tables.

<p>basic row format</p> <p>A row format in which values for columns are stored in the row in the order in which the columns are defined by the CREATE TABLE statement. Contrast with reordered row format.</p> <p>basic sequential access method (BSAM)</p> <p>An access method for storing or retrieving data blocks in a continuous sequence, using either a sequential-access or a direct-access device.</p> <p>BEFORE trigger</p> <p>A trigger that is specified to be activated before a defined trigger event (an insert, an update, or a delete operation on the table that is specified in a trigger definition). Contrast with AFTER trigger and INSTEAD OF trigger.</p> <p>binary large object (BLOB)</p> <p>A binary string data type that contains a sequence of bytes that can range in size from 0 bytes to 2 GB, less 1 byte. This string does not have an associated code page and character set. BLOBs can contain, for example, image, audio, or video data. In general, BLOB values are used whenever a binary string might exceed the limits of the VARBINARY type.</p> <p>binary string</p> <p>A sequence of bytes that is not associated with a CCSID. Binary string data type can be further classified as BINARY, VARBINARY, or BLOB.</p> <p>bind</p> <p>A process by which a usable control structure with SQL statements is generated; the structure is often called an access plan, an application plan, or a package. During this bind process, access paths to the data are selected, and some authorization checking is performed. See also automatic bind.</p> <p>bit data</p> <ul style="list-style-type: none"> • Data with character type CHAR or VARCHAR that is defined with the FOR BIT DATA clause. Note that using BINARY or VARBINARY rather than FOR BIT DATA is highly recommended. • Data with character type CHAR or VARCHAR that is defined with the FOR BIT DATA clause. 	<ul style="list-style-type: none"> • A form of character data. Binary data is generally more highly recommended than character-for-bit data. <p>BLOB See binary large object.</p> <p>block fetch</p> <p>A capability in which DB2 can retrieve, or fetch, a large set of rows together. Using block fetch can significantly reduce the number of messages that are being sent across the network. Block fetch applies only to non-rowset cursors that do not update data.</p> <p>bootstrap data set (BSDS)</p> <p>A VSAM data set that contains name and status information for DB2 and RBA range specifications for all active and archive log data sets. The BSDS also contains passwords for the DB2 directory and catalog, and lists of conditional restart and checkpoint records.</p> <p>BSAM</p> <p>See basic sequential access method.</p> <p>BSDS See bootstrap data set.</p> <p>buffer pool</p> <p>An area of memory into which data pages are read, modified, and held during processing.</p> <p>built-in data type</p> <p>A data type that IBM supplies. Among the built-in data types for DB2 for z/OS are string, numeric, XML, ROWID, and datetime. Contrast with distinct type.</p> <p>built-in function</p> <p>A function that is generated by DB2 and that is in the SYSIBM schema. Contrast with user-defined function. See also function, cast function, external function, sourced function, and SQL function.</p> <p>business dimension</p> <p>A category of data, such as products or time periods, that an organization might want to analyze.</p> <p>cache structure</p> <p>A coupling facility structure that stores data that can be available to all members of a Sysplex. A DB2 data sharing group uses cache structures as group buffer pools.</p> <p>CAF See call attachment facility.</p>
---	---

call attachment facility (CAF)

A DB2 attachment facility for application programs that run in TSO or z/OS batch. The CAF is an alternative to the DSN command processor and provides greater control over the execution environment. Contrast with Recoverable Resource Manager Services attachment facility.

call-level interface (CLI)

A callable application programming interface (API) for database access, which is an alternative to using embedded SQL.

cascade delete

A process by which DB2 enforces referential constraints by deleting all descendent rows of a deleted parent row.

CASE expression

An expression that is selected based on the evaluation of one or more conditions.

cast function

A function that is used to convert instances of a (source) data type into instances of a different (target) data type.

castout

The DB2 process of writing changed pages from a group buffer pool to disk.

castout owner

The DB2 member that is responsible for casting out a particular page set or partition.

catalog

In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

catalog table

Any table in the DB2 catalog.

CCSID

See coded character set identifier.

CDB

See communications database.

CDRA

See Character Data Representation Architecture.

CEC

See central processor complex.

central electronic complex (CEC)

See central processor complex.

central processor complex (CPC)

A physical collection of hardware that consists of main storage, one or more central processors, timers, and channels.

central processor (CP)

The part of the computer that contains the sequencing and processing facilities for instruction execution, initial program load, and other machine operations.

CFRM See coupling facility resource management.

CFRM policy

The allocation rules for a coupling facility structure that are declared by a z/OS administrator.

character conversion

The process of changing characters from one encoding scheme to another.

Character Data Representation Architecture (CDRA)

An architecture that is used to achieve consistent representation, processing, and interchange of string data.

character large object (CLOB)

A character string data type that contains a sequence of bytes that represent characters (single-byte, multibyte, or both) that can range in size from 0 bytes to 2 GB, less 1 byte. In general, CLOB values are used whenever a character string might exceed the limits of the VARCHAR type.

character set

A defined set of characters.

character string

A sequence of bytes that represent bit data, single-byte characters, or a mixture of single-byte and multibyte characters. Character data can be further classified as CHARACTER, VARCHAR, or CLOB.

check constraint

A user-defined constraint that specifies the values that specific columns of a base table can contain.

check integrity

The condition that exists when each row in a table conforms to the check constraints that are defined on that table.

check pending

A state of a table space or partition that prevents its use by some utilities and by some SQL statements because of rows that violate referential constraints, check constraints, or both.

checkpoint

A point at which DB2 records status information on the DB2 log; the recovery process uses this information if DB2 abnormally terminates.

child lock

For explicit hierarchical locking, a lock that is held on either a table, page, row, or a large object (LOB). Each child lock has a parent lock. See also parent lock.

CI See control interval.

CICS Represents (in this information): CICS Transaction Server for z/OS: Customer Information Control System Transaction Server for z/OS.

CICS attachment facility

A facility that provides a multithread connection to DB2 to allow applications that run in the CICS environment to execute DB2 statements.

claim A notification to DB2 that an object is being accessed. Claims prevent drains from occurring until the claim is released, which usually occurs at a commit point. Contrast with drain.

claim class

A specific type of object access that can be one of the following isolation levels:

- Cursor stability (CS)
- Repeatable read (RR)
- Write

class of service

A VTAM term for a list of routes through a network, arranged in an order of preference for their use.

clause In SQL, a distinct part of a statement, such as a SELECT clause or a WHERE clause.

CLI See call-level interface.

client See requester.

CLOB See character large object.

clone object

An object that is associated with a clone table, including the clone table itself and check constraints, indexes, and BEFORE triggers on the clone table.

clone table

A table that is structurally identical to a base table. The base and clone table each

have separate underlying VSAM data sets, which are identified by their data set instance numbers. Contrast with base table.

closed application

An application that requires exclusive use of certain statements on certain DB2 objects, so that the objects are managed solely through the external interface of that application.

clustering index

An index that determines how rows are physically ordered (*clustered*) in a table space. If a clustering index on a partitioned table is not a partitioning index, the rows are ordered in cluster sequence within each data partition instead of spanning partitions.

CM See compatibility mode.

CM* See compatibility mode*.

C++ member

A data object or function in a structure, union, or class.

C++ member function

An operator or function that is declared as a member of a class. A member function has access to the private and protected data members and to the member functions of objects in its class. Member functions are also called methods.

C++ object

A region of storage. An object is created when a variable is defined or a new function is invoked.

An instance of a class.

coded character set

A set of unambiguous rules that establish a character set and the one-to-one relationships between the characters of the set and their coded representations.

coded character set identifier (CCSID)

A 16-bit number that uniquely identifies a coded representation of graphic characters. It designates an encoding scheme identifier and one or more pairs that consist of a character set identifier and an associated code page identifier.

code page

A set of assignments of characters to code

points. Within a code page, each code point has only one specific meaning. In EBCDIC, for example, the character *A* is assigned code point X'C1', and character *B* is assigned code point X'C2'.

code point

In CDRA, a unique bit pattern that represents a character in a code page.

code unit

The fundamental binary width in a computer architecture that is used for representing character data, such as 7 bits, 8 bits, 16 bits, or 32 bits. Depending on the character encoding form that is used, each code point in a coded character set can be represented by one or more code units.

coexistence

During migration, the period of time in which two releases exist in the same data sharing group.

cold start

A process by which DB2 restarts without processing any log records. Contrast with warm start.

collection

A group of packages that have the same qualifier.

column

The vertical component of a table. A column has a name and a particular data type (for example, character, decimal, or integer).

column function

See aggregate function.

"come from" checking

An LU 6.2 security option that defines a list of authorization IDs that are allowed to connect to DB2 from a partner LU.

command

A DB2 operator command or a DSN subcommand. A command is distinct from an SQL statement.

command prefix

A 1- to 8-character command identifier. The command prefix distinguishes the command as belonging to an application or subsystem rather than to z/OS.

command recognition character (CRC)

A character that permits a z/OS console

operator or an IMS subsystem user to route DB2 commands to specific DB2 subsystems.

command scope

The scope of command operation in a data sharing group.

commit

The operation that ends a unit of work by releasing locks so that the database changes that are made by that unit of work can be perceived by other processes. Contrast with rollback.

commit point

A point in time when data is considered consistent.

common service area (CSA)

In z/OS, a part of the common area that contains data areas that are addressable by all address spaces. Most DB2 use is in the extended CSA, which is above the 16-MB line.

communications database (CDB)

A set of tables in the DB2 catalog that are used to establish conversations with remote database management systems.

comparison operator

A token (such as =, >, or <) that is used to specify a relationship between two values.

compatibility mode* (CM*)

A stage of the version-to-version migration process that applies to a DB2 subsystem or data sharing group that was in enabling-new-function mode (ENFM), enabling-new-function mode* (ENFM*), or new-function mode (NFM) at one time. Fallback to a prior version is not supported. When in compatibility mode*, a DB2 data sharing group cannot coexist with members that are still at the prior version level. Contrast with compatibility mode, enabling-new-function mode, enabling-new-function mode*, and new-function mode.

compatibility mode (CM)

The first stage of the version-to-version migration process. In a DB2 data sharing group, members in compatibility mode can coexist with members that are still at the prior version level. Fallback to the prior version is also supported. When in compatibility mode, the DB2 subsystem

cannot use any new functions of the new version. Contrast with compatibility mode*, enabling-new-function mode, enabling-new-function mode*, and new-function mode.

composite key

An ordered set of key columns or expressions of the same table.

compression dictionary

The dictionary that controls the process of compression and decompression. This dictionary is created from the data in the table space or table space partition.

concurrency

The shared use of resources by more than one application process at the same time.

conditional restart

A DB2 restart that is directed by a user-defined conditional restart control record (CRCR).

connection

In SNA, the existence of a communication path between two partner LUs that allows information to be exchanged (for example, two DB2 subsystems that are connected and communicating by way of a conversation).

connection context

In SQLJ, a Java object that represents a connection to a data source.

connection declaration clause

In SQLJ, a statement that declares a connection to a data source.

connection handle

The data object containing information that is associated with a connection that DB2 ODBC manages. This includes general status information, transaction status, and diagnostic information.

connection ID

An identifier that is supplied by the attachment facility and that is associated with a specific address space connection.

consistency token

A timestamp that is used to generate the version identifier for an application. See also version.

constant

A language element that specifies an unchanging value. Constants are classified

as string constants or numeric constants. Contrast with variable.

constraint

A rule that limits the values that can be inserted, deleted, or updated in a table. See referential constraint, check constraint, and unique constraint.

context

An application's logical connection to the data source and associated DB2 ODBC connection information that allows the application to direct its operations to a data source. A DB2 ODBC context represents a DB2 thread.

contracting conversion

A process that occurs when the length of a converted string is smaller than that of the source string. For example, this process occurs when an EBCDIC mixed-data string that contains DBCS characters is converted to ASCII mixed data; the converted string is shorter because the shift codes are removed.

control interval (CI)

- A unit of information that VSAM transfers between virtual and auxiliary storage.
- In a key-sequenced data set or file, the set of records that an entry in the sequence-set index record points to.

conversation

Communication, which is based on LU 6.2 or Advanced Program-to-Program Communication (APPC), between an application and a remote transaction program over an SNA logical unit-to-logical unit (LU-LU) session that allows communication while processing a transaction.

coordinator

The system component that coordinates the commit or rollback of a unit of work that includes work that is done on one or more other systems.

coprocessor

See SQL statement coprocessor.

copy pool

A collection of names of storage groups that are processed collectively for fast replication operations.

copy target

A named set of SMS storage groups that are to be used as containers for copy pool volume copies. A copy target is an SMS construct that lets you define which storage groups are to be used as containers for volumes that are copied by using FlashCopy functions.

copy version

A point-in-time FlashCopy copy that is managed by HSM. Each copy pool has a version parameter that specifies the number of copy versions to be maintained on disk.

correlated columns

A relationship between the value of one column and the value of another column.

correlated subquery

A subquery (part of a WHERE or HAVING clause) that is applied to a row or group of rows of a table or view that is named in an outer subselect statement.

correlation ID

An identifier that is associated with a specific thread. In TSO, it is either an authorization ID or the job name.

correlation name

An identifier that is specified and used within a single SQL statement as the exposed name for objects such as a table, view, table function reference, nested table expression, or result of a data change statement. Correlation names are useful in an SQL statement to allow two distinct references to the same base table and to allow an alternative name to be used to represent an object.

cost category

A category into which DB2 places cost estimates for SQL statements at the time the statement is bound. The cost category is externalized in the COST_CATEGORY column of the DSN_STATEMNT_TABLE when a statement is explained.

coupling facility

A special PR/SM logical partition (LPAR) that runs the coupling facility control program and provides high-speed caching, list processing, and locking functions in a Parallel Sysplex.

coupling facility resource management (CFRM)

A component of z/OS that provides the

services to manage coupling facility resources in a Parallel Sysplex. This management includes the enforcement of CFRM policies to ensure that the coupling facility and structure requirements are satisfied.

CP See central processor.

CPC See central processor complex.

CRC See command recognition character.

created temporary table

A persistent table that holds temporary data and is defined with the SQL statement CREATE GLOBAL TEMPORARY TABLE. Information about created temporary tables is stored in the DB2 catalog and can be shared across application processes. Contrast with declared temporary table. See also temporary table.

cross-system coupling facility (XCF)

A component of z/OS that provides functions to support cooperation between authorized programs that run within a Sysplex.

cross-system extended services (XES)

A set of z/OS services that allow multiple instances of an application or subsystem, running on different systems in a Sysplex environment, to implement high-performance, high-availability data sharing by using a coupling facility.

CS See cursor stability.

CSA See common service area.

CT See cursor table.

current data

Data within a host structure that is current with (identical to) the data within the base table.

current status rebuild

The second phase of restart processing during which the status of the subsystem is reconstructed from information on the log.

cursor A control structure that an application program uses to point to a single row or multiple rows within some ordered set of rows of a result table. A cursor can be used to retrieve, update, or delete rows from a result table.

cursor sensitivity

The degree to which database updates are visible to the subsequent FETCH statements in a cursor.

cursor stability (CS)

The isolation level that provides maximum concurrency without the ability to read uncommitted data. With cursor stability, a unit of work holds locks only on its uncommitted changes and on the current row of each of its cursors. See also read stability, repeatable read, and uncommitted read.

cursor table (CT)

The internal representation of a cursor.

cycle A set of tables that can be ordered so that each table is a descendent of the one before it, and the first table is a descendent of the last table. A self-referencing table is a cycle with a single member. See also referential cycle.

database

A collection of tables, or a collection of table spaces and index spaces.

database access thread (DBAT)

A thread that accesses data at the local subsystem on behalf of a remote subsystem.

database administrator (DBA)

An individual who is responsible for designing, developing, operating, safeguarding, maintaining, and using a database.

database alias

The name of the target server if it is different from the location name. The database alias is used to provide the name of the database server as it is known to the network.

database descriptor (DBD)

An internal representation of a DB2 database definition, which reflects the data definition that is in the DB2 catalog. The objects that are defined in a database descriptor are table spaces, tables, indexes, index spaces, relationships, check constraints, and triggers. A DBD also contains information about accessing tables in the database.

database exception status

In a data sharing environment, an indication that something is wrong with a database.

database identifier (DBID)

An internal identifier of the database.

database management system (DBMS)

A software system that controls the creation, organization, and modification of a database and the access to the data that is stored within it.

database request module (DBRM)

A data set member that is created by the DB2 precompiler and that contains information about SQL statements. DBRMs are used in the bind process.

database server

The target of a request from a local application or a remote intermediate database server.

data currency

The state in which the data that is retrieved into a host variable in a program is a copy of the data in the base table.

data dictionary

A repository of information about an organization's application programs, databases, logical data models, users, and authorizations.

data partition

A VSAM data set that is contained within a partitioned table space.

data-partitioned secondary index (DPSI)

A secondary index that is partitioned according to the underlying data. Contrast with nonpartitioned secondary index.

data set instance number

A number that indicates the data set that contains the data for an object.

data sharing

A function of DB2 for z/OS that enables applications on different DB2 subsystems to read from and write to the same data concurrently.

data-sharing group

A collection of one or more DB2

subsystems that directly access and change the same data while maintaining data integrity.

data sharing member

A DB2 subsystem that is assigned by XCF services to a data sharing group.

data source

A local or remote relational or non-relational data manager that is capable of supporting data access via an ODBC driver that supports the ODBC APIs. In the case of DB2 for z/OS, the data sources are always relational database managers.

data type

An attribute of columns, constants, variables, parameters, special registers, and the results of functions and expressions.

data warehouse

A system that provides critical business information to an organization. The data warehouse system cleanses the data for accuracy and currency, and then presents the data to decision makers so that they can interpret and use it effectively and efficiently.

DBA See database administrator.

DBAT See database access thread.

DB2 catalog

A collection of tables that are maintained by DB2 and contain descriptions of DB2 objects, such as tables, views, and indexes.

DBCLOB

See double-byte character large object.

DB2 command

An instruction to the DB2 subsystem that a user enters to start or stop DB2, to display information on current users, to start or stop databases, to display information on the status of databases, and so on.

DBCS See double-byte character set.

DBD See database descriptor.

DB2I See DB2 Interactive.

DBID See database identifier.

DB2 Interactive (DB2I)

An interactive service within DB2 that

facilitates the execution of SQL statements, DB2 (operator) commands, and programmer commands, and the invocation of utilities.

DBMS

See database management system.

DBRM

See database request module.

DB2 thread

The database manager structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to the database manager resources and services. Most DB2 for z/OS functions execute under a thread structure.

DCLGEN

See declarations generator.

DDF See distributed data facility.

deadlock

Unresolvable contention for the use of a resource, such as a table or an index.

declarations generator (DCLGEN)

A subcomponent of DB2 that generates SQL table declarations and COBOL, C, or PL/I data structure declarations that conform to the table. The declarations are generated from DB2 system catalog information.

declared temporary table

A non-persistent table that holds temporary data and is defined with the SQL statement DECLARE GLOBAL TEMPORARY TABLE. Information about declared temporary tables is not stored in the DB2 catalog and can be used only by the application process that issued the DECLARE statement. Contrast with created temporary table. See also temporary table.

default value

A predetermined value, attribute, or option that is assumed when no other value is specified. A default value can be defined for column data in DB2 tables by specifying the DEFAULT keyword in an SQL statement that changes data (such as INSERT, UPDATE, and MERGE).

deferred embedded SQL

SQL statements that are neither fully static nor fully dynamic. These statements

are embedded within an application and are prepared during the execution of the application.

deferred write

The process of asynchronously writing changed data pages to disk.

degree of parallelism

The number of concurrently executed operations that are initiated to process a query.

delete hole

The location on which a cursor is positioned when a row in a result table is refetched and the row no longer exists on the base table. See also update hole.

delete rule

The rule that tells DB2 what to do to a dependent row when a parent row is deleted. Delete rules include CASCADE, RESTRICT, SET NULL, or NO ACTION.

delete trigger

A trigger that is defined with the triggering delete SQL operation.

delimited identifier

A sequence of characters that are enclosed within escape characters.

delimiter token

A string constant, a delimited identifier, an operator symbol, or any of the special characters that are shown in DB2 syntax diagrams.

denormalization

The intentional duplication of columns in multiple tables to increase data redundancy. Denormalization is sometimes necessary to minimize performance problems. Contrast with normalization.

dependent

An object (row, table, or table space) that has at least one parent. The object is also said to be a dependent (row, table, or table space) of its parent. See also parent row, parent table, and parent table space.

dependent row

A row that contains a foreign key that matches the value of a primary key in the parent row.

dependent table

A table that is a dependent in at least one referential constraint.

descendent

An object that is a dependent of an object or is the dependent of a descendent of an object.

descendent row

A row that is dependent on another row, or a row that is a descendent of a dependent row.

descendent table

A table that is a dependent of another table, or a table that is a descendent of a dependent table.

deterministic function

A user-defined function whose result is dependent on the values of the input arguments. That is, successive invocations with the same input values produce the same answer. Sometimes referred to as a *not-variant* function. Contrast with nondeterministic function (sometimes called a *variant function*).

dimension

A data category such as time, products, or markets. The elements of a dimension are referred to as members. See also dimension table.

dimension table

The representation of a dimension in a star schema. Each row in a dimension table represents all of the attributes for a particular member of the dimension. See also dimension, star schema, and star join.

directory

The DB2 system database that contains internal objects such as database descriptors and skeleton cursor tables.

disk

A direct-access storage device that records data magnetically.

distinct type

A user-defined data type that is represented as an existing type (its source type), but is considered to be a separate and incompatible type for semantic purposes.

distributed data

Data that resides on a DBMS other than the local system.

distributed data facility (DDF)

A set of DB2 components through which DB2 communicates with another relational database management system.

Distributed Relational Database Architecture (DRDA)

A connection protocol for distributed relational database processing that is used by IBM relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems. See also DRDA access.

DNS See domain name server.

DOCID

See document ID.

document ID

A value that uniquely identifies a row that contains an XML column. This value is stored with the row and never changes.

domain

The set of valid values for an attribute.

domain name

The name by which TCP/IP applications refer to a TCP/IP host within a TCP/IP network.

domain name server (DNS)

A special TCP/IP network server that manages a distributed directory that is used to map TCP/IP host names to IP addresses.

double-byte character large object (DBCLOB)

A graphic string data type in which a sequence of bytes represent double-byte characters that range in size from 0 bytes to 2 GB, less 1 byte. In general, DBCLOB values are used whenever a double-byte character string might exceed the limits of the VARGRAPHIC type.

double-byte character set (DBCS)

A set of characters, which are used by national languages such as Japanese and Chinese, that have more symbols than can be represented by a single byte. Each character is 2 bytes in length. Contrast with single-byte character set and multibyte character set.

double-precision floating point number

A 64-bit approximate representation of a real number.

DPSI See data-partitioned secondary index.

drain The act of acquiring a locked resource by quiescing access to that object. Contrast with claim.

drain lock

A lock on a claim class that prevents a claim from occurring.

DRDA

See Distributed Relational Database Architecture.

DRDA access

An open method of accessing distributed data that you can use to connect to another database server to execute packages that were previously bound at the server location.

DSN

- The default DB2 subsystem name.
- The name of the TSO command processor of DB2.
- The first three characters of DB2 module and macro names.

dynamic cursor

A named control structure that an application program uses to change the size of the result table and the order of its rows after the cursor is opened. Contrast with static cursor.

dynamic dump

A dump that is issued during the execution of a program, usually under the control of that program.

dynamic SQL

SQL statements that are prepared and executed at run time. In dynamic SQL, the SQL statement is contained as a character string in a host variable or as a constant, and it is not precompiled.

EA-enabled table space

A table space or index space that is enabled for extended addressability and that contains individual partitions (or pieces, for LOB table spaces) that are greater than 4 GB.

EB See exabyte.

EBCDIC

Extended binary coded decimal interchange code. An encoding scheme that is used to represent character data in the z/OS, VM, VSE, and iSeries environments. Contrast with ASCII and Unicode.

embedded SQL

SQL statements that are coded within an application program. See static SQL.

enabling-new-function mode* (ENFM*)

A transitional stage of the version-to-version migration process that applies to a DB2 subsystem or data sharing group that was in new-function mode (NFM) at one time. When in enabling-new-function mode*, a DB2 subsystem or data sharing group is preparing to use the new functions of the new version but cannot yet use them. A data sharing group that is in enabling-new-function mode* cannot coexist with members that are still at the prior version level. Fallback to a prior version is not supported. Contrast with compatibility mode, compatibility mode*, enabling-new-function mode, and new-function mode.

enabling-new-function mode (ENFM)

A transitional stage of the version-to-version migration process during which the DB2 subsystem or data sharing group is preparing to use the new functions of the new version. When in enabling-new-function mode, a DB2 data sharing group cannot coexist with members that are still at the prior version level. Fallback to a prior version is not supported, and new functions of the new version are not available for use in enabling-new-function mode. Contrast with compatibility mode, compatibility mode*, enabling-new-function mode*, and new-function mode.

enclave

In Language Environment, an independent collection of routines, one of which is designated as the main routine. An enclave is similar to a program or run unit. See also WLM enclave.

encoding scheme

A set of rules to represent character data (ASCII, EBCDIC, or Unicode).

| ENFM See enabling-new-function mode.

ENFM*

| See enabling-new-function mode*.

| **entity** A person, object, or concept about which
| information is stored. In a relational
| database, entities are represented as
| tables. A database includes information
| about the entities in an organization or
| business, and their relationships to each
| other.

enumerated list

A set of DB2 objects that are defined with a LISTDEF utility control statement in which pattern-matching characters (*, %, _, or ?) are not used.

environment

A collection of names of logical and physical resources that are used to support the performance of a function.

environment handle

A handle that identifies the global context for database access. All data that is pertinent to all objects in the environment is associated with this handle.

equijoin

A join operation in which the join-condition has the form *expression = expression*. See also join, full outer join, inner join, left outer join, outer join, and right outer join.

error page range

A range of pages that are considered to be physically damaged. DB2 does not allow users to access any pages that fall within this range.

escape character

The symbol, a double quotation (") for example, that is used to enclose an SQL delimited identifier.

exabyte

A unit of measure for processor, real and virtual storage capacities, and channel volume that has a value of 1 152 921 504 606 846 976 bytes or 2⁶⁰.

exception

An SQL operation that involves the EXCEPT set operator, which combines two result tables. The result of an exception operation consists of all of the rows that are in only one of the result tables.

exception table

A table that holds rows that violate referential constraints or check constraints that the CHECK DATA utility finds.

exclusive lock

A lock that prevents concurrently executing application processes from reading or changing data. Contrast with share lock.

executable statement

An SQL statement that can be embedded in an application program, dynamically prepared and executed, or issued interactively.

execution context

In SQLJ, a Java object that can be used to control the execution of SQL statements.

exit routine

A user-written (or IBM-provided default) program that receives control from DB2 to perform specific functions. Exit routines run as extensions of DB2.

expanding conversion

A process that occurs when the length of a converted string is greater than that of the source string. For example, this process occurs when an ASCII mixed-data string that contains DBCS characters is converted to an EBCDIC mixed-data string; the converted string is longer because shift codes are added.

explicit hierarchical locking

Locking that is used to make the parent-child relationship between resources known to IRLM. This kind of locking avoids global locking overhead when no inter-DB2 interest exists on a resource.

explicit privilege

A privilege that has a name and is held as the result of an SQL GRANT statement and revoked as the result of an SQL REVOKE statement. For example, the SELECT privilege.

exposed name

A correlation name or a table or view name for which a correlation name is not specified.

expression

An operand or a collection of operators and operands that yields a single value.

Extended Recovery Facility (XRF)

A facility that minimizes the effect of failures in z/OS, VTAM, the host processor, or high-availability applications during sessions between high-availability applications and designated terminals. This facility provides an alternative subsystem to take over sessions from the failing subsystem.

Extensible Markup Language (XML)

A standard metalanguage for defining markup languages that is a subset of Standardized General Markup Language (SGML).

external function

A function that has its functional logic implemented in a programming language application that resides outside the database, in the file system of the database server. The association of the function with the external code application is specified by the EXTERNAL clause in the CREATE FUNCTION statement. External functions can be classified as external scalar functions and external table functions. Contrast with sourced function, built-in function, and SQL function.

external procedure

A procedure that has its procedural logic implemented in an external programming language application. The association of the procedure with the external application is specified by a CREATE PROCEDURE statement with a LANGUAGE clause that has a value other than SQL and an EXTERNAL clause that implicitly or explicitly specifies the name of the external application. Contrast with external SQL procedure and native SQL procedure.

external routine

A user-defined function or stored procedure that is based on code that is written in an external programming language.

external SQL procedure

An SQL procedure that is processed using a generated C program that is a representation of the procedure. When an external SQL procedure is called, the C program representation of the procedure is executed in a stored procedures address

space. Contrast with external procedure and native SQL procedure.

failed member state

A state of a member of a data sharing group in which the member's task, address space, or z/OS system terminates before the state changes from active to quiesced.

fallback

The process of returning to a previous release of DB2 after attempting or completing migration to a current release. Fallback is supported only from a subsystem that is in compatibility mode.

false global lock contention

A contention indication from the coupling facility that occurs when multiple lock names are hashed to the same indicator and when no real contention exists.

fan set

A direct physical access path to data, which is provided by an index, hash, or link; a fan set is the means by which DB2 supports the ordering of data.

federated database

The combination of a DB2 server (in Linux, UNIX, and Windows environments) and multiple data sources to which the server sends queries. In a federated database system, a client application can use a single SQL statement to join data that is distributed across multiple database management systems and can view the data as if it were local.

fetch orientation

The specification of the desired placement of the cursor as part of a FETCH statement. The specification can be before or after the rows of the result table (with BEFORE or AFTER). The specification can also have either a single-row fetch orientation (for example, NEXT, LAST, or ABSOLUTE *n*) or a rowset fetch orientation (for example, NEXT ROWSET, LAST ROWSET, or ROWSET STARTING AT ABSOLUTE *n*).

field procedure

A user-written exit routine that is designed to receive a single value and transform (encode or decode) it in any way the user can specify.

file reference variable

A host variable that is declared with one of the derived data types (BLOB_FILE, CLOB_FILE, DBCLOB_FILE); file reference variables direct the reading of a LOB from a file or the writing of a LOB into a file.

filter factor

A number between zero and one that estimates the proportion of rows in a table for which a predicate is true.

fixed-length string

A character, graphic, or binary string whose length is specified and cannot be changed. Contrast with varying-length string.

FlashCopy

A function on the IBM Enterprise Storage Server that can, in conjunction with the BACKUP SYSTEM utility, create a point-in-time copy of data while an application is running.

foreign key

A column or set of columns in a dependent table of a constraint relationship. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table. Each foreign key value must either match a parent key value in the related parent table or be null.

forest An ordered set of subtrees of XML nodes.

forward log recovery

The third phase of restart processing during which DB2 processes the log in a forward direction to apply all REDO log records.

free space

The total amount of unused space in a page; that is, the space that is not used to store records or control information is free space.

full outer join

The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of both tables. See also join, equijoin, inner join, left outer join, outer join, and right outer join.

fullselect

A subselect, a fullselect in parentheses, or

| a number of both that are combined by
| set operators. Fullselect specifies a result
| table. If a set operator is not used, the
| result of the fullselect is the result of the
| specified subselect or fullselect.

fully escaped mapping

A mapping from an SQL identifier to an XML name when the SQL identifier is a column name.

function

A mapping, which is embodied as a program (the function body) that is invocable by means of zero or more input values (arguments) to a single value (the result). See also aggregate function and scalar function.

Functions can be user-defined, built-in, or generated by DB2. (See also built-in function, cast function, external function, sourced function, SQL function, and user-defined function.)

function definer

The authorization ID of the owner of the schema of the function that is specified in the CREATE FUNCTION statement.

function package

A package that results from binding the DBRM for a function program.

function package owner

The authorization ID of the user who binds the function program's DBRM into a function package.

function signature

The logical concatenation of a fully qualified function name with the data types of all of its parameters.

GB Gigabyte. A value of (1 073 741 824 bytes).

GBP See group buffer pool.

GBP-dependent

The status of a page set or page set partition that is dependent on the group buffer pool. Either read/write interest is active among DB2 subsystems for this page set, or the page set has changed pages in the group buffer pool that have not yet been cast out to disk.

generalized trace facility (GTF)

A z/OS service program that records significant system events such as I/O

interrupts, SVC interrupts, program interrupts, or external interrupts.

generic resource name

A name that VTAM uses to represent several application programs that provide the same function in order to handle session distribution and balancing in a Sysplex environment.

geographic feature

An object on the surface of the Earth (such as a city or river), a space (such as a safety zone around a hazardous site), or an event that occurs at a location (such as an auto accident that occurred at a particular intersection).

geographic information system

A complex of objects, data, and applications that is used to create and analyze spatial information about geographic features.

getpage

An operation in which DB2 accesses a data page.

global lock

A lock that provides concurrency control within and among DB2 subsystems. The scope of the lock is across all DB2 subsystems of a data sharing group.

global lock contention

Conflicts on locking requests between different DB2 members of a data sharing group when those members are trying to serialize shared resources.

governor

See resource limit facility.

graphic string

A sequence of DBCS characters. Graphic data can be further classified as GRAPHIC, VARGRAPHIC, or DBCLOB.

GRECP

See group buffer pool recovery pending.

gross lock

The *shared*, *update*, or *exclusive* mode locks on a table, partition, or table space.

group buffer pool duplexing

The ability to write data to two instances of a group buffer pool structure: a primary group buffer pool and a secondary group buffer pool. z/OS publications refer to these instances as the

“old” (for primary) and “new” (for secondary) structures.

group buffer pool (GBP)

A coupling facility cache structure that is used by a data sharing group to cache data and to ensure that the data is consistent for all members.

group buffer pool recovery pending (GRECP)

The state that exists after the buffer pool for a data sharing group is lost. When a page set is in this state, changes that are recorded in the log must be applied to the affected page set before the page set can be used.

group level

The release level of a data sharing group, which is established when the first member migrates to a new release.

group name

The z/OS XCF identifier for a data sharing group.

group restart

A restart of at least one member of a data sharing group after the loss of either locks or the shared communications area.

GTF See generalized trace facility.

handle

In DB2 ODBC, a variable that refers to a data structure and associated resources. See also statement handle, connection handle, and environment handle.

help panel

A screen of information that presents tutorial text to assist a user at the workstation or terminal.

heuristic damage

The inconsistency in data between one or more participants that results when a heuristic decision to resolve an indoubt LUW at one or more participants differs from the decision that is recorded at the coordinator.

heuristic decision

A decision that forces indoubt resolution at a participant by means other than automatic resynchronization between coordinator and participant.

histogram statistics

A way of summarizing data distribution. This technique divides up the range of

possible values in a data set into intervals, such that each interval contains approximately the same percentage of the values. A set of statistics are collected for each interval.

hole A row of the result table that cannot be accessed because of a delete or an update that has been performed on the row. See also delete hole and update hole.

home address space

The area of storage that z/OS currently recognizes as *dispatched*.

host The set of programs and resources that are available on a given TCP/IP instance.

host expression

A Java variable or expression that is referenced by SQL clauses in an SQLJ application program.

host identifier

A name that is declared in the host program.

host language

A programming language in which you can embed SQL statements.

host program

An application program that is written in a host language and that contains embedded SQL statements.

host structure

In an application program, a structure that is referenced by embedded SQL statements.

host variable

In an application program written in a host language, an application variable that is referenced by embedded SQL statements.

host variable array

An array of elements, each of which corresponds to a value for a column. The dimension of the array determines the maximum number of rows for which the array can be used.

IBM System z9 Integrated Processor (zIIP)

A specialized processor that can be used for some DB2 functions.

IDCAMS

An IBM program that is used to process access method services commands. It can

be invoked as a job or jobstep, from a TSO terminal, or from within a user's application program.

IDCAMS LISTCAT

A facility for obtaining information that is contained in the access method services catalog.

identity column

| A column that provides a way for DB2 to
| automatically generate a numeric value
| for each row. Identity columns are
| defined with the AS IDENTITY clause.
| Uniqueness of values can be ensured by
| defining a unique index that contains
| only the identity column. A table can
| have no more than one identity column.

IFCID See instrumentation facility component identifier.

IFI See instrumentation facility interface.

IFI call

An invocation of the instrumentation facility interface (IFI) by means of one of its defined functions.

image copy

An exact reproduction of all or part of a table space. DB2 provides utility programs to make full image copies (to copy the entire table space) or incremental image copies (to copy only those pages that have been modified since the last image copy).

IMS attachment facility

A DB2 subcomponent that uses z/OS subsystem interface (SSI) protocols and cross-memory linkage to process requests from IMS to DB2 and to coordinate resource commitment.

in-abort

A status of a unit of recovery. If DB2 fails after a unit of recovery begins to be rolled back, but before the process is completed, DB2 continues to back out the changes during restart.

in-commit

A status of a unit of recovery. If DB2 fails after beginning its phase 2 commit processing, it "knows," when restarted, that changes made to data are consistent. Such units of recovery are termed *in-commit*.

independent

An object (row, table, or table space) that is neither a parent nor a dependent of another object.

index A set of pointers that are logically ordered by the values of a key. Indexes can provide faster access to data and can enforce uniqueness on the rows in a table.

index-controlled partitioning

A type of partitioning in which partition boundaries for a partitioned table are controlled by values that are specified on the CREATE INDEX statement. Partition limits are saved in the LIMITKEY column of the SYSIBM.SYSINDEXPART catalog table.

index key

The set of columns in a table that is used to determine the order of index entries.

index partition

A VSAM data set that is contained within a partitioning index space.

index space

A page set that is used to store the entries of one index.

indicator column

A 4-byte value that is stored in a base table in place of a LOB column.

indicator variable

A variable that is used to represent the null value in an application program. If the value for the selected column is null, a negative value is placed in the indicator variable.

indoubt

A status of a unit of recovery. If DB2 fails after it has finished its phase 1 commit processing and before it has started phase 2, only the commit coordinator knows if an individual unit of recovery is to be committed or rolled back. At restart, if DB2 lacks the information it needs to make this decision, the status of the unit of recovery is *indoubt* until DB2 obtains this information from the coordinator. More than one unit of recovery can be *indoubt* at restart.

indoubt resolution

The process of resolving the status of an *indoubt* logical unit of work to either the committed or the rollback state.

inflight

A status of a unit of recovery. If DB2 fails before its unit of recovery completes phase 1 of the commit process, it merely backs out the updates of its unit of recovery at restart. These units of recovery are termed *inflight*.

inheritance

The passing downstream of class resources or attributes from a parent class in the class hierarchy to a child class.

initialization file

For DB2 ODBC applications, a file containing values that can be set to adjust the performance of the database manager.

inline copy

A copy that is produced by the LOAD or REORG utility. The data set that the inline copy produces is logically equivalent to a full image copy that is produced by running the COPY utility with read-only access (SHRLEVEL REFERENCE).

inner join

The result of a join operation that includes only the matched rows of both tables that are being joined. See also join, equijoin, full outer join, left outer join, outer join, and right outer join.

inoperative package

A package that cannot be used because one or more user-defined functions or procedures that the package depends on were dropped. Such a package must be explicitly rebound. Contrast with invalid package.

insensitive cursor

A cursor that is not sensitive to inserts, updates, or deletes that are made to the underlying rows of a result table after the result table has been materialized.

insert trigger

A trigger that is defined with the triggering SQL operation, an insert.

install The process of preparing a DB2 subsystem to operate as a z/OS subsystem.

INSTEAD OF trigger

A trigger that is associated with a single view and is activated by an insert, update, or delete operation on the view and that can define how to propagate the

insert, update, or delete operation on the view to the underlying tables of the view. Contrast with BEFORE trigger and AFTER trigger.

instrumentation facility component identifier (IFCID)

A value that names and identifies a trace record of an event that can be traced. As a parameter on the START TRACE and MODIFY TRACE commands, it specifies that the corresponding event is to be traced.

instrumentation facility interface (IFI)

A programming interface that enables programs to obtain online trace data about DB2, to submit DB2 commands, and to pass data to DB2.

Interactive System Productivity Facility (ISPF)

An IBM licensed program that provides interactive dialog services in a z/OS environment.

inter-DB2 R/W interest

A property of data in a table space, index, or partition that has been opened by more than one member of a data sharing group and that has been opened for writing by at least one of those members.

intermediate database server

The target of a request from a local application or a remote application requester that is forwarded to another database server.

internal resource lock manager (IRLM)

A z/OS subsystem that DB2 uses to control communication and database locking.

internationalization

The support for an encoding scheme that is able to represent the code points of characters from many different geographies and languages. To support all geographies, the Unicode standard requires more than 1 byte to represent a single character. See also Unicode.

intersection

An SQL operation that involves the INTERSECT set operator, which combines two result tables. The result of an intersection operation consists of all of the rows that are in both result tables.

invalid package

A package that depends on an object (other than a user-defined function) that is dropped. Such a package is implicitly rebound on invocation. Contrast with inoperative package.

IP address

A value that uniquely identifies a TCP/IP host.

IRLM See internal resource lock manager.

isolation level

The degree to which a unit of work is isolated from the updating operations of other units of work. See also cursor stability, read stability, repeatable read, and uncommitted read.

ISPF See Interactive System Productivity Facility.

iterator

In SQLJ, an object that contains the result set of a query. An iterator is equivalent to a cursor in other host languages.

iterator declaration clause

In SQLJ, a statement that generates an iterator declaration class. An iterator is an object of an iterator declaration class.

JAR See Java Archive.

Java Archive (JAR)

A file format that is used for aggregating many files into a single file.

JDBC A Sun Microsystems database application programming interface (API) for Java that allows programs to access database management systems by using callable SQL.

join A relational operation that allows retrieval of data from two or more tables based on matching column values. See also equijoin, full outer join, inner join, left outer join, outer join, and right outer join.

KB Kilobyte. A value of 1024 bytes.

Kerberos

A network authentication protocol that is designed to provide strong authentication for client/server applications by using secret-key cryptography.

Kerberos ticket

A transparent application mechanism that transmits the identity of an initiating

principal to its target. A simple ticket contains the principal's identity, a session key, a timestamp, and other information, which is sealed using the target's secret key.

key A column, an ordered collection of columns, or an expression that is identified in the description of a table, index, or referential constraint. The same column or expression can be part of more than one key.

key-sequenced data set (KSDS)

A VSAM file or data set whose records are loaded in key sequence and controlled by an index.

KSDS See key-sequenced data set.

large object (LOB)

A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single- and double-byte characters. A LOB can be up to 2 GB minus 1 byte in length. See also binary large object, character large object, and double-byte character large object.

last agent optimization

An optimized commit flow for either presumed-nothing or presumed-abort protocols in which the last agent, or final participant, becomes the commit coordinator. This flow saves at least one message.

latch A DB2 mechanism for controlling concurrent events or the use of system resources.

LCID See log control interval definition.

LDS See linear data set.

leaf page

An index page that contains pairs of keys and RIDs and that points to actual data. Contrast with nonleaf page.

left outer join

The result of a join operation that includes the matched rows of both tables that are being joined, and that preserves the unmatched rows of the first table. See also join, equijoin, full outer join, inner join, outer join, and right outer join.

limit key

The highest value of the index key for a partition.

linear data set (LDS)

A VSAM data set that contains data but no control information. A linear data set can be accessed as a byte-addressable string in virtual storage.

linkage editor

A computer program for creating load modules from one or more object modules or load modules by resolving cross references among the modules and, if necessary, adjusting addresses.

link-edit

The action of creating a loadable computer program using a linkage editor.

list

A type of object, which DB2 utilities can process, that identifies multiple table spaces, multiple index spaces, or both. A list is defined with the LISTDEF utility control statement.

list structure

A coupling facility structure that lets data be shared and manipulated as elements of a queue.

L-lock See logical lock.

load module

A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.

LOB See large object.

LOB locator

A mechanism that allows an application program to manipulate a large object value in the database system. A LOB locator is a fullword integer value that represents a single LOB value. An application program retrieves a LOB locator into a host variable and can then apply SQL operations to the associated LOB value using the locator.

LOB lock

A lock on a LOB value.

LOB table space

A table space that contains all the data for a particular LOB column in the related base table.

local

A way of referring to any object that the local DB2 subsystem maintains. A *local table*, for example, is a table that is maintained by the local DB2 subsystem. Contrast with remote.

locale The definition of a subset of a user's environment that combines a CCSID and characters that are defined for a specific language and country.

local lock

A lock that provides intra-DB2 concurrency control, but not inter-DB2 concurrency control; that is, its scope is a single DB2.

local subsystem

The unique relational DBMS to which the user or application program is directly connected (in the case of DB2, by one of the DB2 attachment facilities).

location

The unique name of a database server. An application uses the location name to access a DB2 database server. A database alias can be used to override the location name when accessing a remote server.

location alias

Another name by which a database server identifies itself in the network. Applications can use this name to access a DB2 database server.

lock

A means of controlling concurrent events or access to data. DB2 locking is performed by the IRLM.

lock duration

The interval over which a DB2 lock is held.

lock escalation

The promotion of a lock from a row, page, or LOB lock to a table space lock because the number of page locks that are concurrently held on a given resource exceeds a preset limit.

locking

The process by which the integrity of data is ensured. Locking prevents concurrent users from accessing inconsistent data. See also claim, drain, and latch.

lock mode

A representation for the type of access that concurrently running programs can have to a resource that a DB2 lock is holding.

lock object

The resource that is controlled by a DB2 lock.

lock promotion

The process of changing the size or mode of a DB2 lock to a higher, more restrictive level.

lock size

The amount of data that is controlled by a DB2 lock on table data; the value can be a row, a page, a LOB, a partition, a table, or a table space.

lock structure

A coupling facility data structure that is composed of a series of lock entries to support shared and exclusive locking for logical resources.

log

A collection of records that describe the events that occur during DB2 execution and that indicate their sequence. The information thus recorded is used for recovery in the event of a failure during DB2 execution.

log control interval definition

A suffix of the physical log record that tells how record segments are placed in the physical control interval.

logical claim

A claim on a logical partition of a nonpartitioning index.

logical index partition

The set of all keys that reference the same data partition.

logical lock (L-lock)

The lock type that transactions use to control intra- and inter-DB2 data concurrency between transactions. Contrast with physical lock (P-lock).

logically complete

A state in which the concurrent copy process is finished with the initialization of the target objects that are being copied. The target objects are available for update.

logical page list (LPL)

A list of pages that are in error and that cannot be referenced by applications until the pages are recovered. The page is in *logical error* because the actual media (coupling facility or disk) might not contain any errors. Usually a connection to the media has been lost.

logical partition

A set of key or RID pairs in a

nonpartitioning index that are associated with a particular partition.

logical recovery pending (LRECP)

The state in which the data and the index keys that reference the data are inconsistent.

logical unit (LU)

An access point through which an application program accesses the SNA network in order to communicate with another application program. See also LU name.

logical unit of work

The processing that a program performs between synchronization points.

logical unit of work identifier (LUWID)

A name that uniquely identifies a thread within a network. This name consists of a fully-qualified LU network name, an LUW instance number, and an LUW sequence number.

log initialization

The first phase of restart processing during which DB2 attempts to locate the current end of the log.

log record header (LRH)

A prefix, in every log record, that contains control information.

log record sequence number (LRSN)

An identifier for a log record that is associated with a data sharing member. DB2 uses the LRSN for recovery in the data sharing environment.

log truncation

A process by which an explicit starting RBA is established. This RBA is the point at which the next byte of log data is to be written.

LPL See logical page list.

LRECP

See logical recovery pending.

LRH See log record header.

LRSN See log record sequence number.

LU See logical unit.

LU name

Logical unit name, which is the name by which VTAM refers to a node in a network.

LUW See logical unit of work.

LUWID

See logical unit of work identifier.

mapping table

A table that the REORG utility uses to map the associations of the RIDs of data records in the original copy and in the shadow copy. This table is created by the user.

mass delete

The deletion of all rows of a table.

materialize

- The process of putting rows from a view or nested table expression into a work file for additional processing by a query.
- The placement of a LOB value into contiguous storage. Because LOB values can be very large, DB2 avoids materializing LOB data until doing so becomes absolutely necessary.

materialized query table

A table that is used to contain information that is derived and can be summarized from one or more source tables. Contrast with base table.

MB Megabyte (1 048 576 bytes).

MBCS See multibyte character set.

member name

The z/OS XCF identifier for a particular DB2 subsystem in a data sharing group.

menu A displayed list of available functions for selection by the operator. A menu is sometimes called a *menu panel*.

metalanguage

A language that is used to create other specialized languages.

migration

The process of converting a subsystem with a previous release of DB2 to an updated or current release. In this process, you can acquire the functions of the updated or current release without losing the data that you created on the previous release.

mixed data string

A character string that can contain both single-byte and double-byte characters.

mode name

A VTAM name for the collection of physical and logical characteristics and attributes of a session.

modify locks

An L-lock or P-lock with a MODIFY attribute. A list of these active locks is kept at all times in the coupling facility lock structure. If the requesting DB2 subsystem fails, that DB2 subsystem's modify locks are converted to retained locks.

multibyte character set (MBCS)

A character set that represents single characters with more than a single byte. UTF-8 is an example of an MBCS. Characters in UTF-8 can range from 1 to 4 bytes in DB2. Contrast with single-byte character set and double-byte character set. See also Unicode.

multidimensional analysis

The process of assessing and evaluating an enterprise on more than one level.

Multiple Virtual Storage (MVS)

An element of the z/OS operating system. This element is also called the Base Control Program (BCP).

multisite update

Distributed relational database processing in which data is updated in more than one location within a single unit of work.

multithreading

Multiple TCBs that are executing one copy of DB2 ODBC code concurrently (sharing a processor) or in parallel (on separate central processors).

MVS See Multiple Virtual Storage.

native SQL procedure

An SQL procedure that is processed by converting the procedural statements to a native representation that is stored in the database directory, as is done with other SQL statements. When a native SQL procedure is called, the native representation is loaded from the directory, and DB2 executes the procedure. Contrast with external procedure and external SQL procedure.

nested table expression

A fullselect in a FROM clause (surrounded by parentheses).

network identifier (NID)

The network ID that is assigned by IMS or CICS, or if the connection type is RRSAF, the RRS unit of recovery ID (URID).

new-function mode (NFM)

The normal mode of operation that exists after successful completion of a version-to-version migration. At this stage, all new functions of the new version are available for use. A DB2 data sharing group cannot coexist with members that are still at the prior version level, and fallback to a prior version is not supported. Contrast with compatibility mode, compatibility mode*, enabling-new-function mode, and enabling-new-function mode*.

NFM See new-function mode.

NID See network identifier.

node ID index

See XML node ID index.

nondeterministic function

A user-defined function whose result is not solely dependent on the values of the input arguments. That is, successive invocations with the same argument values can produce a different answer. This type of function is sometimes called a *variant* function. Contrast with deterministic function (sometimes called a *not-variant function*).

nonleaf page

A page that contains keys and page numbers of other pages in the index (either leaf or nonleaf pages). Nonleaf pages never point to actual data. Contrast with leaf page.

nonpartitioned index

An index that is not physically partitioned. Both partitioning indexes and secondary indexes can be nonpartitioned.

nonpartitioned secondary index (NPSI)

An index on a partitioned table space that is not the partitioning index and is not partitioned. Contrast with data-partitioned secondary index.

nonpartitioning index

See secondary index.

nonscrollable cursor

A cursor that can be moved only in a

forward direction. Nonscrollable cursors are sometimes called forward-only cursors or serial cursors.

normalization

A key step in the task of building a logical relational database design. Normalization helps you avoid redundancies and inconsistencies in your data. An entity is normalized if it meets a set of constraints for a particular normal form (first normal form, second normal form, and so on). Contrast with denormalization.

not-variant function

See deterministic function.

NPSI See nonpartitioned secondary index.

NUL The null character ('\0'), which is represented by the value X'00'. In C, this character denotes the end of a string.

null A special value that indicates the absence of information.

null terminator

In C, the value that indicates the end of a string. For EBCDIC, ASCII, and Unicode UTF-8 strings, the null terminator is a single-byte value (X'00'). For Unicode UTF-16 or UCS-2 (wide) strings, the null terminator is a double-byte value (X'0000').

ODBC

See Open Database Connectivity.

ODBC driver

A dynamically-linked library (DLL) that implements ODBC function calls and interacts with a data source.

OLAP See online analytical processing.

online analytical processing (OLAP)

The process of collecting data from one or many sources; transforming and analyzing the consolidated data quickly and interactively; and examining the results across different dimensions of the data by looking for patterns, trends, and exceptions within complex relationships of that data.

Open Database Connectivity (ODBC)

A Microsoft database application programming interface (API) for C that allows access to database management systems by using callable SQL. ODBC

does not require the use of an SQL preprocessor. In addition, ODBC provides an architecture that lets users add modules called *database drivers*, which link the application to their choice of database management systems at run time. This means that applications no longer need to be directly linked to the modules of all the database management systems that are supported.

ordinary identifier

An uppercase letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. An ordinary identifier must not be a reserved word.

ordinary token

A numeric constant, an ordinary identifier, a host identifier, or a keyword.

originating task

In a parallel group, the primary agent that receives data from other execution units (referred to as *parallel tasks*) that are executing portions of the query in parallel.

outer join

The result of a join operation that includes the matched rows of both tables that are being joined and preserves some or all of the unmatched rows of the tables that are being joined. See also join, equijoin, full outer join, inner join, left outer join, and right outer join.

overloaded function

A function name for which multiple function instances exist.

package

An object containing a set of SQL statements that have been statically bound and that is available for processing. A package is sometimes also called an *application package*.

package list

An ordered list of package names that may be used to extend an application plan.

package name

The name of an object that is used for an application package or an SQL procedure package. An application package is a bound version of a database request module (DBRM) that is created by a

BIND PACKAGE or REBIND PACKAGE command. An SQL procedural language package is created by a CREATE or ALTER PROCEDURE statement for a native SQL procedure. The name of a package consists of a location name, a collection ID, a package ID, and a version ID.

page

A unit of storage within a table space (4 KB, 8 KB, 16 KB, or 32 KB) or index space (4 KB, 8 KB, 16 KB, or 32 KB). In a table space, a page contains one or more rows of a table. In a LOB or XML table space, a LOB or XML value can span more than one page, but no more than one LOB or XML value is stored on a page.

page set

Another way to refer to a table space or index space. Each page set consists of a collection of VSAM data sets.

page set recovery pending (PSRCP)

A restrictive state of an index space. In this case, the entire page set must be recovered. Recovery of a logical part is prohibited.

panel

A predefined display image that defines the locations and characteristics of display fields on a display surface (for example, a *menu panel*).

parallel complex

A cluster of machines that work together to handle multiple transactions and applications.

parallel group

A set of consecutive operations that execute in parallel and that have the same number of parallel tasks.

parallel I/O processing

A form of I/O processing in which DB2 initiates multiple concurrent requests for a single user query and performs I/O processing concurrently (in *parallel*) on multiple data partitions.

parallelism assistant

In Sysplex query parallelism, a DB2 subsystem that helps to process parts of a parallel query that originates on another DB2 subsystem in the data sharing group.

parallelism coordinator

In Sysplex query parallelism, the DB2 subsystem from which the parallel query originates.

Parallel Sysplex

A set of z/OS systems that communicate and cooperate with each other through certain multisystem hardware components and software services to process customer workloads.

parallel task

The execution unit that is dynamically created to process a query in parallel. A parallel task is implemented by a z/OS service request block.

parameter marker

A question mark (?) that appears in a statement string of a dynamic SQL statement. The question mark can appear where a variable could appear if the statement string were a static SQL statement.

parameter-name

An SQL identifier that designates a parameter in a routine that is written by a user. Parameter names are required for SQL procedures and SQL functions, and they are used in the body of the routine to refer to the values of the parameters. Parameter names are optional for external routines.

parent key

A primary key or unique key in the parent table of a referential constraint. The values of a parent key determine the valid values of the foreign key in the referential constraint.

parent lock

For explicit hierarchical locking, a lock that is held on a resource that might have child locks that are lower in the hierarchy. A parent lock is usually the table space lock or the partition intent lock. See also child lock.

parent row

A row whose primary key value is the foreign key value of a dependent row.

parent table

A table whose primary key is referenced by the foreign key of a dependent table.

parent table space

A table space that contains a parent table. A table space containing a dependent of that table is a dependent table space.

participant

An entity other than the commit coordinator that takes part in the commit process. The term participant is synonymous with agent in SNA.

partition

A portion of a page set. Each partition corresponds to a single, independently extendable data set. The maximum size of a partition depends on the number of partitions in the partitioned page set. All partitions of a given page set have the same maximum size.

partition-by-growth table space

A table space whose size can grow to accommodate data growth. DB2 for z/OS manages partition-by-growth table spaces by automatically adding new data sets when the database needs more space to satisfy an insert operation. Contrast with range-partitioned table space. See also universal table space.

partitioned data set (PDS)

A data set in disk storage that is divided into partitions, which are called members. Each partition can contain a program, part of a program, or data. A program library is an example of a partitioned data set.

partitioned index

An index that is physically partitioned. Both partitioning indexes and secondary indexes can be partitioned.

partitioned page set

A partitioned table space or an index space. Header pages, space map pages, data pages, and index pages reference data only within the scope of the partition.

partitioned table space

A table space that is based on a single table and that is subdivided into partitions, each of which can be processed independently by utilities. Contrast with segmented table space and universal table space.

partitioning index

An index in which the leftmost columns

- are the partitioning columns of the table. The index can be partitioned or nonpartitioned.
- partner logical unit**
An access point in the SNA network that is connected to the local DB2 subsystem by way of a VTAM conversation.
- path** See SQL path.
- PDS** See partitioned data set.
- physical consistency**
The state of a page that is not in a partially changed state.
- physical lock (P-lock)**
A type of lock that DB2 acquires to provide consistency of data that is cached in different DB2 subsystems. Physical locks are used only in data sharing environments. Contrast with logical lock (L-lock).
- physically complete**
The state in which the concurrent copy process is completed and the output data set has been created.
- piece** A data set of a nonpartitioned page set.
- plan** See application plan.
- plan allocation**
The process of allocating DB2 resources to a plan in preparation for execution.
- plan member**
The bound copy of a DBRM that is identified in the member clause.
- plan name**
The name of an application plan.
- P-lock** See physical lock.
- point of consistency**
A time when all recoverable data that an application accesses is consistent with other data. The term point of consistency is synonymous with sync point or commit point.
- policy** See CFRM policy.
- postponed abort UR**
A unit of recovery that was in-flight or in-abort, was interrupted by system failure or cancellation, and did not complete backout during restart.
- precision**
In SQL, the total number of digits in a decimal number (called the *size* in the C language). In the C language, the number of digits to the right of the decimal point (called the *scale* in SQL). The DB2 information uses the SQL terms.
- precompilation**
A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that are recognized by the host language compiler. Output from this precompilation includes source code that can be submitted to the compiler and the database request module (DBRM) that is input to the bind process.
- predicate**
An element of a search condition that expresses or implies a comparison operation.
- prefix** A code at the beginning of a message or record.
- preformat**
The process of preparing a VSAM linear data set for DB2 use, by writing specific data patterns.
- prepare**
The first phase of a two-phase commit process in which all participants are requested to prepare for commit.
- prepared SQL statement**
A named object that is the executable form of an SQL statement that has been processed by the PREPARE statement.
- primary authorization ID**
The authorization ID that is used to identify the application process to DB2.
- primary group buffer pool**
For a duplexed group buffer pool, the structure that is used to maintain the coherency of cached data. This structure is used for page registration and cross-invalidation. The z/OS equivalent is *old* structure. Compare with secondary group buffer pool.
- primary index**
An index that enforces the uniqueness of a primary key.
- primary key**
In a relational database, a unique, nonnull key that is part of the definition of a

table. A table cannot be defined as a parent unless it has a unique key or primary key.

principal

An entity that can communicate securely with another entity. In Kerberos, principals are represented as entries in the Kerberos registry database and include users, servers, computers, and others.

principal name

The name by which a principal is known to the DCE security services.

privilege

The capability of performing a specific function, sometimes on a specific object. See also explicit privilege.

privilege set

- For the installation SYSADM ID, the set of all possible privileges.
- For any other authorization ID, including the PUBLIC authorization ID, the set of all privileges that are recorded for that ID in the DB2 catalog.

process

In DB2, the unit to which DB2 allocates resources and locks. Sometimes called an application process, a process involves the execution of one or more programs. The execution of an SQL statement is always associated with some process. The means of initiating and terminating a process are dependent on the environment.

program

A single, compilable collection of executable statements in a programming language.

program temporary fix (PTF)

A solution or bypass of a problem that is diagnosed as a result of a defect in a current unaltered release of a licensed program. An authorized program analysis report (APAR) fix is corrective service for an existing problem. A PTF is preventive service for problems that might be encountered by other users of the product. A PTF is *temporary*, because a permanent fix is usually not incorporated into the product until its next release.

protected conversation

A VTAM conversation that supports two-phase commit flows.

PSRCP

See page set recovery pending.

PTF

See program temporary fix.

QSAM

See queued sequential access method.

query

A component of certain SQL statements that specifies a result table.

query block

The part of a query that is represented by one of the FROM clauses. Each FROM clause can have multiple query blocks, depending on DB2 processing of the query.

query CP parallelism

Parallel execution of a single query, which is accomplished by using multiple tasks. See also Sysplex query parallelism.

query I/O parallelism

Parallel access of data, which is accomplished by triggering multiple I/O requests within a single query.

queued sequential access method (QSAM)

An extended version of the basic sequential access method (BSAM). When this method is used, a queue of data blocks is formed. Input data blocks await processing, and output data blocks await transfer to auxiliary storage or to an output device.

quiesce point

A point at which data is consistent as a result of running the DB2 QUIESCE utility.

RACF

Resource Access Control Facility. A component of the z/OS Security Server.

range-partitioned table space

A type of universal table space that is based on partitioning ranges and that contains a single table. Contrast with partition-by-growth table space. See also universal table space.

RBA

See relative byte address.

RCT

See resource control table.

RDO

See resource definition online.

read stability (RS)

An isolation level that is similar to repeatable read but does not completely isolate an application process from all other concurrently executing application

processes. See also cursor stabilityrepeatable read, and uncommitted read.

rebind

The creation of a new application plan for an application program that has been bound previously. If, for example, you have added an index for a table that your application accesses, you must rebind the application in order to take advantage of that index.

rebuild

The process of reallocating a coupling facility structure. For the shared communications area (SCA) and lock structure, the structure is repopulated; for the group buffer pool, changed pages are usually cast out to disk, and the new structure is populated only with changed pages that were not successfully cast out.

record The storage representation of a row or other data.

record identifier (RID)

A unique identifier that DB2 uses to identify a row of data in a table. Compare with row identifier.

record identifier (RID) pool

An area of main storage that is used for sorting record identifiers during list-prefetch processing.

record length

The sum of the length of all the columns in a table, which is the length of the data as it is physically stored in the database. Records can be fixed length or varying length, depending on how the columns are defined. If all columns are fixed-length columns, the record is a fixed-length record. If one or more columns are varying-length columns, the record is a varying-length record.

Recoverable Resource Manager Services attachment facility (RRSAF)

A DB2 subcomponent that uses Resource Recovery Services to coordinate resource commitment between DB2 and all other resource managers that also use RRS in a z/OS system.

recovery

The process of rebuilding databases after a system failure.

recovery log

A collection of records that describes the events that occur during DB2 execution and indicates their sequence. The recorded information is used for recovery in the event of a failure during DB2 execution.

recovery manager

A subcomponent that supplies coordination services that control the interaction of DB2 resource managers during commit, abort, checkpoint, and restart processes. The recovery manager also supports the recovery mechanisms of other subsystems (for example, IMS) by acting as a participant in the other subsystem's process for protecting data that has reached a point of consistency.

A coordinator or a participant (or both), in the execution of a two-phase commit, that can access a recovery log that maintains the state of the logical unit of work and names the immediate upstream coordinator and downstream participants.

recovery pending (RECP)

A condition that prevents SQL access to a table space that needs to be recovered.

recovery token

An identifier for an element that is used in recovery (for example, NID or URID).

RECP See recovery pending.

redo A state of a unit of recovery that indicates that changes are to be reapplied to the disk media to ensure data integrity.

reentrant code

Executable code that can reside in storage as one shared copy for all threads. Reentrant code is not self-modifying and provides separate storage areas for each thread. See also threadsafe.

referential constraint

The requirement that nonnull values of a designated foreign key are valid only if they equal values of the primary key of a designated table.

referential cycle

A set of referential constraints such that each base table in the set is a descendent of itself. The tables that are involved in a referential cycle are ordered so that each

| table is a descendent of the one before it,
| and the first table is a descendent of the
| last table.

referential integrity

The state of a database in which all values of all foreign keys are valid. Maintaining referential integrity requires the enforcement of referential constraints on all operations that change the data in a table on which the referential constraints are defined.

referential structure

A set of tables and relationships that includes at least one table and, for every table in the set, all the relationships in which that table participates and all the tables to which it is related.

refresh age

The time duration between the current time and the time during which a materialized query table was last refreshed.

registry

See registry database.

registry database

A database of security information about principals, groups, organizations, accounts, and security policies.

relational database

A database that can be perceived as a set of tables and manipulated in accordance with the relational model of data.

relational database management system (RDBMS)

A collection of hardware and software that organizes and provides access to a relational database.

| **relational schema**

| See SQL schema.

relationship

A defined connection between the rows of a table or the rows of two tables. A relationship is the internal representation of a referential constraint.

relative byte address (RBA)

The offset of a data record or control interval from the beginning of the storage space that is allocated to the data set or file to which it belongs.

remigration

The process of returning to a current release of DB2 following a fallback to a previous release. This procedure constitutes another migration process.

remote

Any object that is maintained by a remote DB2 subsystem (that is, by a DB2 subsystem other than the local one). A *remote view*, for example, is a view that is maintained by a remote DB2 subsystem. Contrast with local.

remote subsystem

Any relational DBMS, except the local subsystem, with which the user or application can communicate. The subsystem need not be remote in any physical sense, and might even operate on the same processor under the same z/OS system.

reoptimization

The DB2 process of reconsidering the access path of an SQL statement at run time; during reoptimization, DB2 uses the values of host variables, parameter markers, or special registers.

| **reordered row format**

| A row format that facilitates improved
| performance in retrieval of rows that have
| varying-length columns. DB2 rearranges
| the column order, as defined in the
| CREATE TABLE statement, so that the
| fixed-length columns are stored at the
| beginning of the row and the
| varying-length columns are stored at the
| end of the row. Contrast with basic row
| format.

REORG pending (REORP)

A condition that restricts SQL access and most utility access to an object that must be reorganized.

REORP

See REORG pending.

repeatable read (RR)

The isolation level that provides maximum protection from other executing application programs. When an application program executes with repeatable read protection, rows that the program references cannot be changed by other programs until the program reaches

a commit point. See also cursor stability, read stability, and uncommitted read.

repeating group

A situation in which an entity includes multiple attributes that are inherently the same. The presence of a repeating group violates the requirement of first normal form. In an entity that satisfies the requirement of first normal form, each attribute is independent and unique in its meaning and its name. See also normalization.

replay detection mechanism

A method that allows a principal to detect whether a request is a valid request from a source that can be trusted or whether an untrustworthy entity has captured information from a previous exchange and is replaying the information exchange to gain access to the principal.

request commit

The vote that is submitted to the prepare phase if the participant has modified data and is prepared to commit or roll back.

requester

The source of a request to access data at a remote server. In the DB2 environment, the requester function is provided by the distributed data facility.

resource

The object of a lock or claim, which could be a table space, an index space, a data partition, an index partition, or a logical partition.

resource allocation

The part of plan allocation that deals specifically with the database resources.

resource control table

A construct of previous versions of the CICS attachment facility that defines authorization and access attributes for transactions or transaction groups. Beginning in CICS Transaction Server Version 1.3, resources are defined by using resource definition online instead of the resource control table. See also resource definition online.

resource definition online (RDO)

The recommended method of defining resources to CICS by creating resource definitions interactively, or by using a utility, and then storing them in the CICS

definition data set. In earlier releases of CICS, resources were defined by using the resource control table (RCT), which is no longer supported.

resource limit facility (RLF)

A portion of DB2 code that prevents dynamic manipulative SQL statements from exceeding specified time limits. The resource limit facility is sometimes called the governor.

resource limit specification table (RLST)

A site-defined table that specifies the limits to be enforced by the resource limit facility.

resource manager

- A function that is responsible for managing a particular resource and that guarantees the consistency of all updates made to recoverable resources within a logical unit of work. The resource that is being managed can be physical (for example, disk or main storage) or logical (for example, a particular type of system service).
- A participant, in the execution of a two-phase commit, that has recoverable resources that could have been modified. The resource manager has access to a recovery log so that it can commit or roll back the effects of the logical unit of work to the recoverable resources.

restart pending (RESTP)

A restrictive state of a page set or partition that indicates that restart (backout) work needs to be performed on the object.

RESTP

See restart pending.

result set

The set of rows that a stored procedure returns to a client application.

result set locator

A 4-byte value that DB2 uses to uniquely identify a query result set that a stored procedure returns.

result table

The set of rows that are specified by a SELECT statement.

retained lock

A MODIFY lock that a DB2 subsystem

was holding at the time of a subsystem failure. The lock is retained in the coupling facility lock structure across a DB2 for z/OS failure.

RID See record identifier.

RID pool

See record identifier pool.

right outer join

The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of the second join operand. See also join, equijoin, full outer join, inner join, left outer join, and outer join.

RLF See resource limit facility.

RLST See resource limit specification table.

role A database entity that groups together one or more privileges and that can be assigned to a primary authorization ID or to PUBLIC. The role is available only in a trusted context.

rollback

The process of restoring data that was changed by SQL statements to the state at its last commit point. All locks are freed. Contrast with commit.

root page

The index page that is at the highest level (or the beginning point) in an index.

routine

A database object that encapsulates procedural logic and SQL statements, is stored on the database server, and can be invoked from an SQL statement or by using the CALL statement. The main classes of routines are procedures and functions.

row The horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

row identifier (ROWID)

A value that uniquely identifies a row. This value is stored with the row and never changes.

row lock

A lock on a single row of data.

row-positioned fetch orientation

The specification of the desired placement

of the cursor as part of a FETCH statement, with respect to a single row (for example, NEXT, LAST, or ABSOLUTE *n*). Contrast with rowset-positioned fetch orientation.

rowset

A set of rows for which a cursor position is established.

rowset cursor

A cursor that is defined so that one or more rows can be returned as a rowset for a single FETCH statement, and the cursor is positioned on the set of rows that is fetched.

rowset-positioned fetch orientation

The specification of the desired placement of the cursor as part of a FETCH statement, with respect to a rowset (for example, NEXT ROWSET, LAST ROWSET, or ROWSET STARTING AT ABSOLUTE *n*). Contrast with row-positioned fetch orientation.

row trigger

A trigger that is defined with the trigger granularity FOR EACH ROW.

RRSAF

See Recoverable Resource Manager Services attachment facility.

RS

See read stability.

savepoint

A named entity that represents the state of data and schemas at a particular point in time within a unit of work.

SBCS See single-byte character set.

SCA See shared communications area.

scalar function

An SQL operation that produces a single value from another value and is expressed as a function name, followed by a list of arguments that are enclosed in parentheses.

scale

In SQL, the number of digits to the right of the decimal point (called the precision in the C language). The DB2 information uses the SQL definition.

schema

The organization or structure of a database.

A collection of, and a way of qualifying, database objects such as tables, views, routines, indexes or triggers that define a database. A database schema provides a logical classification of database objects.

scrollability

The ability to use a cursor to fetch in either a forward or backward direction. The FETCH statement supports multiple fetch orientations to indicate the new position of the cursor. See also fetch orientation.

scrollable cursor

A cursor that can be moved in both a forward and a backward direction.

search condition

A criterion for selecting rows from a table. A search condition consists of one or more predicates.

secondary authorization ID

An authorization ID that has been associated with a primary authorization ID by an authorization exit routine.

secondary group buffer pool

For a duplexed group buffer pool, the structure that is used to back up changed pages that are written to the primary group buffer pool. No page registration or cross-invalidation occurs using the secondary group buffer pool. The z/OS equivalent is *new* structure.

secondary index

A nonpartitioning index that is useful for enforcing a uniqueness constraint, for clustering data, or for providing access paths to data for queries. A secondary index can be partitioned or nonpartitioned. See also data-partitioned secondary index (DPSI) and nonpartitioned secondary index (NPSI).

section

The segment of a plan or package that contains the executable structures for a single SQL statement. For most SQL statements, one section in the plan exists for each SQL statement in the source program. However, for cursor-related statements, the DECLARE, OPEN, FETCH, and CLOSE statements reference the same section because they each refer to the SELECT statement that is named in the DECLARE CURSOR statement. SQL

statements such as COMMIT, ROLLBACK, and some SET statements do not use a section.

| **security label**

| A classification of users' access to objects
| or data rows in a multilevel security
| environment."

segment

A group of pages that holds rows of a single table. See also segmented table space.

segmented table space

A table space that is divided into equal-sized groups of pages called segments. Segments are assigned to tables so that rows of different tables are never stored in the same segment. Contrast with partitioned table space and universal table space.

self-referencing constraint

A referential constraint that defines a relationship in which a table is a dependent of itself.

self-referencing table

A table with a self-referencing constraint.

sensitive cursor

A cursor that is sensitive to changes that are made to the database after the result table has been materialized.

sequence

A user-defined object that generates a sequence of numeric values according to user specifications.

sequential data set

A non-DB2 data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Several of the DB2 database utilities require sequential data sets.

sequential prefetch

A mechanism that triggers consecutive asynchronous I/O operations. Pages are fetched before they are required, and several pages are read with a single I/O operation.

serialized profile

A Java object that contains SQL statements and descriptions of host variables. The SQLJ translator produces a serialized profile for each connection context.

server The target of a request from a remote requester. In the DB2 environment, the server function is provided by the distributed data facility, which is used to access DB2 data from remote applications.

service class
An eight-character identifier that is used by the z/OS Workload Manager to associate user performance goals with a particular DDF thread or stored procedure. A service class is also used to classify work on parallelism assistants.

service request block
A unit of work that is scheduled to execute.

session
A link between two nodes in a VTAM network.

session protocols
The available set of SNA communication requests and responses.

set operator
The SQL operators UNION, EXCEPT, and INTERSECT corresponding to the relational operators union, difference, and intersection. A set operator derives a result table by combining two other result tables.

shared communications area (SCA)
A coupling facility list structure that a DB2 data sharing group uses for inter-DB2 communication.

share lock
A lock that prevents concurrently executing application processes from changing data, but not from reading data. Contrast with exclusive lock.

shift-in character
A special control character (X'0F') that is used in EBCDIC systems to denote that the subsequent bytes represent SBCS characters. See also shift-out character.

shift-out character
A special control character (X'0E') that is used in EBCDIC systems to denote that the subsequent bytes, up to the next shift-in control character, represent DBCS characters. See also shift-in character.

sign-on
A request that is made on behalf of an individual CICS or IMS application

process by an attachment facility to enable DB2 to verify that it is authorized to use DB2 resources.

simple page set
A nonpartitioned page set. A simple page set initially consists of a single data set (page set piece). If and when that data set is extended to 2 GB, another data set is created, and so on, up to a total of 32 data sets. DB2 considers the data sets to be a single contiguous linear address space containing a maximum of 64 GB. Data is stored in the next available location within this address space without regard to any partitioning scheme.

simple table space
A table space that is neither partitioned nor segmented. Creation of simple table spaces is not supported in DB2 Version 9.1 for z/OS. Contrast with partitioned table space, segmented table space, and universal table space.

single-byte character set (SBCS)
A set of characters in which each character is represented by a single byte. Contrast with double-byte character set or multibyte character set.

single-precision floating point number
A 32-bit approximate representation of a real number.

SMP/E
See System Modification Program/Extended.

SNA See Systems Network Architecture.

SNA network
The part of a network that conforms to the formats and protocols of Systems Network Architecture (SNA).

socket A callable TCP/IP programming interface that TCP/IP network applications use to communicate with remote TCP/IP partners.

sourced function
A function that is implemented by another built-in or user-defined function that is already known to the database manager. This function can be a scalar function or an aggregate function; it returns a single value from a set of values

(for example, MAX or AVG). Contrast with built-in function, external function, and SQL function.

source program

A set of host language statements and SQL statements that is processed by an SQL precompiler.

source table

A table that can be a base table, a view, a table expression, or a user-defined table function.

source type

An existing type that DB2 uses to represent a distinct type.

space A sequence of one or more blank characters.

spatial column

A column in a table that is defined using one of the spatial data types provided by IBM Spatial Support for DB2 for z/OS.

spatial data

Data that is made up of coordinates that identify a geographic location or geographic region.

spatial function

A function provided by IBM Spatial Support for DB2 for z/OS that performs various operations on spatial data.

spatial reference system

A set of parameters that includes coordinates that define the maximum possible extent of space that is referenced by a given range of coordinates, an identifier of the coordinate system from which the coordinates are derived, and numbers that convert coordinates into positive integers to improve performance when the coordinates are processed.

special register

A storage area that DB2 defines for an application process to use for storing information that can be referenced in SQL statements. Examples of special registers are SESSION_USER and CURRENT DATE.

specific function name

A particular user-defined function that is known to the database manager by its specific name. Many specific user-defined functions can have the same function name. When a user-defined function is

defined to the database, every function is assigned a specific name that is unique within its schema. Either the user can provide this name, or a default name is used.

SPUFI See SQL Processor Using File Input.

SQL See Structured Query Language.

SQL authorization ID (SQL ID)

The authorization ID that is used for checking dynamic SQL statements in some situations.

SQLCA

See SQL communication area.

SQL communication area (SQLCA)

A structure that is used to provide an application program with information about the execution of its SQL statements.

SQL connection

An association between an application process and a local or remote application server or database server.

SQLDA

See SQL descriptor area.

SQL descriptor area (SQLDA)

A structure that describes input variables, output variables, or the columns of a result table.

SQL escape character

The symbol that is used to enclose an SQL delimited identifier. This symbol is the double quotation mark ("). See also escape character.

SQL function

A user-defined function in which the CREATE FUNCTION statement contains the source code. The source code is a single SQL expression that evaluates to a single value. The SQL user-defined function can return the result of an expression. See also built-in function, external function, and sourced function.

SQL ID

See SQL authorization ID.

SQLJ Structured Query Language (SQL) that is embedded in the Java programming language.

SQL path

An ordered list of schema names that are used in the resolution of unqualified

references to user-defined functions, distinct types, and stored procedures. In dynamic SQL, the SQL path is found in the CURRENT PATH special register. In static SQL, it is defined in the PATH bind option.

SQL procedure

A user-written program that can be invoked with the SQL CALL statement. An SQL procedure is written in the SQL procedural language. Two types of SQL procedures are supported: external SQL procedures and native SQL procedures. See also external procedure and native SQL procedure.

SQL processing conversation

Any conversation that requires access of DB2 data, either through an application or by dynamic query requests.

SQL Processor Using File Input (SPUFI)

A facility of the TSO attachment subcomponent that enables the DB2I user to execute SQL statements without embedding them in an application program.

SQL return code

Either SQLCODE or SQLSTATE.

SQL routine

A user-defined function or stored procedure that is based on code that is written in SQL.

SQL schema

A collection of database objects such as tables, views, indexes, functions, distinct types, schemas, or triggers that defines a database. An SQL schema provides a logical classification of database objects.

SQL statement coprocessor

An alternative to the DB2 precompiler that lets the user process SQL statements at compile time. The user invokes an SQL statement coprocessor by specifying a compiler option.

SQL string delimiter

A symbol that is used to enclose an SQL string constant. The SQL string delimiter is the apostrophe ('), except in COBOL applications, where the user assigns the symbol, which is either an apostrophe or a double quotation mark (").

SRB See service request block.

stand-alone

An attribute of a program that means that it is capable of executing separately from DB2, without using DB2 services.

star join

A method of joining a dimension column of a fact table to the key column of the corresponding dimension table. See also join, dimension, and star schema.

star schema

The combination of a fact table (which contains most of the data) and a number of dimension tables. See also star join, dimension, and dimension table.

statement handle

In DB2 ODBC, the data object that contains information about an SQL statement that is managed by DB2 ODBC. This includes information such as dynamic arguments, bindings for dynamic arguments and columns, cursor information, result values, and status information. Each statement handle is associated with the connection handle.

statement string

For a dynamic SQL statement, the character string form of the statement.

statement trigger

A trigger that is defined with the trigger granularity FOR EACH STATEMENT.

static cursor

A named control structure that does not change the size of the result table or the order of its rows after an application opens the cursor. Contrast with dynamic cursor.

static SQL

SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of variables that are specified by the statement might change).

storage group

A set of storage objects on which DB2 for z/OS data can be stored. A storage object can have an SMS data class, a management class, a storage class, and a list of volume serial numbers.

stored procedure

A user-written application program that can be invoked through the use of the SQL CALL statement. Stored procedures are sometimes called procedures.

string See binary string, character string, or graphic string.

strong typing

A process that guarantees that only user-defined functions and operations that are defined on a distinct type can be applied to that type. For example, you cannot directly compare two currency types, such as Canadian dollars and U.S. dollars. But you can provide a user-defined function to convert one currency to the other and then do the comparison.

structure

- A name that refers collectively to different types of DB2 objects, such as tables, databases, views, indexes, and table spaces.
- A construct that uses z/OS to map and manage storage on a coupling facility. See also cache structure, list structure, or lock structure.

Structured Query Language (SQL)

A standardized language for defining and manipulating data in a relational database.

structure owner

In relation to group buffer pools, the DB2 member that is responsible for the following activities:

- Coordinating rebuild, checkpoint, and damage assessment processing
- Monitoring the group buffer pool threshold and notifying castout owners when the threshold has been reached

subcomponent

A group of closely related DB2 modules that work together to provide a general function.

subject table

The table for which a trigger is created. When the defined triggering event occurs on this table, the trigger is activated.

subquery

A SELECT statement within the WHERE

or HAVING clause of another SQL statement; a nested SQL statement.

subselect

| That form of a query that includes only a
| SELECT clause, FROM clause, and
| optionally a WHERE clause, GROUP BY
| clause, HAVING clause, ORDER BY
| clause, or FETCH FIRST clause.

substitution character

A unique character that is substituted during character conversion for any characters in the source program that do not have a match in the target coding representation.

subsystem

A distinct instance of a relational database management system (RDBMS).

surrogate pair

A coded representation for a single character that consists of a sequence of two 16-bit code units, in which the first value of the pair is a high-surrogate code unit in the range U+D800 through U+DBFF, and the second value is a low-surrogate code unit in the range U+DC00 through U+DFFF. Surrogate pairs provide an extension mechanism for encoding 917 476 characters without requiring the use of 32-bit characters.

SVC dump

A dump that is issued when a z/OS or a DB2 functional recovery routine detects an error.

sync point

See commit point.

syncpoint tree

The tree of recovery managers and resource managers that are involved in a logical unit of work, starting with the recovery manager, that make the final commit decision.

synonym

| In SQL, an alternative name for a table or
| view. Synonyms can be used to refer only
| to objects at the subsystem in which the
| synonym is defined. A synonym cannot
| be qualified and can therefore not be used
| by other users. Contrast with alias.

Sysplex

See Parallel Sysplex.

Sysplex query parallelism

Parallel execution of a single query that is accomplished by using multiple tasks on more than one DB2 subsystem. See also query CP parallelism.

system administrator

The person at a computer installation who designs, controls, and manages the use of the computer system.

system agent

A work request that DB2 creates such as prefetch processing, deferred writes, and service tasks. See also allied agent.

system authorization ID

The primary DB2 authorization ID that is used to establish a trusted connection. A system authorization ID is derived from the system user ID that is provided by an external entity, such as a middleware server.

system conversation

The conversation that two DB2 subsystems must establish to process system messages before any distributed processing can begin.

System Modification Program/Extended (SMP/E)

A z/OS tool for making software changes in programming systems (such as DB2) and for controlling those changes.

Systems Network Architecture (SNA)

The description of the logical structure, formats, protocols, and operational sequences for transmitting information through and controlling the configuration and operation of networks.

table A named data object consisting of a specific number of columns and some number of unordered rows. See also base table or temporary table. Contrast with auxiliary table, clone table, materialized query table, result table, and transition table.

table-controlled partitioning

A type of partitioning in which partition boundaries for a partitioned table are controlled by values that are defined in the CREATE TABLE statement.

table function

A function that receives a set of arguments and returns a table to the SQL statement that references the function. A

table function can be referenced only in the FROM clause of a subselect.

table locator

A mechanism that allows access to trigger tables in SQL or from within user-defined functions. A table locator is a fullword integer value that represents a transition table.

table space

A page set that is used to store the records in one or more tables. See also partitioned table space, segmented table space, and universal table space.

table space set

A set of table spaces and partitions that should be recovered together for one of the following reasons:

- Each of them contains a table that is a parent or descendent of a table in one of the others.
- The set contains a base table and associated auxiliary tables.

A table space set can contain both types of relationships.

task control block (TCB)

A z/OS control block that is used to communicate information about tasks within an address space that is connected to a subsystem. See also address space connection.

TB Terabyte. A value of 1 099 511 627 776 bytes.

TCB See task control block.

TCP/IP

A network communication protocol that computer systems use to exchange information across telecommunication links.

TCP/IP port

A 2-byte value that identifies an end user or a TCP/IP network application within a TCP/IP host.

template

A DB2 utilities output data set descriptor that is used for dynamic allocation. A template is defined by the TEMPLATE utility control statement.

temporary table

A table that holds temporary data. Temporary tables are useful for holding

or sorting intermediate results from queries that contain a large number of rows. The two types of temporary table, which are created by different SQL statements, are the created temporary table and the declared temporary table. Contrast with result table. See also created temporary table and declared temporary table.

thread See DB2 thread.

threadsafe

A characteristic of code that allows multithreading both by providing private storage areas for each thread, and by properly serializing shared (global) storage areas.

three-part name

The full name of a table, view, or alias. It consists of a location name, a schema name, and an object name, separated by a period.

time A three-part value that designates a time of day in hours, minutes, and seconds.

timeout

Abnormal termination of either the DB2 subsystem or of an application because of the unavailability of resources. Installation specifications are set to determine both the amount of time DB2 is to wait for IRLM services after starting, and the amount of time IRLM is to wait if a resource that an application requests is unavailable. If either of these time specifications is exceeded, a timeout is declared.

Time-Sharing Option (TSO)

An option in z/OS that provides interactive time sharing from remote terminals.

timestamp

A seven-part value that consists of a date and time. The timestamp is expressed in years, months, days, hours, minutes, seconds, and microseconds.

trace A DB2 facility that provides the ability to monitor and collect DB2 monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

transaction

An atomic series of SQL statements that make up a logical unit of work. All of the

data modifications made during a transaction are either committed together as a unit or rolled back as a unit.

transaction lock

A lock that is used to control concurrent execution of SQL statements.

transaction program name

In SNA LU 6.2 conversations, the name of the program at the remote logical unit that is to be the other half of the conversation.

transition table

A temporary table that contains all the affected rows of the subject table in their state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the table of changed rows in the old state or the new state. Contrast with auxiliary table, base table, clone table, and materialized query table.

transition variable

A variable that contains a column value of the affected row of the subject table in its state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the set of old values or the set of new values.

tree structure

A data structure that represents entities in nodes, with a most one parent node for each node, and with only one root node.

trigger

A database object that is associated with a single base table or view and that defines a rule. The rule consists of a set of SQL statements that run when an insert, update, or delete database operation occurs on the associated base table or view.

trigger activation

The process that occurs when the trigger event that is defined in a trigger definition is executed. Trigger activation consists of the evaluation of the triggered action condition and conditional execution of the triggered SQL statements.

trigger activation time

An indication in the trigger definition of whether the trigger should be activated before or after the triggered event.

trigger body

The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. A trigger body is also called triggered SQL statements.

trigger cascading

The process that occurs when the triggered action of a trigger causes the activation of another trigger.

triggered action

The SQL logic that is performed when a trigger is activated. The triggered action consists of an optional triggered action condition and a set of triggered SQL statements that are executed only if the condition evaluates to true.

triggered action condition

An optional part of the triggered action. This Boolean condition appears as a WHEN clause and specifies a condition that DB2 evaluates to determine if the triggered SQL statements should be executed.

triggered SQL statements

The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. Triggered SQL statements are also called the trigger body.

trigger granularity

In SQL, a characteristic of a trigger, which determines whether the trigger is activated:

- Only once for the triggering SQL statement
- Once for each row that the SQL statement modifies

triggering event

The specified operation in a trigger definition that causes the activation of that trigger. The triggering event is comprised of a triggering operation (insert, update, or delete) and a subject table or view on which the operation is performed.

triggering SQL operation

The SQL operation that causes a trigger to be activated when performed on the subject table or view.

trigger package

A package that is created when a CREATE TRIGGER statement is executed. The package is executed when the trigger is activated.

trust attribute

An attribute on which to establish trust. A trusted relationship is established based on one or more trust attributes.

trusted connection

A database connection whose attributes match the attributes of a unique trusted context defined at the DB2 database server.

trusted connection reuse

The ability to switch the current user ID on a trusted connection to a different user ID.

trusted context

A database security object that enables the establishment of a trusted relationship between a DB2 database management system and an external entity.

trusted context default role

A role associated with a trusted context. The privileges granted to the trusted context default role can be acquired only when a trusted connection based on the trusted context is established or reused.

trusted context user

A user ID to which switching the current user ID on a trusted connection is permitted.

trusted context user-specific role

A role that is associated with a specific trusted context user. It overrides the trusted context default role if the current user ID on the trusted connection matches the ID of the specific trusted context user.

trusted relationship

A privileged relationship between two entities such as a middleware server and a database server. This relationship allows for a unique set of interactions between the two entities that would be impossible otherwise.

TSO See Time-Sharing Option.

TSO attachment facility

A DB2 facility consisting of the DSN command processor and DB2I. Applications that are not written for the

CICS or IMS environments can run under the TSO attachment facility.

typed parameter marker

A parameter marker that is specified along with its target data type. It has the general form:

CAST(? AS data-type)

type 2 indexes

Indexes that are created on a release of DB2 after Version 7 or that are specified as type 2 indexes in Version 4 or later.

UCS-2 Universal Character Set, coded in 2 octets, which means that characters are represented in 16-bits per character.

UDF See user-defined function.

UDT User-defined data type. In DB2 for z/OS, the term distinct type is used instead of user-defined data type. See distinct type.

uncommitted read (UR)

The isolation level that allows an application to read uncommitted data. See also cursor stability, read stability, and repeatable read.

underlying view

The view on which another view is directly or indirectly defined.

undo A state of a unit of recovery that indicates that the changes that the unit of recovery made to recoverable DB2 resources must be backed out.

Unicode

A standard that parallels the ISO-10646 standard. Several implementations of the Unicode standard exist, all of which have the ability to represent a large percentage of the characters that are contained in the many scripts that are used throughout the world.

union An SQL operation that involves the UNION set operator, which combines the results of two SELECT statements. Unions are often used to merge lists of values that are obtained from two tables.

unique constraint

An SQL rule that no two values in a primary key, or in the key of a unique index, can be the same.

unique index

An index that ensures that no identical

key values are stored in a column or a set of columns in a table.

unit of recovery (UOR)

A recoverable sequence of operations within a single resource manager, such as an instance of DB2. Contrast with unit of work.

unit of work (UOW)

A recoverable sequence of operations within an application process. At any time, an application process is a single unit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations. In a multisite update operation, a single unit of work can include several *units of recovery*. Contrast with unit of recovery.

universal table space

A table space that is both segmented and partitioned. Contrast with partitioned table space, segmented table space, partition-by-growth table space, and range-partitioned table space.

unlock

The act of releasing an object or system resource that was previously locked and returning it to general availability within DB2.

untyped parameter marker

A parameter marker that is specified without its target data type. It has the form of a single question mark (?).

updatability

The ability of a cursor to perform positioned updates and deletes. The updatability of a cursor can be influenced by the SELECT statement and the cursor sensitivity option that is specified on the DECLARE CURSOR statement.

update hole

The location on which a cursor is positioned when a row in a result table is fetched again and the new values no longer satisfy the search condition. See also delete hole.

update trigger

A trigger that is defined with the triggering SQL operation update.

UR See uncommitted read.

user-defined data type (UDT)

See distinct type.

user-defined function (UDF)

A function that is defined to DB2 by using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A user-defined function can be an external function, a sourced function, or an SQL function. Contrast with built-in function.

user view

In logical data modeling, a model or representation of critical information that the business requires.

UTF-16

Unicode Transformation Format, 16-bit encoding form, which is designed to provide code values for over a million characters and a superset of UCS-2. The CCSID value for data in UTF-16 format is 1200. DB2 for z/OS supports UTF-16 in graphic data fields.

UTF-8

Unicode Transformation Format, 8-bit encoding form, which is designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208. DB2 for z/OS supports UTF-8 in mixed data fields.

value

The smallest unit of data that is manipulated in SQL.

variable

A data element that specifies a value that can be changed. A COBOL elementary data item is an example of a host variable. Contrast with constant.

variant function

See nondeterministic function.

varying-length string

A character, graphic, or binary string whose length varies within set limits. Contrast with fixed-length string.

version

A member of a set of similar programs, DBRMs, packages, or LOBs.

- **A version of a program** is the source code that is produced by precompiling the program. The program version is identified by the program name and a timestamp (consistency token).
- **A version of an SQL procedural language routine** is produced by

issuing the CREATE or ALTER PROCEDURE statement for a native SQL procedure.

- **A version of a DBRM** is the DBRM that is produced by precompiling a program. The DBRM version is identified by the same program name and timestamp as a corresponding program version.
- **A version of an application package** is the result of binding a DBRM within a particular database system. The application package version is identified by the same program name and consistency token as the DBRM.
- **A version of a LOB** is a copy of a LOB value at a point in time. The version number for a LOB is stored in the auxiliary index entry for the LOB.
- **A version of a record** is a copy of the record at a point in time.

view

A logical table that consists of data that is generated by a query. A view can be based on one or more underlying base tables or views, and the data in a view is determined by a SELECT statement that is run on the underlying base tables or views.

Virtual Storage Access Method (VSAM)

An access method for direct or sequential processing of fixed- and varying-length records on disk devices.

Virtual Telecommunications Access Method (VTAM)

An IBM licensed program that controls communication and the flow of data in an SNA network (in z/OS).

volatile table

A table for which SQL operations choose index access whenever possible.

VSAM

See Virtual Storage Access Method.

VTAM

See Virtual Telecommunications Access Method.

warm start

The normal DB2 restart process, which involves reading and processing log records so that data that is under the control of DB2 is consistent. Contrast with cold start.

WLM application environment

A z/OS Workload Manager attribute that is associated with one or more stored procedures. The WLM application environment determines the address space in which a given DB2 stored procedure runs.

WLM enclave

A construct that can span multiple dispatchable units (service request blocks and tasks) in multiple address spaces, allowing them to be reported on and managed by WLM as part of a single work request.

write to operator (WTO)

An optional user-coded service that allows a message to be written to the system console operator informing the operator of errors and unusual system conditions that might need to be corrected (in z/OS).

WTO See write to operator.

WTOR

Write to operator (WTO) with reply.

XCF See cross-system coupling facility.

XES See cross-system extended services.

XML See Extensible Markup Language.

XML attribute

A name-value pair within a tagged XML element that modifies certain features of the element.

XML column

A column of a table that stores XML values and is defined using the data type XML. The XML values that are stored in XML columns are internal representations of well-formed XML documents.

XML data type

A data type for XML values.

XML element

A logical structure in an XML document that is delimited by a start and an end tag. Anything between the start tag and the end tag is the content of the element.

XML index

An index on an XML column that provides efficient access to nodes within an XML document by providing index keys that are based on XML patterns.

XML lock

A column-level lock for XML data. The operation of XML locks is similar to the operation of LOB locks.

XML node

The smallest unit of valid, complete structure in a document. For example, a node can represent an element, an attribute, or a text string.

XML node ID index

An implicitly created index, on an XML table that provides efficient access to XML documents and navigation among multiple XML data rows in the same document.

XML pattern

A slash-separated list of element names, an optional attribute name (at the end), or kind tests, that describe a path within an XML document in an XML column. The pattern is a restrictive form of path expressions, and it selects nodes that match the specifications. XML patterns are specified to create indexes on XML columns in a database.

XML publishing function

A function that returns an XML value from SQL values. An XML publishing function is also known as an XML constructor.

XML schema

In XML, a mechanism for describing and constraining the content of XML files by indicating which elements are allowed and in which combinations. XML schemas are an alternative to document type definitions (DTDs) and can be used to extend functionality in the areas of data typing, inheritance, and presentation.

XML schema repository (XSR)

A repository that allows the DB2 database system to store XML schemas. When registered with the XSR, these objects have a unique identifier and can be used to validate XML instance documents.

XML serialization function

A function that returns a serialized XML string from an XML value.

XML table

An auxiliary table that is implicitly created when an XML column is added to

| a base table. This table stores the XML
| data, and the column in the base table
| points to it.

| **XML table space**

A table space that is implicitly created when an XML column is added to a base table. The table space stores the XML table. If the base table is partitioned, one partitioned table space exists for each XML column of data.

X/Open

An independent, worldwide open systems organization that is supported by most of the world's largest information systems suppliers, user organizations, and software companies. X/Open's goal is to increase the portability of applications by combining existing and emerging standards.

XRF See Extended Recovery Facility.

| **XSR** See XML schema repository.

| **zIIP** See IBM System z9 Integrated Processor.

z/OS An operating system for the System z product line that supports 64-bit real and virtual storage.

z/OS Distributed Computing Environment (z/OS DCE) A set of technologies that are provided by the Open Software Foundation to implement distributed computing.

Index

Special characters

- _ (underscore)
 - in DDL registration tables 314
- % (percent sign)
 - in DDL registration tables 314

Numerics

- 16-KB page size 75
- 32-KB page size 75
- 8-KB page size 75

A

- abend
 - AEY9 606
 - after SQLCODE -923 611
 - ASP7 606
 - backward log recovery 645
 - CICS
 - abnormal termination of application 606
 - scenario 611
 - transaction abends when disconnecting from DB2 445, 446
 - waits 607
 - current status rebuild 631
 - disconnects DB2 456
 - DXR122E 597
 - effects of 502
 - forward log recovery 639
 - IMS
 - U3047 605
 - U3051 605
 - IMS, procedure 602
 - IMS, scenario 605
 - IRLM
 - scenario 597
 - STOP command 430
 - STOP DB2 430
 - log
 - damage 626
 - initialization 629
 - lost information 651
 - page problem 649
 - restart 628
 - starting DB2 after 367
 - VVDS (VSAM volume data set)
 - destroyed 667
 - out of space 667
 - access control
 - authorization exit routine 764
 - closed application 322
 - DB2 subsystem
 - local 152, 267
 - process overview 249
 - RACF 151
 - remote 152, 273
 - external DB2 data sets 152
 - field level 187
 - internal DB2 data 149
 - access method services
 - bootstrap data set definition 498
 - commands
 - ALTER 32, 668
 - ALTER ADDVOLUMES 23, 32
 - ALTER REMOVEVOLUMES 23
 - DEFINE 32, 649
 - DEFINE CLUSTER 32
 - EXPORT 565
 - IMPORT 138, 649
 - PRINT 575
 - REPRO 575, 623
 - data set management 22, 32
 - delete damaged BSDS 621
 - redefine user work file 567
 - rename damaged BSDS 621
 - table space re-creation 649
- access profile, in RACF 250
 - accessibility
 - keyboard xviii
 - shortcut keys xviii
 - active log
 - data set
 - changing high-level qualifier 132
 - changing high-level qualifier for 131, 132, 133, 134, 135
 - copying with IDCAMS REPRO statement 499
 - effect of stopped 616
 - offloaded to archive log 481
 - VSAM linear 728
 - dual logging 481
 - offloading 482
 - problems 613
 - recovery procedure 613
 - truncation 483
 - writing 481
 - ADD VOLUMES clause of ALTER STOGROUP statement 89
 - adding XML columns 114
 - address space
 - started-task 252
 - stored procedures 253
 - ADMIN_COMMAND_DB2 stored procedure 841
 - ADMIN_INFO_SSID stored procedure 881
 - ADMIN_TASK_ADD 379
 - ADMIN_TASK_LIST function 387
 - ADMIN_TASK_REMOVE 396
 - ADMIN_TASK_STATUS function 392
 - administrative authority 173
 - administrative task scheduler
 - adding a task 375
 - ADMIN_TASK_ADD 379
 - ADMIN_TASK_REMOVE 396
 - architecture 402
 - data sharing environment 405
 - data sharing
 - specifying a scheduler 379
 - synchronization 398
 - disabling tracing 399
 - enabling tracing 399
 - how stored procedures are executed 411
 - interacting 375
 - JCL jobs 411

- administrative task scheduler (*continued*)
 - lifecycle 403
 - listing scheduled tasks 387
 - listing task status 391
 - multi-threaded execution of tasks 409
 - overview 375
 - protecting resources 408
 - protecting the interface 407
 - recovering the task list 399
 - removing a task 395
 - sample schedule definitions 377
 - scheduling capabilities 375
 - secure execution of tasks 408
 - security 406
 - SQL code returned by stored procedure 401
 - SQL code returned by user-defined table function 401
 - starting 397
 - stopping 398
 - task did not execute 400
 - task execution 409
 - task execution in a data sharing environment 412
 - task lists 404
 - troubleshooting 399
 - Unicode restrictions 411
 - user roles 407
- alias
 - ownership 203
 - qualified name 203
- ALL
 - clause of GRANT statement 167
- ALL PRIVILEGES clause
 - GRANT statement 171
- allocating space
 - table 32
- allocating storage
 - dictionary 83
 - index 78
 - table 75
- already-verified acceptance option 281
- ALTER ADDVOLUMES 35
- ALTER command of access method services 668
- ALTER command, access method services
 - FOR option 32
 - TO option 32
- ALTER DATABASE statement
 - options 87
 - usage 87
- ALTER FUNCTION statement
 - usage 130
- ALTER INDEX statement 61
- ALTER privilege
 - description 167
- ALTER PROCEDURE statement 129
- ALTER STOGROUP statement 88
- ALTER TABLE statement
 - AUDIT clause 343
 - default column value 96
 - description 94
- ALTER TABLESPACE statement
 - description 89
- ALTERIN privilege
 - description 170
- altering
 - materialized query table 115
 - tables 94
 - XML objects 130
- APPL statement
 - options
 - SECACPT 280
- application errors
 - backing out
 - without a quiesce point 601
- application plan
 - controlling use of DDL 311, 322
 - inoperative, when privilege is revoked 197
 - invalidated
 - dropping a table 120
 - dropping a view 124
 - dropping an index 128
 - when privilege is revoked 197
 - list of dependent objects 122
 - privileges
 - explicit 171
 - retrieving catalog information 228
- application program
 - coding SQL statements
 - error checking in IMS 371
 - internal integrity reports 351
 - recovery procedures
 - CICS 606
 - IMS 605
 - running
 - batch 372
 - CAF (call attachment facility) 373
 - CICS transactions 372
 - error recovery procedure 600
 - IMS 371
 - RRSAF (Resource Recovery Services attachment facility) 373
 - TSO online 369
 - security measures in 209
- application programmer
 - description 180
 - privileges 190
- application registration table (ART) 312
- archive log
 - BSDS 498
 - data set
 - changing high-level qualifier 132
 - changing high-level qualifier for 131, 132, 133, 134, 135
 - description 485
 - offloading 481
 - types 485
 - deleting 487
 - device type 485
 - dual logging 485
 - dynamic allocation of data sets 485
 - multivolume data sets 486
 - recovery procedure 618
 - retention period 487
 - SMS 486
 - writing 482
- ARCHIVE LOG command
 - cancels offloads 490
 - use 490
- ARCHIVE privilege
 - description 169
- archiving to disk volumes 485
- ARCHWTOR option of DSNZPxxx module 483
- ART (application registration table) 312
- ASUSER 308
- attachment request
 - come-from check 283

- attachment request *(continued)*
 - definition 280
 - translating IDs 281, 296
 - using secondary IDs 284
- AUDIT
 - clause of ALTER TABLE statement 343
 - clause of CREATE TABLE statement 343
 - option of START TRACE command 342
- audit trace
 - class descriptions 338
 - controlling 343
 - description 342
 - records 345
- auditing
 - access attempts 340
 - authorization IDs 344
 - classes of events 338, 339
 - in sample security plan
 - attempted access 162
 - payroll data 157
 - payroll updates 160
 - reporting trace records 345
 - security measures in force 345
 - table access 343
- authority
 - administrative 173
 - controlling access to
 - CICS 372
 - DB2 catalog and directory 179
 - DB2 commands 361
 - DB2 functions 361
 - IMS application program 371
 - TSO application program 370
 - description 149, 150, 167
 - explicitly granted 173
 - hierarchy 173
 - level SYS for z/OS command group 358
 - levels 361
 - types
 - DBADM 178
 - DBCTRL 178
 - DBMAINT 178
 - installation SYSADM 175
 - installation SYSOPR 177
 - PACKADM 177
 - SYSADM 176
 - SYSCTRL 177
 - SYSOPR 177
- authorization
 - data definition statements, to use 311
 - exit routines. 753
- authorization ID
 - auditing 344, 345
 - dynamic SQL, determining 216
 - exit routine input parameter 758
 - inbound from remote location 273
 - initial
 - connection processing 269
 - sign-on processing 271
 - package execution 206
 - primary
 - connection processing 268, 269
 - exit routine input 758
 - privileges exercised by 180
 - sign-on processing 271
 - retrieving catalog information 226

- authorization ID *(continued)*
 - role
 - privileges exercised by 180
 - routine, determining 215
 - secondary
 - attachment requests 284
 - connection processing 269
 - exit routine output 759, 778
 - identifying RACF groups 256
 - ownership held by 203
 - privileges exercised by 180
 - sign-on processing 272
- SQL
 - changing 166
 - exit routine output 759, 778
 - privileges exercised by 180
- translating
 - inbound IDs 281
 - outbound IDs 296
- automatic
 - data management 538
 - deletion of archive log data sets 487
 - restart function of z/OS 507
- auxiliary storage 22
- availability
 - recovering
 - data sets 550
 - page sets 550
 - recovery planning 531

B

- BACKOUT DURATION field of panel DSNTIPL 510
- backup
 - data set using DFSMSHsm 538
 - database
 - DSN1COPY 593
 - image copies 552
 - planning 531
 - moving data 559
 - system procedures 531
- BACKUP SYSTEM utility 30, 31
- backward log recovery phase
 - failure during 645
 - restart 506
- base table
 - distinctions from temporary tables 49
- base tables
 - creating 47
- basic direct access method (BDAM) 485
- basic sequential access method (BSAM) 485
- batch message processing (BMP) program 453
- batch processing
 - TSO 371
- BDAM (basic direct access method) 485
- BIND PACKAGE subcommand of DSN
 - options
 - DISABLE 210
 - ENABLE 210
 - OWNER 204
 - privileges for remote bind 211
- BIND PLAN subcommand of DSN
 - options
 - DISABLE 210
 - ENABLE 210
 - OWNER 204

- BIND privilege
 - description 171
- BINDADD privilege
 - description 169
- BINDAGENT privilege
 - description 169
 - naming plan or package owner 204
- binding
 - privileges needed 182
- bit data
 - altering subtype 119
- blank
 - column with a field procedure 797
- BMP (batch message processing) program
 - connecting from dependent regions 453
- bootstrap data set (BSDS) 169
- BSAM (basic sequential access method)
 - reading archive log data sets 485
- BSDS (bootstrap data set)
 - archive log information 498
 - changing high-level qualifier 132
 - changing high-level qualifier of 131, 132, 133, 134, 135
 - changing log inventory 499
 - defining 498
 - dual copies 498
 - dual recovery 623
 - failure symptoms 626
 - managing 498
 - recovery procedure 621
 - recovery scenario 648
 - registers log data 498
 - restart use 502
 - restoring from the archive log 623
 - single recovery 623
 - stand-alone log services role 739
- BSDS privilege
 - description 169
- buffer pool
 - use in logging 481
- BUFFERPOOL clause 38
- BUFFERPOOL privilege
 - description 172
- built-in functions for encryption 325

C

- cache for authorization IDs 208
- CAF (call attachment facility)
 - application program
 - running 373
 - submitting 373
- CANCEL THREAD command
 - CICS threads 445
 - disconnecting from TSO 441
 - use in controlling DB2 connections 466
- capturing changed data
 - altering a table for 118
- catalog alias
 - defining 132
- catalog tables
 - frequency of image copies 536, 537
 - retrieving information about
 - multiple grants 226
 - plans and packages 228
 - privileges 224
 - routines 228
 - SYSCOLAUTH 224
- catalog tables (*continued*)
 - SYSCOLUMNS
 - column description of a value 796
 - updated by DROP TABLE 120
 - SYSCONTEXT 224
 - SYSCONTEXTAUTHIDS 224
 - SYSCOPY
 - discarding records 544
 - holds image copy information 542
 - image copy in log 724
 - used by RECOVER utility 534
 - SYSCTXTRUSTATTRS 224
 - SYSDBAUTH 224
 - SYSINDEXES
 - dropping a table 122
 - SYSINDEXPART
 - space allocation information 35
 - SYSPACKAUTH 224
 - SYSPLANAUTH
 - plan authorization 224
 - SYSPLANDEP 122
 - SYSRESAUTH 224
 - SYSROUTINES
 - using EXTERNAL_SECURITY column of 261
 - SYSSTOGROUP
 - storage groups 22
 - SYSSTRINGS
 - establishing conversion procedure 792
 - SYSSYNONYMS 120
 - SYSTABAUTH
 - authorization information 224
 - dropping a table 122
 - SYSTABLEPART
 - table spaces associated with storage group 89
 - SYSTABLES
 - updated by DROP TABLE 120
 - SYSUSERAUTH 224
 - SYSVIEWDEP
 - view dependencies 122
 - SYSVOLUMES 22
 - views of 229
- catalog, DB2
 - authority for access 179
 - changing high-level qualifier 135
 - DSNDB06 database 542
 - point-in-time recovery 568
 - recovery 568
 - recovery procedure 666
- CD-ROM, books on 955
- CDB (communications database)
 - backing up 533
 - changing high-level qualifier 135
 - updating tables 283
- CHANGE command of IMS
 - purging residual recovery entries 447
- change log inventory utility
 - changing
 - BSDS 427, 499
 - control of data set access 334
 - change number of sessions (CNOS) 677
- CHANGE SUBSYS command of IMS 452
- CHARACTER data type
 - altering 119
- CHECK utilities
 - CHECK DATA 349
 - CHECK INDEX 349
 - CHECK LOB 349

- checkpoint
 - log records 723, 727
 - queue 512
- CI (control interval)
 - description 481, 485
 - reading 739
- CICS
 - commands
 - accessing databases 442
 - DSNC DISCONNECT 445
 - DSNC DISPLAY PLAN 444
 - DSNC DISPLAY TRANSACTION 444
 - DSNC STOP 446
 - response destination 360
 - used in DB2 environment 355
 - connecting to
 - controlling 447
 - disconnecting applications 445
 - thread 443
 - connecting to DB2
 - authorization IDs 372
 - connection processing 268
 - controlling 442
 - disconnecting applications 476
 - sample authorization routines 273
 - security 151, 249
 - sign-on processing 271
 - supplying secondary IDs 269
 - disconnecting from DB2 446
 - dynamic plan selection
 - exit routine 810
 - facilities 810
 - diagnostic trace 474
 - language interface module (DSNCLI)
 - running CICS applications 372
 - operating
 - entering DB2 commands 359
 - identify outstanding indoubt units 524
 - terminates AEY9 611
 - planning
 - environment 372
 - programming
 - applications 372
 - recovery procedures 606
 - application failure 606
 - attachment facility failure 611
 - CICS not operational 607
 - DB2 connection failure 608
 - indoubt resolution failure 608
 - starting a connection 443
 - two-phase commit 515
- CICS transaction invocation stored procedure 810
- CLONE keyword 595
- clone table 56
- clone tables
 - backup and recovery 595
- closed application
 - controlling access 311, 322
 - definition 311
- CNOS (change number of sessions)
 - failure 677
- coding
 - exit routines
 - general rules 810
 - parameters 812
- cold start
 - bypassing the damaged log 626
- cold start (*continued*)
 - recovery operations during 512
 - special situations 651
- collection, package
 - administrator 180
 - privileges on 169
- column
 - adding to a table 95
 - altering
 - default value 96
 - dropping from a table 120
- column description of a value 796
- column value descriptor (CVD) 798
- come-from check 283
- command prefix
 - messages 362
 - multi-character 358
 - usage 358
- command recognition character (CRC) 358
- commands
 - entering 355
 - operator 355, 361
 - prefixes 363
- commit
 - two-phase process 515
- communications database (CDB) 275, 287
- communications failure scenario 710
- COMPRESS clause
 - described 38
- conditional restart
 - control record
 - backward log recovery failure 646
 - current status rebuild failure 638
 - forward log recovery failure 644
 - log initialization failure 638
 - wrap-around queue 512
 - description 508
 - excessive loss of active log data, restart procedure 652
 - total loss of log, restart procedure 651
- connection
 - controlling CICS 442
 - controlling IMS 447
 - DB2
 - controlling commands 442
 - displaying
 - IMS activity 454
 - effect of lost, on restart 522
 - exit routine 269
 - exit routine. 753
 - IDs
 - cited in message DSNR007I 504
 - outstanding unit of recovery 504
 - used by IMS 371
 - used to identify a unit of recovery 600
 - processing 268
 - requests
 - exit point 755
 - initial primary authorization ID 268
 - invoking RACF 268
 - local 267
 - VTAM 251
- connection exit routine
 - debugging 762
 - default 269
 - description 753
 - performance considerations 762

- connection exit routine (*continued*)
 - sample
 - location 754
 - provides secondary IDs 269, 760
 - secondary authorization ID 269
 - using 269
 - writing 753
- connection processing
 - choosing for remote requests 280
 - initial primary authorization ID 269, 759
 - invoking RACF 268
 - supplying secondary IDs 269
 - usage 267
 - using exit routine 269
- consistency groups 704
- continuous operation
 - recovering table spaces and data sets 550
 - recovery planning 531
- control interval
 - sizing 21
- control interval (CI) 481
- control region, IMS 453
- conversation acceptance option 280, 281
- conversation-level security 280
- conversion procedure
 - description 792
 - writing 792
- coordinator
 - in multi-site update 517
 - in two-phase commit 515
- copy pools 31
 - SMS construct 30
- COPY privilege
 - description 169
- COPY utility
 - backing up 593
 - copying data from table space 552
 - DFSMSdss concurrent copy 554, 563
 - restoring data 593
- COPY-pending status
 - resetting 69
- copying
 - a DB2 subsystem 141
 - a package, privileges for 182, 211
 - a relational database 140
- correlation ID
 - CICS 608
 - duplicate 448, 608
 - identifier for connections from TSO 440
 - IMS 448
 - outstanding unit of recovery 504
 - RECOVER INDOUBT command 445, 451, 458
- CRC (command recognition character)
 - description 358
- CREATE DATABASE statement
 - privileges required 184
 - usage 19
- CREATE FUNCTION statement
 - usage 73
- CREATE GLOBAL TEMPORARY TABLE statement
 - distinctions from base tables 49
- CREATE IN privilege
 - description 169
- CREATE INDEX statement 61
 - privileges required 184
 - USING clause 35
- CREATE PROCEDURE statement 72
- CREATE STOGROUP statement
 - description 22
 - privileges required 184
 - VOLUMES(‘*’) attribute 22, 28
- CREATE TABLE statement 39
 - AUDIT clause 343
 - example 47
 - privileges required 184
 - test table 70
- CREATE TABLESPACE statement
 - creating a table space explicitly 37
 - deferring allocation of data sets 23
 - DEFINE NO clause 23, 37
 - defining table space explicitly 38
 - DSSIZE option 36
 - privileges required 184
 - USING STOGROUP clause 23, 37
- CREATE VIEW statement
 - privileges required 184
- CREATEALIAS privilege
 - description 169
- created temporary table
 - distinctions from base tables 49
- created temporary tables 48
 - creating 48
- CREATEDBA privilege
 - description 169
- CREATEDBC privilege
 - description 169
- CREATEIN privilege
 - description 170
- CREATESG privilege
 - description 169
- CREATETAB privilege
 - description 169
- CREATETMTAB privilege
 - description 169
- CREATETS privilege
 - description 169
- cron format 386
- current status rebuild
 - phase of restart 504
 - recovery procedure 628
- Customer Information Control System (CICS) 273, 355
- CVD (column value descriptor) 798, 800

D

- damage
 - heuristic 522
- data
 - access control
 - description 149
 - field-level 187
 - using option of START DB2 366
 - backing up 593
 - checking consistency of updates 349
 - coding
 - conversion procedures 792
 - date and time exit routines 788
 - edit exit routines 781
 - field procedures 795
 - compressing 83
 - consistency
 - ensuring 345
 - verifying 348, 351
 - definition control support 311

- data (*continued*)
 - encrypting 781
 - loading into tables 66
 - moving 140
 - restoring 593
- DATA CAPTURE clause
 - ALTER TABLE statement 118
- data class
 - assigning data sets to 36
 - SMS construct 36
- data compression 83
 - dictionary
 - description 83
 - estimating disk storage 83, 84
 - estimating virtual storage 85
 - edit routine 781
 - effect on log records 723
 - Huffman 781
 - logging 481
- data definition control support
 - bypassing 321
 - controlling by
 - application name 315
 - application name with exceptions 315
 - object name 317
 - object name with exceptions 318
 - description 311
 - registration tables 311
 - restarting 321
 - stopping 321
- Data Facility Product (DFSMSdfp) 138
- data mirroring 703
 - recovery with 705
- data set
 - adding 668
 - adding groups to control 334
 - backing up using DFSMS 554
 - changing high-level qualifier 131
 - control over creating 336
 - controlling access 334
 - copying 552
 - DB2-managed
 - description 23
 - extending 23, 24
 - extension failures 23
 - nonpartitioned spaces 23
 - partitioned spaces 23
 - primary space allocation 24, 27
 - secondary space allocation 24, 25, 27
 - extending 668
 - generic profiles 334
 - managing 32
 - access method services 32
 - migrating to DFSMSShsm 28
 - reasons for using DB2 20
 - using DFSMSShsm 28
 - with DB2 storage groups 20
 - your own 32
 - moving 141
 - with utilities 142
 - without utilities 141
 - naming 34
 - naming convention 32
 - number 32
 - partition of
 - partitioned partitioning index 32
 - partitioned secondary index 32
- data set (*continued*)
 - partition of (*continued*)
 - partitioned table space 32
 - recovering
 - using non-DB2 dump 575
 - renaming 547
 - table space, deferring allocation 23
- Data Set Services (DFSMSDss) 138
- data set, adding a 671
- data sets
 - managing 20
- data type
 - altering 120
 - implications 98
 - codes for numeric data 818
 - subtypes 119
- data type of attributes 7
- database
 - access thread
 - failure 675
 - security failures in 678
 - altering 87
 - design 87
 - backup
 - copying data 552
 - planning 531
 - balancing 347
 - controlling access 460
 - creating 19
 - dropping 20
 - DSNDB07 (work file database). 552
 - implementing 19
 - monitoring 415
 - page set control log records 728
 - privileges
 - administrator 180, 189
 - description 169
 - ownership 172
 - recovery
 - description 550
 - failure scenarios 660
 - planning 531
 - RECOVER TOCOPY 582
 - RECOVER TOLOGPOINT 582
 - RECOVER TORBA 582
 - starting 413
 - status information 415
 - stopping 422
- database controller privileges 189
- database design
 - implementing 19
- database exception table, log records
 - exception states 724
 - image copies of special table spaces 724
 - LPL 728
 - WEPR 728
- DataRefresher 72
- DATE FORMAT field of panel DSNTIPF 789
- date routine
 - DATE FORMAT field at installation 789
 - description 788
 - LOCAL DATE LENGTH field at installation 789
 - writing 788
- datetime
 - exit routine for. 788
 - format
 - table 788

- DB2
 - checking
 - inbound remote IDs 280
 - DB2 books online 955
 - DB2 coded format for numeric data 818
 - DB2 commands
 - authority 361
 - authorized for SYSOPR 361
 - commands
 - RESET INDOUBT 529
 - START DB2 366
 - START DDF 461
 - STOP DDF 472
 - STOP DDF MODE(SUSPEND) 461
 - description 355
 - destination of responses 360
 - entering from
 - CICS 359
 - DSN session 370
 - IMS 358
 - TSO 359
 - z/OS 358
 - users authorized to enter 361
 - DB2 DataPropagator
 - altering a table for 118
 - DB2 decoded procedure for numeric data 818
 - DB2 Information Center for z/OS solutions 955
 - DB2 system
 - recovering 581
 - restoring 581
 - DB2-defined extents 94
 - DB2-managed data sets
 - enlarging 670
 - DB2-managed objects, changing data set high-level
 - qualifier 137
 - DB2DB2 commands
 - commands
 - RECOVER INDOUBT 528
 - DB2I (DB2 Interactive)
 - description 369
 - used to connect from TSO 439
 - DBA (database administrator)
 - description 180
 - sample privileges 189
 - DBADM authority
 - description 178
 - DBCTRL authority
 - description 178
 - DBD01 directory table space
 - quiescing 564
 - recovery after conditional restart 561
 - recovery information 542
 - DBMAINT authority
 - description 178
 - DDF (distributed data facility)
 - controlling connections 460
 - resuming 461
 - suspending 461
 - DDL
 - performing 308
 - DDL, controlling usage of 311
 - decision
 - heuristic 522
 - DECLARE GLOBAL TEMPORARY TABLE statement
 - distinctions from base tables 49
 - declared temporary table
 - distinctions from base tables 49
 - declared temporary tables 48
 - creating 49
 - DECRYPT_BIT 326, 328
 - DECRYPT_CHAR 326, 328
 - DECRYPT_DB 326, 328
 - decryption 325
 - default database 38
 - default database (DSNDB04)
 - changing high-level qualifier 135
 - DEFER ALL field of panel DSNTIPS 510
 - DEFINE CLUSTER command, access method services
 - defining extents 32
 - example of use 32
 - LINEAR option 32
 - REUSE option 32
 - SHAREOPTIONS 32
 - DEFINE command of access method services
 - re-creating table space 649
 - redefine user work file 567
 - DEFINE command, access method services
 - FOR option 32
 - TO option 32
 - DEFINE NO clause
 - CREATE INDEX statement 61
 - described 45
 - definer, description 212
 - DELETE
 - command of access method services 649
 - statement
 - validation routine 786
 - DELETE CLUSTER command 35
 - DELETE privilege
 - description 167
 - deleting
 - archive logs 487
 - denormalizing tables 15
 - dependent
 - regions, disconnecting from 454
 - DFSLI000 (IMS language interface module) 371
 - DFSMS (Data Facility Storage Management Subsystem)
 - archive log data sets 486
 - backup 554
 - concurrent copy
 - backup 554
 - recovery 554
 - DFSMSdfp (Data Facility Product) 138
 - DFSMSdss (Data Set Services) 138
 - DFSMSdss RESTORE command
 - RECOVER utility 29
 - DFSMSHsm
 - assigning indexes to data classes 28
 - assigning table spaces to data classes 28
 - migrating data sets 28
 - recalling archive logs 29
 - using with BACKUP SYSTEM utility 30
 - DFSMSHsm (Data Facility Hierarchical Storage Manager)
 - advantages 28
 - backup 538
 - moving data sets 138
 - recovery 538
 - DFSMSsms
 - using with BACKUP SYSTEM utility 31
 - DFSxxx messages 362
 - dictionary 83
 - disk requirements 84
 - directory
 - authority for access 179

- directory (*continued*)
 - changing high-level qualifier 135
 - frequency of image copies 536, 537
 - order of recovering
 - I/O errors 666
 - point-in-time recovery 568
 - recovery 568
 - SYSLGRNX table
 - discarding records 544
 - records log RBA ranges 542
- disability xviii
- DISABLE option
 - limits plan and package use 210
- disaster recovery
 - archive logs, using 680, 685
 - data mirroring 703
 - image copies, using 680, 685
 - preparation 545
 - recovery with data mirroring 703
 - scenario 679
 - using a tracker site 694
- disconnecting
 - applications 445, 447
 - CICS from DB2, command 442
 - DB2 from TSO 441
- discretionary access checking 233
- disk
 - altering storage group assignment 88
 - disk storage
 - estimating 73
 - requirements 73
 - disk requirements 84
- DISPLAY command of IMS
 - SUBSYS option 447, 454
- DISPLAY DATABASE command
 - displaying LPL entries 420
 - SPACENAM option 417
 - status checking 350
- DISPLAY DDF command
 - displays connections to DDF 462
- DISPLAY FUNCTION SPECIFIC command
 - displaying statistics about external user-defined functions 425
- DISPLAY LOCATION command
 - controls connections to DDF 464
- DISPLAY OASN command of IMS
 - displaying RREs 452
 - produces OASN 602
- DISPLAY privilege
 - description 169
- DISPLAY PROCEDURE command
 - example 468
- DISPLAY THREAD command
 - extensions to control DDF connections
 - DETAIL option 434
 - LOCATION option 433
 - LUWID option 435
 - options
 - DETAIL 434
 - LOCATION 433
 - LUWID 435
 - TYPE (INDOUBT) 608
 - shows IMS threads 449, 454
- DISPLAY TRACE command
 - AUDIT option 343
- DISPLAY UTILITY command
 - data set control log record 723
- DISPLAYDB privilege
 - description 169
- displaying
 - indoubt units of recovery 451, 608
 - information about
 - originating threads 440
 - parallel threads 440
 - postponed units of recovery 452
- distinct type
 - privileges of ownership 172
- DISTINCT TYPE privilege, description 171
- distributed data
 - controlling connections 460
- DL/I
 - loading data 72
- down-level detection
 - controlling 662
 - LEVEL UPDATE FREQ field of panel DSNTIPN 662
- down-level page sets 662
- DRDA access
 - security mechanisms 273
- DROP
 - statement
 - TABLE 120
 - TABLESPACE 92
- DROP privilege
 - description 169
- DROP statement 72, 73
 - usage 20, 60
- DROPIN privilege
 - description 170
- dropping
 - columns from a table 120
 - privileges needed for package 182
 - table spaces 92
 - tables 120
 - views 124
 - volumes from a storage group 88
- DSN command of TSO
 - command processor
 - connecting from TSO 439
 - invoked by TSO batch work 372
 - issues commands 370
 - running TSO programs 369
 - subcommands
 - END 441
- DSN message prefix 362
- DSN1CHKR utility
 - control of data set access 334
- DSN1COPY utility
 - control of data set access 334
 - resetting log RBA 658
 - restoring data 593
- DSN1LOGP output 636
- DSN1LOGP utility
 - control of data set access 334
 - example 634
 - extract log records 723
 - limitations 658
 - print log records 723
 - shows lost work 626
- DSN1PRNT utility
 - description 334
- DSN3@ATH connection exit routine. 754
- DSN3@SGN sign-on exit routine. 754
- DSN8EAE1 exit routine 781
- DSN8HUFF edit routine 781

- DSNACICS stored procedure 821
- DSNC command of CICS
 - destination 360
 - prefix 363
- DSNC DISCONNECT command of CICS
 - description 445
 - terminate DB2 threads 442
- DSNC DISPLAY command of CICS
 - description 442
 - DSNC DISPLAY PLAN 444
 - DSNC DISPLAY TRANSACTION 444
- DSNC STOP command of CICS
 - stop DB2 connection to CICS 442
- DSNC STRT command of CICS
 - example 443
 - processing 443
 - start DB2 connection to CICS 442
- DSNC transaction code
 - entering DB2 commands 359
- DSNCLI (CICS language interface module)
 - running CICS applications 372
- DSNDAIDL mapping macro 758
- DSNDB01 database
 - authority for access 179
- DSNDB04 default database 38
- DSNDB06 database
 - authority for access 179
 - changing high-level qualifier 135
- DSNDB07
 - extending data set for 672
- DSNDB07 database. 552
- DSNDDTXP mapping macro 790
- DSNDEDIT mapping macro 782
- DSNDEXPL mapping macro 812
- DSNDFPPB mapping macro 798
- DSNDROW mapping macro 817
- DSNDRVAL mapping macro 786
- DSNDSLRLB mapping macro 739
- DSNDSLRF mapping macro 745
- DSNDXAPL parameter list 770
- DSNELI (TSO language interface module) 369
- DSNJSRLR macro
 - capturing log records 723
 - stand-alone CLOSE 747
 - stand-alone sample program 748
- DSNLEUSR
 - Encrypting inbound IDs 283
 - Encrypting outbound IDs 298
 - Encrypting passwords 298
- DSNMxxx messages 362
- DSNTEJIS job 65, 66
- DSNTIJEX job
 - exit routines 754
- DSNTIJIC job
 - improving recovery of inconsistent data 547
- DSNX@XAC access control authorization exit routine 764
- DSNZPxxx
 - subsystem parameters module
 - specifying an alternate 366
- DSSIZE clause, CREATE TABLESPACE statement 38
- dual logging
 - active log 481
 - archive logs 485
 - restoring 498
 - synchronization 482
- dump
 - caution about using disk dump and restore 550

- dynamic plan selection in CICS
 - exit routine. 810
- dynamic SQL
 - authorization 216
 - example 219
 - privileges required 182, 184
- DYNAMICRULES
 - description 216
 - example 219

E

- EA-enabled page sets 36
- EA-enabled partitioned table spaces
 - example 45
- edit procedure, changing 119
- edit routine
 - description 346, 781
 - ensuring data accuracy 346
 - row formats 813
 - specified by EDITPROC option 781
 - writing 781
- EDITPROC clause
 - exit points 782
 - specifies edit exit routine 782
- ENABLE
 - option of BIND PLAN subcommand 210
- ENCRYPT 326, 328
- encrypting
 - data 781
 - passwords from workstation 299
 - passwords on attachment requests 281, 298
- encryption 325
 - built-in functions for 325
 - column level 326
 - data 325
 - defining columns for 325
 - non-character values 329
 - password hints 327, 328
 - performance recommendations 329
 - predicate evaluation 329
 - value level 328
 - with viewsl 327
- END
 - subcommand of DSN
 - disconnecting from TSO 441
- enlarging data set for work file database 672
- Enterprise Storage Server
 - backup 554
- entity attribute names 7
- entity attributes
 - data type 7
 - values 8
 - default 9
 - domain 8
 - null 9
- entity normalization 9
 - first normal form 10
 - fourth normal form 13
 - second normal form 10
 - third normal form 11
- entity relationships
 - business rules 7
 - many-to-many relationships 7
 - many-to-one relationships 6
 - on-to-one relationships 6
 - one-to-many relationships 6

- entity relationships (*continued*)
 - one-to-one relationships 6
- environment, operating
 - CICS 372
 - IMS 371
 - TSO 369
- ERRDEST option
 - DSNC MODIFY 442
- error
 - application program 600
 - physical RW 419
 - SQL query 349
- escape character
 - example 317
 - in DDL registration tables 314
- EXECUTE privilege
 - after BIND REPLACE 209
 - description 167, 171
 - effect 204
- exit parameter list (EXPL) 812
- exit point
 - authorization routines 755
 - connection routine 755
 - conversion procedure 793
 - date and time routines 789
 - edit routine 782
 - field procedure 797
 - sign-on routine 755
 - validation routine 786
- exit routine
 - authorization control 764
 - determining if active 781
 - general considerations 810
 - writing 753
- exit routine. 818
- EXPL (exit parameter list) 812
- EXPORT command of access method services 138, 565
- Extended Remote Copy (XRC) 708
- EXTENDED SECURITY field of panel DSNTIPR 277
- extending a data set, procedure 668
- external security profile
 - trusted context 307
- external storage 22
- EXTERNAL_SECURITY column of SYSIBM.SYSROUTINES
 - catalog table, RACF access to non-DB2 resources 261

F

- failure symptoms
 - abend shows
 - log problem during restart 645
 - restart failed 628, 639
 - BSDS 626
 - CICS
 - attachment abends 608
 - task abends 611
 - waits 607
 - log 626
 - lost log information 651
 - message
 - DFH2206 606
 - DFS555 605
 - DSNB207I 660
 - DSNJ 648
 - DSNJ001I 622
 - DSNJ004I 615
 - DSNJ100 648
 - failure symptoms (*continued*)
 - message (*continued*)
 - DSNJ103I 618
 - DSNJ105I 614
 - DSNJ106I 616
 - DSNJ107 648
 - DSNJ110E 613
 - DSNJ111E 613
 - DSNJ114I 618
 - DSNM002I 602
 - DSNM004I 602
 - DSNM005I 604
 - DSNM3201I 607
 - DSNP007I 668
 - DSNP012I 667
 - DSNU086I 665, 666
 - no processing is occurring 597
 - subsystem termination 611
- fast copy function
 - Enterprise Storage Server FlashCopy 554
 - RVA SnapShot 554
- fast log apply
 - use during RECOVER processing 549
- field decoding operation
 - definition 795
 - input 805
 - output 805
- field definition operation
 - definition 795
 - input 801
 - output 801
- field description of a value 796
- field encoding operation
 - definition 795
 - input 804
 - output 804
- field procedure
 - changing 119
 - description 346, 795
 - ensuring data accuracy 346
 - specified by the FIELDPROC clause 796
 - writing 795
- field procedure information block (FPIB) 799
- field procedure parameter list (FPPL) 798
- field procedure parameter value list (FPPVL) 798
- field value descriptor (FVD) 798
- field-level access control 187
- FIELDPROC clause
 - ALTER TABLE statement 796
 - CREATE TABLE statement 796
- FlashCopy backup
 - incremental 31
- FlashCopy backups 591
- FORCE option
 - START DATABASE command 414
 - STOP DB2 command 456, 509
- foreign keys
 - adding
 - implications 104
- format
 - column 817
 - data passed to FPPVL 799
 - data set names 32
 - message 362
 - row 817
 - value descriptors 793, 801

- forward log recovery
 - phase of restart 505
 - scenario 639
- FPIB (field procedure information block) 799
- FPPL (field procedure parameter list) 798
- FPPLV (field procedure parameter value list) 798, 799
- FREE PACKAGE subcommand of DSN
 - privileges needed 182
- FREE PLAN subcommand of DSN
 - privileges needed 184
- FREEPAGE clause
 - described 38
- function
 - table
 - ADMIN_TASK_LIST function 387
 - ADMIN_TASK_STATUS function 392
- FUNCTION privilege, description 169
- function, user-defined 211
- FVD (field value descriptor) 798, 800

G

- general-use programming information, described 965
- GET_CONFIG stored procedure 912
 - filtering output 948
- GET_MESSAGE stored procedure 922
 - filtering output 948
- GET_SYSTEM_INFO stored procedure 932
 - filtering output 948
- GETHINT 327
- global transaction
 - definition of 525
- GRANT statement
 - examples 188, 193
 - format 188
 - privileges required 184
- granting privileges and authorities 188
- GROUP DD statement for stand-alone log services OPEN
 - request 740
- GUPI symbols 965

H

- heuristic damage 522
- heuristic decision 522
- Hierarchical Storage Manager (DFSMSHsm) 138
- HMIGRATE command of DFSMSHsm (Hierarchical Storage Manager) 138
- HRECALL command of DFSMSHsm (Hierarchical Storage Manager) 138
- Huffman compression.
 - exit routine 781

I

- I/O error
 - catalog 666
 - directory 666
 - occurrence 498
 - table spaces 665
- identity column
 - altering attributes 120
 - loading data 67
- identity columns
 - conditional restart 509

- IEFSSNxx member of SYS1.PARMLIB
 - IRLM 428
- IFCID (instrumentation facility component identifier)
 - 0330 483, 613
 - identifiers by number
 - 0306 737
 - 0314 781
- IFI (instrumentation facility interface)
 - decompressing log data 737
- IMAGCOPY privilege
 - description 169
- image copy
 - catalog 536, 537
 - directory 536, 537
 - frequency vs. recovery speed 536
 - incremental
 - frequency 536
 - making after loading a table 69
 - recovery speed 536
- implementor, description 212
- IMPORT command of access method services 138, 649
- IMS 515
 - commands
 - CHANGE SUBSYS 447, 452
 - DISPLAY OASN 452
 - DISPLAY SUBSYS 447, 454
 - response destination 360
 - START REGION 454
 - START SUBSYS 447
 - STOP REGION 454
 - STOP SUBSYS 447, 456
 - TRACE SUBSYS 447
 - used in DB2 environment 355
 - connecting to DB2
 - attachment facility 453
 - authorization IDs 371
 - connection ID 371
 - connection processing 268
 - controlling 447
 - dependent region connections 453
 - disconnecting applications 456
 - security 151, 249
 - sign-on processing 271
 - supplying secondary IDs 269
 - facilities
 - message format 362
 - indoubt units of recovery 523
 - language interface module (DFSLI000)
 - link-editing 371
 - loops 602
 - LTERM authorization ID
 - for message-driven regions 371
 - shown by /DISPLAY SUBSYS 454
 - used with GRANT 361
 - operating
 - batch work 371
 - entering DB2 commands 358
 - running programs 371
 - tracing 474
 - planning
 - environment 371
 - programming
 - error checking 371
 - recovery
 - resolution of indoubt units of recovery 523
 - recovery procedures 601, 602, 604
 - thread 448, 450

- IMS (*continued*)
 - waits 602
- IMS.PROCLIB library
 - connecting from dependent regions 453
- IN clause 45
- inconsistent data
 - identifying 634
- index
 - altering 125
 - add column 126
 - clustering 127
 - dropping 128
 - redefining 128
 - varying-length columns 127
 - copying 552
 - creating 60
 - implicitly 62
 - estimating storage 78
 - implementing 60
 - ownership 203
 - privileges of ownership 172
 - rebalancing partitioned table spaces 93
 - reorganizing 128
 - space
 - recovery procedure 665
 - storage allocated 35
 - storage space
 - estimating size 79
 - structure
 - index tree 78
 - leaf pages 78
 - overall 78
 - root page 78
 - subpages 78
 - version numbers
 - recycling 128
 - versions 63
- index page size
 - choosing 62
- INDEX privilege
 - description 167
- index space data sets, deferred allocation of 61
- index-controlled partitioning 53
 - creating tables 56
- indexes
 - coding index definitions 61
 - compression 64
 - index space data sets, deferred allocation of 61
 - large objects 61
 - naming 61
 - reasons to use 18
 - sequence of entries 61
- indoubt thread
 - displaying information about 432
 - recovering 528
 - resetting status 529
 - resolving 708
- indoubt units of recover
 - recovering 604
- information center consultant 180
- INSERT privilege
 - description 167
- INSERT statement 71
 - example 70
 - load data 66
- installation
 - macros
 - automatic IRLM start 430
- installation SYSADM authority
 - privileges 175
 - use of RACF profiles 336
- installation SYSOPR authority
 - privilege 177
 - use of RACF profiles 336
- integrated catalog facility
 - changing alias name for DB2 data sets 131
 - controlling storage 22
- integrity
 - reports 351
- Interactive System Productivity Facility (ISPF) 369
- internal resource lock manager (IRLM) 428
- invalid LOB, recovering 664
- invoker, description 212
- IRLM (internal resource lock manager)
 - controlling 428
 - diagnostic trace 476
 - element name
 - global mode 430
 - local mode 430
 - failure 597
 - monitoring connection 429
 - recovery procedure 597
 - starting
 - automatically 366, 430
 - manually 430
 - stopping 430

J

- JAR 202, 203
 - privileges of ownership 172
- Java class for a routine 202, 203
 - privileges of ownership 172
- Java class privilege
 - description 171
- JCL jobs
 - scheduled execution 411

K

- key
 - adding 103, 104
 - dropping 104
 - foreign 103, 104
 - parent 103, 104
 - unique 103, 104

L

- language interface modules
 - DSNCLI
 - usage 372
- large objects (LOBs)
 - indexes 61
- LCID (log control interval definition) 731
- leaf page
 - description 78
 - index 78
- LEVEL UPDATE FREQ field of panel DSNTIPN 662
- library 955
- LIMIT BACKOUT field of panel DSNTIPN 510

materialized query table (*continued*)
 changing definition of 117
 changing to a base table 116
 registering 115
 registering an existing table as 115

materialized query tables
 creating 52

media failures
 recovering from 700

message
 DSNJ106I 641
 format
 DB2 362
 IMS 362
 prefix for DB2 362
 receiving subsystem 362
 z/OS abend
 IEC030I 620
 IEC031I 620
 IEC032I 620

message by identifier
 \$HASP373 365
 DFS058 448
 DFS058I 456
 DFS3602I 604
 DFS3613I 448
 DFS554I 605
 DFS555A 605
 DFS555I 605
 DSN1150I 646
 DSN1151I 601
 DSN1157I 634, 646
 DSN1160I 634, 646
 DSN1162I 601, 634, 646
 DSN1213I 654
 DSN2001I 608
 DSN2025I 611
 DSN2034I 608
 DSN2035I 608
 DSN2036I 608
 DSN3100I 365, 368, 611
 DSN3104I 368, 611
 DSN3201I 607
 DSN9032I 461
 DSNB204I 660
 DSNB207I 660
 DSNB232I 662
 DSNC012I 446
 DSNC016I 524
 DSNC025I 446
 DSN1006I 420
 DSN1021I 420
 DSNJ001I 365, 484, 503, 626, 628
 DSNJ002I 484
 DSNJ003I 484, 623
 DSNJ004I 484, 615
 DSNJ005I 484
 DSNJ007I 632
 DSNJ008E 483
 DSNJ012I 632, 641
 DSNJ072E 486
 DSNJ099I 365
 DSNJ100I 621, 622, 626, 648
 DSNJ103I 618, 632, 641
 DSNJ104I 618, 632
 DSNJ105I 614
 DSNJ106I 616, 632, 641

message by identifier (*continued*)
 DSNJ107I 621, 626, 648
 DSNJ108I 621
 DSNJ110E 483, 613
 DSNJ111E 483, 613
 DSNJ113E 632, 641, 647
 DSNJ114I 618
 DSNJ115I 618
 DSNJ119I 626
 DSNJ119I 648
 DSNJ120I 503, 621, 622
 DSNJ124I 616
 DSNJ125I 499
 DSNJ126I 621
 DSNJ127I 365
 DSNJ128I 620
 DSNJ130I 503
 DSNJ139I 484
 DSNJ311E 488
 DSNJ312I 488
 DSNJ317I 488
 DSNJ318I 488
 DSNJ319I 488
 DSNL001I 461
 DSNL002I 473
 DSNL003I 461
 DSNL004I 461
 DSNL005I 473
 DSNL006I 473
 DSNL009I 466
 DSNL010I 466
 DSNL030I 678
 DSNL080I 462
 DSNL200I 464
 DSNL432I 473
 DSNL433I 473
 DSNL500I 677
 DSNL501I 673, 677
 DSNL502I 673, 677
 DSNL700I 674
 DSNL701I 675
 DSNL702I 675
 DSNL703I 675
 DSNL704I 675
 DSNL705I 675
 DSNM001I 448, 454
 DSNM002I 454, 456, 602, 611
 DSNM003I 448, 454
 DSNM004I 523, 602
 DSNM005I 452, 523, 604
 DSNP001I 668
 DSNP007I 668
 DSNP012I 667
 DSNR001I 365
 DSNR002I 365, 626
 DSNR003I 365, 495, 601, 644, 646
 DSNR004I 365, 504, 505, 626, 628, 639
 DSNR005I 365, 505, 626, 628, 645
 DSNR006I 365, 506, 626
 DSNR007I 365, 504, 505
 DSNR031I 505
 DSNT360I 415, 417, 418, 421
 DSNT361I 415, 417, 418, 421
 DSNT362I 415, 417, 418, 421
 DSNT392I 421, 724
 DSNT397I 417, 418, 421
 DSNU086I 665, 666

- message by identifier (*continued*)
 - DSNU561I 672
 - DSNU563I 672
 - DSNV086E 611
 - DSNV400I 488
 - DSNV401I 444, 451, 452, 488, 608
 - DSNV402I 359, 433, 454, 488
 - DSNV404I 440, 454
 - DSNV406I 444, 451, 452, 608
 - DSNV408I 444, 445, 451, 458, 514, 608
 - DSNV414I 445, 451, 458, 608
 - DSNV415I 445, 451, 458, 608
 - DSNV431I 445
 - DSNV435I 513
 - DSNX940I 468
 - DSNY001I 365
 - DSNY002I 368
 - DSNZ002I 365
 - DXR105E 430
 - DXR117I 430
 - DXR121I 430
 - DXR122E 597
 - DXR165I 430
 - EDC3009I 667
 - IEC161I 660
- message processing program (MPP) 453
- MIGRATE command of DFSMSHsm (Hierarchical Storage Manager) 138
- mixed data
 - altering subtype 119
- MODIFY irlmproc,ABEND command of z/OS
 - stopping IRLM 430
- MODIFY utility
 - retaining image copies 547
- MONITOR1 privilege
 - description 169
- MONITOR2 privilege
 - description 169
- monitoring
 - CAF connections 440
 - connections activity 454
 - databases 415
 - DSNC commands for 444
 - IRLM 429
 - threads 444
 - TSO connections 440
 - user-defined functions 425
- moving
 - table
 - different page size 123
- moving data set 141
 - with utilities 142
 - without utilities 141
- moving DB2 data 140
 - tools 138
- MPP (message processing program), connection control 453
- MSGQUEUE attribute
 - unsolicited CICS messages 363
- multi-character command prefix 358
- multi-site update
 - illustration 518
 - process 517
- multilevel security
 - advantages 230
 - constraints 232
 - DB2 resource classes 235
 - definition 230

- multilevel security (*continued*)
 - edit procedures 232
 - field procedures 232
 - global temporary tables 232
 - hierarchies for objects 235
 - implementing 235
 - in a distributed environment 247
 - mandatory access checking 233
 - row-level granularity 237
 - security category 231
 - security label 230
 - security level 231
 - SNA support 249
 - SQL statements 237
 - TCP/IP support 247
 - triggers 233
 - using utilities with 246
 - validation procedures 232
 - views 239
- multivolume archive log data sets 486
- MxxACT DD statement for stand-alone log services OPEN
 - request 740
- MxxARCHV DD statement for stand-alone log services OPEN
 - request 740
- MxxBSDS DD statement for stand-alone log services OPEN
 - request 740

N

- naming convention
 - VSAM data sets 32
- NetView
 - monitoring errors in the network 471
- network ID (NID) 608
- NID (network ID)
 - indoubt threads 602
 - thread identification 448
 - unique value assigned by IMS 448
 - use with CICS 608
- NOT LOGGED 90
- NOT NULL clause
 - CREATE TABLE statement
 - requires presence of data 346
- null value
 - effect on storage space 814
- numeric
 - data
 - format in storage 818

O

- OASN (originating sequence number)
 - indoubt threads 602
 - part of the NID 448
- object
 - controlling access to 229
 - ownership 172, 203
- object ownership
 - trusted context 203
- object registration table (ORT) 313
- objects
 - recovering dropped objects 575
- offloading
 - active log 482
 - description 481
 - messages 484

- offloading (*continued*)
 - trigger events 483
- online 955
- online books 955
- operation
 - description 413
 - log 350
- operator
 - commands 355
- originating sequence number (OASN) 448
- ORT (object registration table) 313
- output, unsolicited
 - CICS 363
 - operational control 363
 - subsystem messages 363
- overflow 726
- OWNER
 - qualifies names in plan or package 202
- ownership
 - changing 203
- ownership of objects
 - establishing 172, 203
 - privileges 172

P

- PACKADM authority
 - description 177
- package
 - administrator 180, 189
 - authorization to execute SQL in 206
 - controlling use of DDL 311, 322
 - inoperative, when privilege is revoked 197
 - invalidated
 - dropping a view 124
 - dropping an index 128
 - when privilege is revoked 197
 - when table is dropped 120
 - list
 - privilege needed to include package 182
 - privileges needed to bind 211
 - privileges
 - description 151, 202
 - explicit 171
 - for copying 211
 - of ownership 172, 205
 - remote bind 211
 - retrieving catalog information 228
 - routine 211
- page
 - 16-KB 75
 - 32-KB 75
 - 8-KB 75
 - number of records
 - description 75
 - root 78
 - size of index 78
- page set
 - altering 94
 - control records 728
 - copying 552
- page size
 - choosing 40, 41
 - choosing for LOBs 42
 - LOBs 43

- parent keys
 - adding
 - implications 104
- PARM option of START DB2 command 366
- partial recovery. 582
- participant
 - in multi-site update 517
 - in two-phase commit 515
- partition
 - adding 106, 113
 - altering 108
 - boundary
 - changing 109
 - extending 111
 - index-controlled, redefining, procedure 668
 - rotating 110
 - table-controlled, redefining, procedure 668
- partition boundary
 - reverting to previous 112
- partition-by-growth universal table spaces
 - example 45
- partitioning methods
 - differences 53
- partner LU
 - trusting 281
 - verifying by VTAM 280
- PassTicket
 - configuring to send 298
- password
 - changing expired ones when using DRDA 277
 - encrypting, for inbound IDs 281
 - encrypting, from workstation 299
 - RACF, encrypted 298
 - requiring, for inbound IDs 281
 - sending, with attachment request 298
- pattern character
 - examples 317
 - in DDL registration tables 314
- PCTFREE clause 38
- performance
 - affected by
 - cache for authorization IDs 208
 - secondary authorization IDs 180
 - storage group 22
- phases of execution
 - restart 502
- physical database design 15
- plan
 - privileges
 - of ownership 172, 205
- PLAN
 - option of DSNC DISPLAY command 444
- plan selection exit routine
 - description 810
 - writing 810
- plan, application 171
- POE 286
- point of consistency
 - CICS 515
 - description 479
 - IMS 515
 - recovering data 556
 - single system 515
- point-in-time recovery 581
 - catalog and directory 568
 - description 582
 - RECOVER utility 558

- point-in-time recovery (*continued*)
 - subsystem 661
- pointer, overflow 726
- populating
 - tables 66
- port of entry 279, 286
 - RACF APPCPORT class 257
 - RACF SERVAUTH class 258
- postponed-abort unit of recovery 520
- power failure recovery procedure, z/OS 597
- primary authorization ID 166
- primary space allocation
 - example 27
- PRINT
 - command of access method services 575
- print log map utility
 - before fall back 649
 - control of data set access 334
 - prints contents of BSDS 427, 495
- PRIQTY clause, CREATE TABLESPACE statement 45
- privilege
 - description 150, 167
 - executing an application plan 151, 202
 - exercised by type of ID 180
 - exercised through a plan or package 204, 210
 - explicitly granted 150, 167
 - granting 149, 185, 186, 194, 198
 - implicitly held 172, 203, 308
 - needed for various roles 180
 - ownership 172
 - remote bind 211
 - remote users 187
 - retrieving catalog information 224, 229
 - revoking 195
 - routine plans, packages 211
 - trusted context 308
 - types 167
 - used in different jobs 180
- privilege selection, sample security plan 154
- PROCEDURE privilege 169
- process
 - description 149
- processing
 - attachment requests 277, 293
 - connection requests 268, 273
 - sign-on requests 271, 272
- product-sensitive programming information, described 965
- production binder
 - description 180
 - privileges 191
- programming interface information, described 965
- protocols
 - SNA 277
 - TCP/IP 284
- PSB name, IMS 371
- PSPI symbols 965
- PSRCP (page set recovery pending) status
 - description 69
- PSTOP transaction type 453
- PUBLIC AT ALL LOCATIONS clause
 - GRANT statement 187
- PUBLIC clause
 - GRANT statement 185, 186
- PUBLIC identifier 186
- PUBLIC* identifier 187

Q

- QSAM (queued sequential access method) 485
- qualification area used in IFI
 - description 738
- qualified objects
 - ownership 203
- QUALIFIER
 - qualifies names in plan or package 202
- queued sequential access method (QSAM) 485
- QUIESCE option
 - STOP DB2 command 456, 509

R

- RACF (Resource Access Control Facility)
 - authorizing
 - access to data sets 152, 334, 336
 - access to protected resources 252
 - access to SERVER resource class 260
 - group access 256
 - IMS access profile 256
 - SYSADM and SYSOPR authorities 256
 - checking
 - connection processing 268, 273
 - inbound remote IDs 280
 - sign-on processing 271, 272
 - defining
 - access profiles 250
 - DB2 resources 250, 261
 - protection for DB2 261
 - remote user IDs 255
 - started procedure table 255
 - user ID for DB2 started tasks 252
 - description 151
 - PassTickets 298
 - passwords, encrypted 298
 - typical external security system 249
 - when supplying secondary authorization ID 270, 272
- RACF access control module 820
- range-partitioned universal table spaces
 - example 45
- RBA (relative byte address)
 - description 723
 - range shown in messages 484
- RDO (resource definition online)
 - MSGQUEUE attribute 363
 - STATSQUEUE attribute 363
 - STATUSQUEUE attribute 442
- re-creating
 - tables 123
- REBALANCE 93
- REBIND PACKAGE subcommand of DSN
 - options
 - OWNER 204
- REBIND PLAN subcommand of DSN
 - options
 - OWNER 204
- rebinding
 - after creating an index 128
 - after dropping a view 124
- REBUILD INDEX utility 61
- REBUILD-pending status
 - description for indexes 533
- record
 - performance considerations 75
 - size 75

- RECORDING MAX field of panel DSNTIPA
 - preventing frequent BSDS wrapping 647
- RECOVER BSDS command
 - copying good BSDS 498
- RECOVER INDOUBT command
 - free locked resources 608
 - recover indoubt thread 528
- RECOVER privilege
 - description 169
- RECOVER TABLESPACE utility
 - DFSMSdss concurrent copy 554
 - recovers data modified after shutdown 649
- RECOVER utility 558
 - cannot use with work file table space 552
 - catalog and directory tables 568
 - data inconsistency problems 547
 - deferred objects during restart 508
 - DFSMSdss RESTORE command 29
 - functions 550
 - kinds of objects 550
 - messages issued 550
 - options
 - TOCOPY 582
 - TOLOGPOINT 582
 - TOLOGPOINT in application program error 600
 - TORBA 582
 - problem on DSNDB07 567
 - recovers pages in error 421
 - recovery cycle, establishing 697
 - running in parallel 549
 - use of fast log apply during processing 549
- RECOVERDB privilege
 - description 168
- recovering
 - DB2 system 591
- recovery
 - BSDS 623
 - catalog and directory 568
 - data set
 - using DFSMS 554
 - using DFSMSHsm 538
 - using non-DB2 dump and restore 575
 - database
 - active log 723
 - using a backup copy 533
 - using RECOVER TOCOPY 582
 - using RECOVER TOLOGPOINT 582
 - using RECOVER TORBA 582
 - down-level page sets 662
 - dropped objects 575
 - dropped table 576
 - dropped table space 578
 - FlashCopy backups 591
 - identifying objects 563
 - indexes 533
 - indoubt threads 708
 - indoubt units of recovery
 - CICS 445, 608
 - IMS 451
 - media 550
 - minimizing outages 539
 - moving data 559
 - multiple systems environment 521
 - operation 534
 - point in time 661
 - point-in-time 582
 - prior point of consistency 556
 - recovery (*continued*)
 - procedures 597
 - reducing time 536
 - reporting information 543
 - restart 565
 - subsystem 661, 723
 - system procedures 531
 - table space
 - COPY 575
 - dropped 578
 - DSN1COPY 575
 - point in time 564
 - procedure 665
 - QUIESCE 564
 - RECOVER TOCOPY 582
 - RECOVER TOLOGPOINT 582
 - RECOVER TORBA 582
 - work file table space 568
 - recovery cycle
 - RECOVER, establishing with 697
 - recovery log
 - record formats 731
 - RECOVERY option of REPORT utility 600
 - recovery procedures 668
 - application program error 600
 - CICS-related failures
 - application failure 606
 - attachment facility failure 611
 - manually recovering indoubt units of recovery 608
 - not operational 607
 - DB2-related failures
 - active log failure 613
 - archive log failure 618
 - BSDS 621
 - catalog or directory I/O errors 666
 - database failures 660
 - subsystem termination 611
 - table space I/O errors 665
 - disk failure 598
 - IMS-related failures 600
 - application failure 605
 - control region failure 602
 - fails during indoubt resolution 602, 604
 - integrated catalog facility catalog VVDS failure 667, 668
 - invalid LOB 664
 - IRLM failure 597
 - out of space 668
 - restart 626
 - z/OS failure 597
 - recovery scenarios
 - communications failure 710
 - correcting an incorrect heuristic decision 720
 - DB2 cold start 719
 - DB2 outage with cold start 715
 - failure during log initialization or current status
 - rebuild 628
 - heuristic decision, making 711
 - IMS outage with cold start 713
 - indoubt threads 708
 - starting 365
 - RECP (RECOVERY-pending) status
 - description 69
 - redefining an index-based partition 668
 - redefining an table-based partition 668
 - redefining partitions
 - index-based partitioning 671
 - table-based partitioning 671

- redo log records 724
- REFERENCES privilege
 - description 167
- referential constraint
 - adding to existing table 102
 - data consistency 348
 - recovering from violating 672
- referential structure, maintaining consistency for recovery 547
- registration tables for DDL
 - adding columns 312, 323
 - CREATE statements 321
 - creating 312
 - escape character 314
 - examples 315, 318
 - function 312, 322
 - indexes 312
 - managing 312
 - pattern characters 314
 - preparing for recovery 533
 - updating 323
- Relative Byte Address 492
- relative byte address (RBA) 484
- relative byte address (RBA). 723
- remote logical unit, failure 677
- remote request 280, 287
- REORG privilege
 - description 168
- REORG UNLOAD EXTERNAL 138
- REORG utility
 - examples 118
 - moving data 138
- REPAIR privilege
 - description 168
- REPAIR utility
 - resolving inconsistent data 658
- REPORT utility
 - options
 - RECOVERY 600
 - TABLESPACESET 600
 - table space recovery 543
- REPRO command of access method services 575, 623
- RESET INDOUBT command
 - reset indoubt thread 529
- residual recovery entry (RRE) 452
- Resource Access Control Facility (RACF) 268
- resource definition online (RDO) 363
- resource limit facility (governor)
 - preparing for recovery 533
- resource manager
 - resolution of indoubt units of recovery 525
- Resource Recovery Services (RRS), controlling
 - connections 457
- Resource Recovery Services attachment facility (RRSAF)
 - RACF profile 259
 - stored procedures and RACF authorization 259
- resource translation table (RTT) 453
- resources
 - defining to RACF 250
- restart 508
 - automatic 507
 - backward log recovery
 - failure during 645
 - phase 506
 - cold start situations 651
 - conditional
 - control record governs 508
 - excessive loss of active log data 652
 - restart (*continued*)
 - conditional (*continued*)
 - total loss of log 651
 - current status rebuild
 - failure during 628
 - phase 504
 - data object availability 511
 - DB2 501
 - deferring processing
 - objects 511
 - effect of lost connections 522
 - forward log recovery
 - failure during 639
 - phase 505
 - log initialization
 - failure during 628
 - phase 503
 - multiple systems environment 521
 - normal 502
 - overriding automatic 510
 - preparing for recovery 565
 - recovery operations for 512
 - resolving inconsistencies after 655
 - unresolvable
 - BSDS problems during 648
 - log data set problems during 648
- RESTART ALL field of panel DSNTIPS 510
- RESTORE phase of RECOVER utility 550
- RESTORE SYSTEM
 - recovery cycle, establishing 695
- RESTORE SYSTEM utility 30, 581
- restoring
 - databases 581
 - DB2 system 581
- restoring data to a prior level 556
- REVOKE statement
 - cascading effect 193
 - delete a view 196, 197
 - examples 193, 198
 - format 193
 - invalidates a plan or package 197
 - privileges required 184
 - revoking SYSADM authority 198
- RFMTTYPE
 - BRF 815
 - RRF 815
- roles 166
- rollback
 - maintaining consistency 520
 - unit of recovery 480
- root page
 - description 78
 - index 78
- route codes for messages 360
- routine
 - example, authorization 212
 - plans, packages 211
 - retrieving information about authorization IDs 228
- routine privileges 169
- row
 - formats for exit routines 813
 - validating 785
- row-level security
 - security label column 237
 - using SQL statements 237
- ROWID column
 - inserting 71

- ROWID column (*continued*)
 - loading data 67
- RRDF (Remote Recovery Data Facility)
 - altering a table for 118
- RRE (residual recovery entry)
 - detect 452
 - logged at IMS checkpoint 523
 - not resolved 523
 - purge 452
- RRSAF (Recoverable Resource Manager Services attachment facility)
 - application program
 - authorization 207
- RRSAF (Resource Recovery Services attachment facility)
 - application program
 - running 373
- RTT (resource translation table)
 - transaction type 453
- RUN
 - subcommand of DSN
 - example 369
- RVA (RAMAC Virtual Array)
 - backup 554

S

- sample exit routine
 - connection
 - location 754
 - processing 760
 - supplies secondary IDs 269
 - edit 781
 - sign-on
 - location 754
 - processing 760
 - uses secondary IDs 272
- sample library 66
- sample security plan
 - new application 188, 193
- SBCS data
 - altering subtype 119
- scenario
 - index-controlled partitioning 142
 - table-controlled partitioning 142
- scheduled tasks
 - checking status 391
 - defining 377
 - listing 387
 - removing 375, 395
- schema
 - authorization to process 65
 - creating 65
 - privileges 170
 - processing 65
 - processor 65
- schema definition
 - authorization to process 66
 - processing 66
- schemas
 - implementing 65
- SDSNLOAD library
 - loading 453
- SDSNSAMP library
 - processing schema definitions 66
- SECACPT option of APPL statement 280
- secondary authorization ID 166
- secondary space allocation
 - example 27
- SECQTY clause, CREATE TABLESPACE statement 45
- Secure Socket Layer
 - configure 331
 - requester 333
 - server 332
 - secure port
 - define 332
- security
 - access to
 - DB2 data sets 334
 - administrator privileges 180
 - CICS 151, 249
 - closed application 322
 - DDL control registration tables 311
 - description 147
 - IMS 151, 249
 - measures in application program 209
 - measures in force 345
 - mechanisms 273
 - system, external 249
- security administrator 180
- security category
 - definition 231
- security label
 - definition 230
- security label column 237
- security level
 - definition 231
- segment of log record 729
- segmented table spaces
 - example 45
- SELECT privilege
 - description 167
- SELECT statement
 - example
 - SYSIBM.SYSPLANDEP 122
 - SYSIBM.SYSTABLEPART 89
 - SYSIBM.SYSVIEWDEP 122
- sequence
 - privileges of ownership 172
- SET ARCHIVE command
 - description 355
- SET CURRENT SQLID statement 166
- SET ENCRYPTION PASSWORD 326
- shortcut keys
 - keyboard xviii
- sign-on
 - exit point 755
 - exit routine. 753
 - processing 271
 - requests 755
- sign-on exit routine
 - debugging 762
 - default 271, 272
 - description 753
 - performance considerations 762
 - sample 272
 - location 754
 - provides secondary IDs 760
 - secondary authorization ID 272
 - using 271
 - writing 753
- sign-on processing
 - choosing for remote requests 280
 - initial primary authorization ID 271

- sign-on processing (*continued*)
 - invoking RACF 271
 - requests 267
 - usage 267
 - using exit routine 271
 - using secondary IDs 272
- SIGNON-ID option of IMS 371
- SMF (System Management Facility)
 - trace record
 - auditing 338
- SMS (Storage Management Subsystem) 486
- SMS archive log data sets 486
- SNA
 - mechanisms 273
 - protocols 277
- softcopy publications 955
- software protection 341
- sorting sequence, altering by a field procedure 795
- space attributes 89
 - specifying 106
- SPACENAM option
 - DISPLAY DATABASE command 417
 - START DATABASE command 414
- SPUFI
 - disconnecting 441
- SQL (Structured Query Language)
 - transaction unit of recovery 479
- SQL authorization ID 166
- SQL CREATE AUXILIARY TABLE statement 61
- SSL
 - configure 331
 - requester 333
 - server 332
 - secure port
 - define 332
- SSM (subsystem member)
 - error options 453
 - specified on EXEC parameter 453
- SSR command of IMS
 - entering 358
 - prefix 363
- stand-alone utilities
 - recommendation 427
- START DATABASE command
 - example 414
 - problem on DSNDB07 567
 - SPACENAM option 414
- START DB2 command
 - description 366
 - entered from z/OS console 365
 - mode identified by reason code 456
 - PARM option 366
 - restart 508
- START FUNCTION SPECIFIC command
 - starting user-defined functions 424
- START REGION command of IMS 454
- START SUBSYS command of IMS 447
- START TRACE command
 - AUDIT option 342
 - controlling data 475
- STARTDBD privilege
 - description 168
- started procedures table in RACF 255
- started-task address space 252
- starting
 - audit trace 342
 - databases 413
- starting (*continued*)
 - DB2
 - after an abend 367
 - process 365
 - IRLM
 - process 430
 - table space or index space having restrictions 414
 - user-defined functions 424
- static SQL
 - privileges required 184
- STATS privilege
 - description 168
- STATSQUEUE attribute
 - unsolicited CICS messages 363
- status
 - CHECK-pending 70
 - COPY-pending, resetting 69
 - REBUILD-pending 69
- STATUS
 - column of DISPLAY DATABASE report 415
- STOGROUP privilege
 - description 172
- STOP DATABASE command 422
 - example 422
 - problem on DSNDB07 567
 - SPACENAM option 414
- STOP DDF command
 - description 472
- STOP FUNCTION SPECIFIC command
 - stopping user-defined functions 426
- STOP REGION command of IMS 454
- STOP SUBSYS command of IMS 447, 456
- STOP TRACE command
 - AUDIT option 343
 - description 475
- STOP transaction type 453
- STOPALL privilege
 - description 169
- STOPDBD privilege
 - description 168
- stopping
 - audit trace 342
 - data definition control 321
 - databases 422
 - DB2 368
 - IRLM 430
 - user-defined functions 426
- storage
 - auxiliary 22
 - external 22
 - LOBs 76
 - managing
 - using DFSMSHsm 28
 - space of dropped table, reclaiming 120
- storage group
 - control interval sizing 21
 - managing
 - using SMS 22
- storage group, DB2
 - adding volumes 88
 - altering 88
 - assigning objects to 22
 - changing to SMS-managed 88
 - changing to use a new high-level qualifier 137
 - creating 22
 - default group 22
 - description 22

- storage group, DB2 (*continued*)
 - managing
 - deferring allocation 20
 - defining data sets 20
 - deleting data sets 20
 - extending data sets 20
 - moving data sets 20
 - reusing data sets 20
 - order of use 22
 - privileges of ownership 172
 - SYSDEFLT 22
- storage groups
 - implementing 20
 - letting SMS manage 88
- stored procedure
 - address space 253
 - altering 129
 - authority to access non-DB2 resources 261
 - authorizations 211
 - commands 467
 - common SQL API 911
 - creating 72
 - dropping 72
 - DSNACICS 821
 - example, authorization 212
 - privileges
 - of ownership 172
 - RACF protection for 259
- stored procedures
 - ADMIN_COMMAND_DB2 841
 - ADMIN_INFO_SSID 881
 - ADMIN_TASK_ADD 379
 - ADMIN_TASK_REMOVE 396
 - GET_CONFIG 912
 - filtering output 948
 - GET_MESSAGE 922
 - filtering output 948
 - GET_SYSTEM_INFO 932
 - filtering output 948
 - implementing 72
 - scheduled execution 411
- STOSPACE privilege
 - description 169
- string conversion exit routine. 792
- subsystem
 - controlling access 152, 249
 - recovery 723
 - termination scenario 611
- subtypes 119
- synonym
 - privileges of ownership 172
- syntax diagram
 - how to read xix
- SYS1.LOGREC data set 611
- SYS1.PARMLIB library
 - specifying IRLM in IEFSSNxx member 428
- SYSADM authority
 - description 176
 - revoking 198
- SYSCOPY
 - catalog table, retaining records in 544
- SYSCTRL authority
 - description 177
- SYSIBM.ADMIN_TASKS 404
- SYSIBM.IPNAMES table of CDB
 - remote request processing 289
 - translating outbound IDs 289
- SYSIBM.LOCATIONS table
 - PORT option
 - specify 333
 - SECURE option
 - specify 333
- SYSIBM.LUNAMES table of CDB
 - accepting inbound remote IDs 275, 287
 - remote request processing 275, 287
 - sample entries 281
 - translating inbound IDs 281
 - translating outbound IDs 275, 287
- SYSIBM.USERNAMES table of CDB
 - managing inbound remote IDs 280
 - remote request processing 275, 287
 - sample entries for inbound translation 282
 - sample entries for outbound translation 296
 - translating inbound and outbound IDs 275, 287
- SYSLGRNX directory table
 - information from the REPORT utility 543
 - table space
 - retaining records 544
- SYSOPR authority
 - description 177
 - usage 361
- system
 - management functions, controlling 474
 - privileges 169
- system administrator
 - description 180
 - privileges 188
- System Management Facility (SMF) 338
- system operator 180
- system programmer 180
- system-level backups 556
 - moving data 559

T

- table
 - altering
 - adding a column 95
 - data type 97
 - auditing 343
 - changing page size 123
 - creating
 - clone 56
 - description 49
 - dropping
 - implications 120
 - estimating storage 75
 - loading
 - INSERT statement 70
 - ownership 203
 - populating
 - loading data into 66
 - privileges 167, 172
 - qualified name 203
 - re-creating 123
 - recovery of dropped 576
 - registration, for DDL 311, 322
 - retrieving
 - IDs allowed to access 226
 - plans and packages that can access 228
 - types 49
- table check constraints
 - adding 105
 - dropping 105

- table names
 - guidelines 48
- table space
 - altering 89
 - changing space allocation 91
 - copying 552
 - creating
 - explicitly 37
 - deferring allocation of data sets 23
 - dropping 92
 - EA-enabled 36
 - loading data into 66
 - partitioned
 - inserting rows 114
 - privileges of ownership 172
 - quiescing 564
 - recovery 665
 - recovery of dropped 578
 - recreating 92
 - reorganizing 101
 - versions 100
 - recycling version numbers 101
- table space definitions, examples of 45
- table spaces
 - coding guidelines 38
 - defining
 - explicitly 38
 - implicitly 39
 - general naming guidelines for 38
 - implementing 37
 - naming guidelines 38
 - page size recommendations 38
 - rebalancing data 93
- table-controlled partitioning
 - automatic conversion to 54
 - creating tables 53
 - description 53
 - using nullable partitioning columns 55
- tables
 - adding XML column 114
 - altering 94
 - data type 97
 - clone
 - exchanging data 57
 - implementing 47
 - loading
 - considerations for INSERT 71
 - inserting a single row 70
 - inserting multiple rows 70
 - LOAD utility 67
- TABLESPACE privilege
 - description 172
- TABLESPACESET option of REPORT utility 600
- TCP/IP
 - authorizing DDF to connect 261
 - failure, recovering from 676
 - protocols 284
- temporary tables
 - creating 48
 - types 48
- TERM UTILITY command
 - when not to use 548
- terminal monitor program (TMP) 372
- terminating 501
 - DB2
 - abend 502
 - concepts 501
- terminating (*continued*)
 - DB2 (*continued*)
 - normal 501
 - normal restart 502
 - scenario 611
- thread
 - allied 460
 - attachment in IMS 448
 - CICS
 - access to DB2 443
 - database access
 - description 460
 - displaying
 - CICS 444
 - IMS 454
 - monitoring in 444
 - termination
 - CICS 442
 - IMS 450, 456
- TIME FORMAT field of panel DSNTIPF 789
- time routine
 - description 788
 - writing 788
- TMP (terminal monitor program)
 - DSN command processor 439
 - sample job 372
 - TSO batch work 372
- TOCOPY option of RECOVER utility 582
- TOLOGPOINT option of RECOVER utility 582
- TORBA option of RECOVER utility 582
- trace
 - controlling
 - DB2 474
 - IMS 474
 - diagnostic
 - CICS 474
 - IRLM 476
- TRACE privilege
 - description 169
- TRACE SUBSYS command of IMS 447
- tracker site 694
 - characteristics 694
 - disaster recovery, in 694
 - maintaining 700
 - recovering with RECOVER 701
 - recovery cycle, establishing
 - with RESTORE SYSTEM 695
 - takeover site, converting to 700
 - takeover site, with RESTORE SYSTEM LOGONLY 700
- transaction
 - CICS
 - accessing 443
 - DSNC codes 359
 - entering 372
 - IMS
 - connecting to DB2 447
 - entering 371
 - thread attachment 448
 - thread termination 450
 - SQL unit of recovery 479
- transaction manager
 - coordinating recovery of distributed transactions 525
- TRANSACTION option
 - DSNC DISPLAY command 444
- translation types 453
- translating
 - inbound authorization IDs 281, 283

translating (*continued*)
outbound authorization IDs 296

truncation

active log 483, 634

trusted connection 301

create

local 303

requester 303

server 304

reuse 305

roles 166

trusted context 301

trusted context 301, 308

define 302

ASUSER 308

external security profile 307

object ownership 203

package ownership 205

plan ownership 205

roles 166

trusted connection 307

TSO

application programs

conditions 369

running 369

background execution 372

commands issued from DSN session 370

connections

controlling 439

DB2 439

disconnecting from DB2 441

monitoring 440

DSNELI language interface module

link editing 369

entering DB2 commands 359

environment 369

two-phase commit

illustration 515

process 515

U

undo log records 724

Unified Modeling Language 13

unit of recovery

description 479

ID 731

illustration 479

in-abort

backward log recovery 506

description 520

excluded in forward log recovery 505

in-commit

description 520

included in forward log recovery 505

indoubt

causes inconsistent state 501

definition 367

description 520

displaying 451, 608

included in forward log recovery 505

recovering CICS 445

recovering IMS 451

recovery in CICS 608

recovery procedure 602

resolving 523, 524, 525

unit of recovery (*continued*)

inflight

backward log recovery 506

description 520

excluded in forward log recovery 505

log records 724

postponed

displaying 452

postponed-abort 520

rollback 480, 520

SQL transaction 479

unit of recovery ID (URID) 731

UNLOAD utility

delimited files 67

unqualified objects, ownership 202

unsolicited output

CICS 360, 363

IMS 360

operational control 363

subsystem messages 363

UPDATE privilege

description 167

updating

registration tables for DDL 323

URID (unit of recovery ID). 731

USAGE privilege

distinct type 171

Java class 171

sequence 171

USE OF privileges 172

user analyst 180

user-defined data sets

adding volumes to 670

extending 670

user-defined function

controlling

START FUNCTION SPECIFIC command 424

dropping 73

example, authorization 212

monitoring 425

privileges of ownership 172

starting 424

stopping 426

user-defined functions

altering 130

creating 73

implementing 73

user-managed data sets

changing high-level qualifier 137

deleting 35

enlarging 670

extending 35

name format 32

requirements 32

specifying data class 36

USING STOGROUP clause 45

utilities

access status needed 426

controlling 426

executing

running on objects with pages in LPL 420

internal integrity reports 351

V

validating

connections from remote application 273

- validating (*continued*)
 - existing rows with a new VALIDPROC 117
 - rows of a table 785
- validation routine
 - altering assignment 117
 - checking existing table rows 117
 - description 346
 - ensuring data accuracy 346
 - row formats 813
 - writing 785
- validation routine.
 - description 785
- VALIDPROC clause
 - ALTER TABLE statement 117
 - exit points 786
- value
 - descriptors in field procedures 800
- values of attributes 8
- VARCHAR
 - data type
 - subtypes 119
- verifying VTAM partner LU 280
- view
 - altering 124
 - INSTEAD OF trigger 124
 - creating
 - on catalog tables 229
 - dropping 60, 124
 - deleted by REVOKE 196, 197
 - INSTEAD OF trigger 124
 - implementing 57
 - list of dependent objects 122
 - name
 - qualified name 203
 - names 59
 - privileges
 - effect of revoking table privileges 196, 197
 - ownership 203
 - table privileges for 187
 - using DELETE 59
 - using INSERT 59
 - using UPDATE 59
- views
 - creating 57
 - reasons to use 17
- virtual storage access method (VSAM) 481
- volume serial number 498
- VSAM (virtual storage access method)
 - control interval
 - block size 485
 - log records 481
 - processing 575
 - volume data set (VVDS) recovery procedure 667, 668
- VTAM
 - failure, recovering from 676
- VTAM (Virtual Telecommunications Access Method)
 - APPL statement 280
 - controlling connections 251, 279
 - conversation-level security 280
 - partner LU verification 280
 - password
 - choosing 279
- VVDS recovery procedure 667, 668

W

- wait
 - recovering from 677
 - wait state at start 367
- WebSphere Application Server
 - identify outstanding indoubt units of recovery 525
- work file database
 - changing high-level qualifier 136
 - enlarging 668
 - error range recovery 568
 - extending 672
 - problems 552
 - starting 413
- WQAxXX fields of qualification area 738
- write error page range (WEPR) 419
- WRITE TO OPER field of panel DSNTIPA 483
- write-down control 234

X

- XML column
 - loading data 67
- XML columns
 - adding 114
- XML objects
 - altering implicit 130
- XRC (Extended Remote Copy) 708
- XRF (extended recovery facility)
 - CICS toleration 532
 - IMS toleration 532

Z

- z/OS
 - command group authorization level (SYS) 358, 361
 - commands
 - MODIFY irlmproc 430
 - STOP irlmproc 430
 - entering DB2 commands 358, 361
 - IRLM commands control 355
 - power failure recovery procedure 597



Program Number: 5635-DB2

Printed in USA

SC18-9840-03



Spine information:

DB2 Version 9.1 for z/OS

Administration Guide

