

Note

Before using this information and the product it supports, read the general information under Appendix C, "Notices," on page 875.

Edition Notice

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1993, 2009.

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

Contents

Common Criteria certification of DB2 products ix

Supported interfaces for a Common Criteria evaluated configuration xi

About this book xiii

Part 1. SQL Statements 1

Chapter 1. SQL Statements for Administrators 3

ALTER AUDIT POLICY	3
ALTER DATABASE PARTITION GROUP	6
ALTER FUNCTION	9
ALTER METHOD	11
ALTER PROCEDURE (External)	12
ALTER SECURITY LABEL COMPONENT	14
ALTER SECURITY POLICY	17
ALTER TABLE	20
ALTER TABLESPACE	66
ALTER VIEW	78
AUDIT	80
COMMENT	83
CREATE AUDIT POLICY	95
CREATE DATABASE PARTITION GROUP	98
CREATE FUNCTION	100
CREATE INDEX	101
CREATE METHOD	119
CREATE PROCEDURE (SQL)	124
CREATE ROLE	134
CREATE SCHEMA	135
CREATE SECURITY LABEL	137
CREATE SECURITY LABEL COMPONENT	139
CREATE SECURITY POLICY	141
CREATE TABLE	143
CREATE TABLESPACE	212
CREATE VIEW	225
DELETE	239
DROP	245
GRANT (Database Authorities)	278
GRANT (Exemption)	283
GRANT (Index Privileges)	285
GRANT (Package Privileges)	286
GRANT (Role)	289
GRANT (routine privileges)	291
GRANT (Schema Privileges)	295
GRANT (Security Label)	297
GRANT (Sequence Privileges)	300
GRANT (Server Privileges)	302
GRANT (SETSESSIONUSER Privilege)	303
GRANT (Table Space Privileges)	305
GRANT (Table, View, or Nickname Privileges)	307
INSERT	313

RENAME	321
RENAME TABLESPACE	323
REVOKE (Database Authorities)	324
REVOKE (Exemption)	328
REVOKE (Index Privileges)	330
REVOKE (Package Privileges)	331
REVOKE (Role)	334
REVOKE (routine privileges)	336
REVOKE (Schema Privileges)	339
REVOKE (Security Label)	341
REVOKE (Sequence Privileges)	342
REVOKE (Server Privileges)	344
REVOKE (SETSESSIONUSER Privilege)	346
REVOKE (Table Space Privileges)	347
REVOKE (Table, View, or Nickname Privileges)	349
SET ROLE	353
UPDATE	354

Chapter 2. SQL Statements for Users 365

COMMIT	365
CONNECT (Type 1)	366
CONNECT (Type 2)	373
DISCONNECT	379
fullselect	382
LOCK TABLE	386
ROLLBACK	388
SELECT	390
select-statement	390
SET INTEGRITY	400
SET SCHEMA	417
SQL queries	418
subselect	419
TRANSFER OWNERSHIP	458

Part 2. Functions 473

Chapter 3. Functions 475

Functions overview	475
DBPARTITIONNUM	476
DECRYPT_BIN and DECRYPT_CHAR	477
ENCRYPT	479
GETHINT	481
HASHEDVALUE	482
SECLABEL	483
SECLABEL_BY_NAME	483
SECLABEL_TO_CHAR	484
TABLE_NAME	486
TABLE_SCHEMA	487

Part 3. Applications 489

Chapter 4. Application considerations 491

About SQL statements	491
How SQL statements are invoked	491

About SQL control statements	495
Function, method, and procedure designators	497
Database connection management via embedded SQL applications	501
Connecting to DB2 databases in embedded SQL applications	501
Disconnecting from embedded SQL applications	502
Considerations for routines	503
Security of routines	503
Securing routines	504
Guidelines for stored procedures	505
Security Considerations when Using SQL in Applications	506
Precompilation of embedded SQL applications with the PRECOMPILE command	506
Compiling and linking source files containing embedded SQL	508
Package recreation using the BIND command and an existing bind file	509
Generating sequential values	509
Managing sequence behavior	510
Sequences compared to identity columns	511
Authorization Considerations for Embedded SQL	512
Effect of DYNAMICRULES bind option on dynamic SQL	513
Units of work and transactions	515
Units of work	515
Remote unit of work	516
Concurrent transactions and multi-threaded database access in embedded SQL applications	517
Security and Java Applications	519
SQLJ SET-TRANSACTION-clause	519
Setting the isolation level for an SQLJ transaction	519
SQLJ context-clause	520
Connecting to a data source using SQLJ	520
SQLJ connection-declaration-clause	520
Closing the connection to a data source in an SQLJ application	521
JDBC Considerations	522
How JDBC applications connect to a data source	522
Connecting to a data source using the DataSource interface	523
JDBC connection objects	525
Committing or rolling back JDBC transactions	525
Disconnecting from data sources in JDBC applications	526
Type 2 JDBC Driver Considerations	526
Security under the DB2 JDBC Type 2 Driver	526
How DB2 applications connect to a data source using the DriverManager interface with the DB2 JDBC Type 2 Driver	527
Universal JDBC Driver Considerations	528
User ID and password security under the IBM Data Server Driver for JDBC and SQLJ	528
User ID-only security under the IBM Data Server Driver for JDBC and SQLJ	530
Kerberos security under the IBM Data Server Driver for JDBC and SQLJ	531

Encrypted password, user ID, or user ID and password security under the IBM Data Server Driver for JDBC and SQLJ	534
Security under the IBM Data Server Driver for JDBC and SQLJ	536
Connecting to a data source using the DriverManager interface with the IBM Data Server Driver for JDBC and SQLJ	538

Chapter 5. Security and Routines . . . 541

Benefits of using routines	541
External scalar functions	542
Methods	543
Security considerations for routines	544
Connection contexts in SQLJ routines	546
External routine library and class management	547
Rebuilding DB2 routine shared libraries	547
Updating the database manager configuration file	547

Chapter 6. SQLCA (SQL communications area) 549

Chapter 7. SQLDA (SQL descriptor area) 555

Chapter 8. Identifiers 565

Part 4. Security plug-ins 591

Chapter 9. An overview of security plug-ins 593

Security plug-ins	593
Security plug-in library locations	597
Security plug-in naming conventions	597
Security plug-in support for two-part user IDs	598
32-bit and 64-bit considerations for security plug-ins	600
Security plug-in problem determination	600
Deploying a group retrieval plug-in	602
Deploying a user ID/password plug-in	602
Deploying a GSS-API plug-in	603
Deploying a Kerberos plug-in	604
Restrictions on security plug-ins	606

Chapter 10. Developing security plug-ins 609

How DB2 loads security plug-ins	609
Calling sequences for the security plug-in APIs	610
Restrictions for developing security plug-in libraries	613
Return codes for security plug-ins	615
Error message handling for security plug-ins	618

Chapter 11. Security plug-in APIs . . . 619

Security plug-in APIs	619
Group plug-in APIs	620
APIs for group retrieval plug-ins	620

db2secGroupPluginInit API - Initialize group plug-in	621
db2secPluginTerm - Clean up group plug-in resources	622
db2secGetGroupsForUser API - Get list of groups for user.	623
db2secDoesGroupExist API - Check if group exists	626
db2secFreeGroupListMemory API - Free group list memory	627
db2secFreeErrorMsg API - Free error message memory	627
User authentication plug-in APIs	627
APIs for user ID/password authentication plug-ins	627
db2secClientAuthPluginInit API - Initialize client authentication plug-in	633
db2secClientAuthPluginTerm API - Clean up client authentication plug-in resources	634
db2secRemapUserid API - Remap user ID and password.	635
db2secGetDefaultLoginContext API - Get default login context	636
db2secGenerateInitialCred API - Generate initial credentials	638
db2secValidatePassword API - Validate password.	639
db2secProcessServerPrincipalName API - Process service principal name returned from server	642
db2secFreeToken API - Free memory held by token	643
db2secFreeInitInfo API - Clean up resources held by the db2secGenerateInitialCred	643
db2secServerAuthPluginInit - Initialize server authentication plug-in	643
db2secServerAuthPluginTerm API - Clean up server authentication plug-in resources	646
db2secGetAuthIDs API - Get authentication IDs	647
db2secDoesAuthIDExist - Check if authentication ID exists	648
GSS-API plug-in APIs	649
Required APIs and definitions for GSS-API authentication plug-ins	649
Restrictions for GSS-API authentication plug-ins	650
Security plug-in API versioning	651
Security plug-in samples	651

Chapter 12. Security Plug-In Configuration Parameters 653

clnt_krb_plugin - Client Kerberos plug-in	653
clnt_pw_plugin - Client userid-password plug-in	653
group_plugin - Group plug-in.	654
local_gssplugin - GSS API plug-in used for local instance level authorization.	654
srvcon_auth - Authentication type for incoming connections at the server	654
srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server	655
srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server	656

srv_plugin_mode - Server plug-in mode	656
---	-----

Part 5. Configuration Parameters 657

Chapter 13. Configuration Parameters 659

Configuration parameters	659
Configuring the DB2 database manager with configuration parameters	660
Configuration parameters summary	663
Changing the database configuration across multiple database partitions	677
Security-Related Configuration Parameters	677
audit_buf_sz - Audit buffer size	677
authentication - Authentication type.	678
authentication - Authentication type DAS	679
catalog_noauth - Cataloging allowed without authority	680
dasadm_group - DAS administration authority group name	680
dftdbpath - Default database path	681
svcname - TCP/IP service name.	681
sysadm_group - System administration authority group name	682
sysctrl_group - System control authority group name	683
sysmaint_group - System maintenance authority group name	683
sysmon_group - System monitor authority group name	684
trust_allclnts - Trust all clients.	685
Locking Configuration Parameters	686
dlchktime - Time interval for checking deadlock	686
locklist - Maximum storage for lock list	686
locktimeout - Lock timeout.	689
maxlocks - Maximum percent of lock list before escalation.	690
SSL Configuration Parameters	692
ssl_svr_keydb - SSL key file path for incoming SSL connections at the server configuration parameter	692
ssl_svr_stash - SSL stash file path for incoming SSL connections at the server configuration parameter	692
ssl_svr_label - Label in the key file for incoming SSL connections at the server configuration parameter	693
ssl_svcname - SSL service name configuration parameter	693
ssl_versions - Supported SSL versions at the server configuration parameter	694
ssl_cipherspecs - Supported cipher specifications at the server configuration parameter	694
ssl_clnt_keydb - SSL key file path for outbound SSL connections at the client configuration parameter	695
ssl_clnt_stash - SSL stash file path for outbound SSL connections at the client configuration parameter	695

Chapter 14. Communications in a partitioned database environment . . . 697

conn_elapse - Connection elapse time . . . 697
fcm_num_buffers - Number of FCM buffers . . . 697
fcm_num_channels - Number of FCM channels . . . 698
max_connretries - Node connection retries. . . . 699
max_time_diff - Maximum time difference among nodes 699
start_stop_time - Start and stop timeout 700

Chapter 15. autorestart - Auto restart enable 703

Chapter 16. database_consistent - Database is consistent 705

Chapter 17. nodetype - Machine node type 707

Chapter 18. restrict_access - Database has restricted access configuration parameter 709

Part 6. Recovery considerations 711

Chapter 19. Crash Recovery and Database Logs. 713

Crash recovery 713

Chapter 20. Application processes, concurrency, and recovery 715

Chapter 21. Recovering from transaction failures in a partitioned database environment 717

Part 7. Appendixes 721

Appendix A. Related topics (linked to from topics in this book) 723

SQL Reference topics 723
 Assignments and comparisons. 723
 CURRENT CLIENT_ACCTNG 739
 CURRENT DATE 739
 CURRENT DECFLOAT ROUNDING MODE 739
 CURRENT DEFAULT TRANSFORM GROUP 740
 CURRENT DEGREE 741
 CURRENT EXPLAIN MODE 741
 CURRENT EXPLAIN SNAPSHOT 742
 CURRENT FEDERATED ASYNCHRONY . . . 743
 CURRENT IMPLICIT XMLPARSE OPTION . 743
 CURRENT ISOLATION 744
 CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 745
 CURRENT MDC ROLLOUT MODE. 745
 CURRENT OPTIMIZATION PROFILE 745

CURRENT PACKAGE PATH 745
CURRENT PATH 746
CURRENT QUERY OPTIMIZATION 747
CURRENT REFRESH AGE 747
CURRENT TIME 747
CURRENT TIMESTAMP 748
CURRENT TIMEZONE 748
CURRENT USER 749
Configuration parameter topics 749
 agent_stack_sz - Agent stack size. 749
 agentpri - Priority of agents 751
 alt_collate - Alternate collating sequence . . . 752
 alternate_auth_enc - Alternate encryption algorithm for incoming connections at server configuration parameter. 752
 appl_memory - Application Memory configuration parameter. 753
 applheapsz - Application heap size 754
 archretrydelay - Archive retry delay on error 755
 aslheapsz - Application support layer heap size 755
 auto_del_rec_obj - Automated deletion of recovery objects configuration parameter . . . 757
 auto_maint - Automatic maintenance 757
 avg_appls - Average number of active applications 759
 backup_pending - Backup pending indicator 760
 blk_log_dsk_ful - Block on log disk full . . . 760
 catalogcache_sz - Catalog cache size. 761
 chnpggs_thresh - Changed pages threshold . . 762
 cluster_mgr - Cluster manager name 763
 codepage - Code page for the database. 763
 codeset - Codeset for the database 763
 collate_info - Collating information 764
 comm_bandwidth - Communications bandwidth 764
 contact_host - Location of contact list 765
 country/region - Database territory code . . . 766
 cpuspeed - CPU speed 766
 cur_commit - Currently committed configuration parameter. 766
 das_codepage - DAS code page 767
 das_territory - DAS territory 768
 database_level - Database release level 768
 database_memory - Database shared memory size. 768
 db2system - Name of the DB2 server system 770
 db_mem_thresh - Database memory threshold 771
 dbheap - Database heap 771
 decflt_rounding - Decimal floating point rounding configuration parameter 773
 dft_account_str - Default charge-back account 774
 dft_degree - Default degree. 775
 dft_extent_sz - Default extent size of table spaces. 776
 dft_loadrec_ses - Default number of load recovery sessions 776
 dft_monswitches - Default database system monitor switches 777
 dft_mttb_types - Default maintained table types for optimization 778
 dft_prefetch_sz - Default prefetch size 778
 dft_queryopt - Default query optimization class 779

dft_refresh_age - Default refresh age	780	max_connections - Maximum number of client connections	813
dft_sqlmathwarn - Continue upon arithmetic exceptions	780	max_coordagents - Maximum number of coordinating agents	814
diaglevel - Diagnostic error capture level	782	max_log - Maximum log per transaction	815
diagpath - Diagnostic data directory path	782	max_querydegree - Maximum query degree of parallelism	815
dir_cache - Directory cache support	783	maxappls - Maximum number of active applications	816
discover - DAS discovery mode	785	maxfilop - Maximum database files open per application	817
discover - Discovery mode	785	min_dec_div_3 - Decimal division scale to 3	818
discover_db - Discover database	786	mincommit - Number of commits to group	819
discover_inst - Discover server instance	786	mirrorlogpath - Mirror log path	820
dyn_query_mgmt - Dynamic SQL and XQuery query management	787	mon_heap_sz - Database system monitor heap size.	821
enable_xmlchar - Enable conversion to XML configuration parameter	787	multipage_alloc - Multipage file allocation enabled	822
exec_exp_task - Execute expired tasks	788	newlogpath - Change the database log path	822
failarchpath - Failover log archive path	788	notifylevel - Notify level.	824
fed_noauth - Bypass federated authentication	788	num_db_backups - Number of database backups	825
federated - Federated database system support	789	num_freqvalues - Number of frequent values retained	825
federated_async - Maximum asynchronous TQs per query configuration parameter	789	num_initagents - Initial number of agents in pool	826
fenced_pool - Maximum number of fenced processes	790	num_initfenced - Initial number of fenced processes	827
hadr_db_role - HADR database role	791	num_iocleaners - Number of asynchronous page cleaners	827
hadr_local_host - HADR local host name	792	num_ioservers - Number of I/O servers	829
hadr_local_svc - HADR local service name	792	num_log_span - Number log span	830
hadr_peer_window - HADR peer window configuration parameter	793	num_poolagents - Agent pool size	830
hadr_remote_host - HADR remote host name	793	num_quantiles - Number of quantiles for columns	831
hadr_remote_inst - HADR instance name of the remote server	794	numarchretry - Number of retries on error	832
hadr_remote_svc - HADR remote service name	794	numdb - Maximum number of concurrently active databases including host and System i databases.	833
hadr_syncmode - HADR synchronization mode for log write in peer state	794	numsegs - Default number of SMS containers	833
hadr_timeout - HADR timeout value	795	overflowlogpath - Overflow log path	834
health_mon - Health monitoring	795	pagesize - Database default page size	835
indexrec - Index re-creation time	796	pckcachesz - Package cache size	835
instance_memory - Instance memory	798	query_heap_sz - Query heap size.	837
intra_parallel - Enable intra-partition parallelism	800	rec_his_retentn - Recovery history retention period.	838
java_heap_sz - Maximum Java interpreter heap size.	801	release - Configuration file release level	838
jdk_64_path - 64-Bit Software Developer's Kit for Java installation path DAS.	802	restore_pending - Restore pending	839
jdk_path - Software Developer's Kit for Java installation path DAS.	802	Restrictions and behavior when configuring max_coordagents and max_connections	839
jdk_path - Software Developer's Kit for Java installation path	803	resync_interval - Transaction resync interval	841
keepfenced - Keep fenced process	803	rollfwd_pending - Roll forward pending indicator	841
log_retain_status - Log retain status indicator	804	rqrioblk - Client I/O block size	842
logarchmeth1 - Primary log archive method	804	sched_enable - Scheduler mode	843
logarchmeth2 - Secondary log archive method	806	sched_userid - Scheduler user ID.	843
logarchopt1 - Primary log archive options	807	self_tuning_mem- Self-tuning memory	843
logarchopt2 - Secondary log archive options	807	seqdetect - Sequential detection flag.	845
logbufsz - Log buffer size	808	sheapthres - Sort heap threshold	845
logfilsiz - Size of log files	808	sheapthres_shr - Sort heap threshold for shared sorts	847
loghead - First active log file	809		
logindexbuild - Log index pages created	809		
logpath - Location of log files	810		
logprimary - Number of primary log files	810		
logretain - Log retain enable	811		
logsecond - Number of secondary log files	812		

smtp_server - SMTP server	848
softmax - Recovery range and soft checkpoint interval	849
sortheap - Sort heap size	850
spm_log_file_sz - Sync point manager log file size.	851
spm_log_path - Sync point manager log file path	852
spm_max_resync - Sync point manager resync agent limit	853
spm_name - Sync point manager name.	853
stat_heap_sz - Statistics heap size.	853
stmheap - Statement heap size	854
territory - Database territory	854
tm_database - Transaction manager database name	855
toolscat_db - Tools catalog database	855
toolscat_inst - Tools catalog database instance	856
toolscat_schema - Tools catalog database schema	856
tp_mon_name - Transaction processor monitor name	856
trackmod - Track modified pages enable	858
trust_clntauth - Trusted clients authentication	858
tsm_mgmtclass - Tivoli Storage Manager management class	859
tsm_nodename - Tivoli Storage Manager node name	859
tsm_owner - Tivoli Storage Manager owner name	860
tsm_password - Tivoli Storage Manager password.	860

user_exit_status - User exit status indicator	861
userexit - User exit enable	861
util_heap_sz - Utility heap size	861
util_impact_lim - Instance impact policy	862
vendoropt - Vendor options	863
wlm_collect_int - Workload management collection interval configuration parameter	863

Appendix B. Overview of the DB2 technical information 865

DB2 technical library in hardcopy or PDF format	865
Ordering printed DB2 books	868
Displaying SQL state help from the command line processor	869
Accessing different versions of the DB2 Information Center	869
Displaying topics in your preferred language in the DB2 Information Center	869
Updating the DB2 Information Center installed on your computer or intranet server	870
Manually updating the DB2 Information Center installed on your computer or intranet server	871
DB2 tutorials	873
DB2 troubleshooting information	873
Terms and Conditions	874

Appendix C. Notices 875

Index 879

Common Criteria certification of DB2 products

For Version 9.7, IBM® DB2® products are certified according to the Common Criteria evaluation assurance level 4 (EAL4), augmented with Flaw remediation ALC_FLR.1.

The following product is certified on the following operating systems:

Table 1.

	Windows® Server 2003	Red Hat Enterprise Linux® 5	SuSE Linux Enterprise Server 10	AIX® 6	Solaris 10
IBM DB2 Version 9.7 Enterprise Server Edition for Linux, UNIX®, and Windows	Yes	Yes	Yes	Yes	Yes

Note:

1. For a Common Criteria certified DB2 environment, DB2 requires 64-bit Windows Server 2003 x64, Red Hat Enterprise Linux 5, or SuSE Linux Enterprise Server 10 operating systems for Intel® EM64T- and AMD64-based systems.
2. In a Common Criteria certified DB2 environment, DB2 clients are supported on the following operating systems:
 - Windows 2003
 - Red Hat Enterprise Linux 5
 - SuSE Linux Enterprise Server 10
 - AIX 6
 - Solaris 10

For more information about Common Criteria, see the Common Criteria web site at: <http://www.commoncriteriaportal.org>.

For information about installing and configuring a DB2 system that conforms to the Common Criteria EAL4, see the following books:

- *Installing IBM DB2 Enterprise Server Edition*
- *IBM DB2 Administration and User Documentation*

These books are available in PDF format from the *DB2 Information Management Library*.

Supported interfaces for a Common Criteria evaluated configuration

The set of DB2 interfaces that are used in the Common Criteria evaluation of the DB2 database manager are as follows:

- The DB2 install program
- The command line processor
- DB2 commands
- DB2 application programming interfaces (APIs)
- SQL statements

You can use these interfaces when installing and configuring a Common Criteria compliant DB2 system.

Other interfaces that are provided by the DB2 database manager, such as the Control Center or Command Editor were not used during the Common Criteria evaluation of DB2 products, **and must not be used in the Common Criteria evaluation configuration.**

The **Workload management** feature, introduced in Version 9.7, **must also not be used in the Common Criteria evaluated configuration.** This feature is designed for complex, multi-layered environments, and its secure operation is intricately linked to the correct functionality of components well outside the boundary of the target of evaluation. The associated statements require SYSADM or DBADM administrative privileges, and are therefore not available to regular users.

Administrative users should not use the following statements to create workload management objects:

- CREATE WORKLOAD statement
- CREATE SERVICE CLASS statement
- CREATE THRESHOLD statement
- CREATE HISTOGRAM statement
- CREATE WORK CLASS SET statement
- CREATE WORK ACTION SET statement

Note:

- NOT FENCED routines are also not supported.
- Data encryption functions ENCRYPT, DECRYPT_BIN, DECRYPT_CHAR and GETHINT must not be used.
- The user-written security plugins must not be used.
- Like table privileges (SELECT, INSERT, UPDATE, DELETE), Label-Based Access Control (LBAC) has no control over access to physical files such as database files and transaction logs. Given that these files contain database data including data protected with LBAC and given that DB2 administrators have direct access to these physical files, DB2 administrators should be treated as having the highest level of access even though this is an indirect access and is outside the scope of the LBAC model.

When using the DB2 Database Partition Feature (DPF), the external security information that is used by the DB2 database manager to perform authentication and authorization must be configured consistently on each partition. This information depends on the authentication type in use. For operating system security, this information is the username, password and group membership of each user that can connect to the database. Identical usernames, passwords and groups must be created at each partition. For LDAP authentication, this information is stored in the LDAP configuration file for the LDAP-based authentication plugin. The LDAP configuration file must have the same contents on each partition. For Kerberos, in order to ensure that the same Key Distribution Center (KDC) is used for each partition, this information is stored in the Kerberos configuration file on the server where the DB2 product is installed. Failure to provide consistent configuration information at each partition could result in users being unable to authenticate, and hence connect to the DB2 database, or users having reduced privileges, if incomplete group membership information is obtained from the local operating system, or from LDAP, or from Kerberos.

About this book

This book, consisting of volumes 1 and 2, is intended for use by assessors validating that specific DB2 database products conform to the Common Criteria EAL4 specification augmented with Flaw remediation ALC_FLR.1. It is also intended for those who want to set up a DB2 environment that conforms to the characteristics of the evaluated environment.

Volume 2 describes:

- SQL statements.
- The security-related considerations for writing applications that interact with the DB2 database manager.
- Security plug-ins. Note that only the default IBM-supplied operating-system based authentication and group plug-ins are supported in Common Criteria compliant environments.

Regarding security considerations on SQL statements and SQL routines (found in chapters 5 and 6):

- Passwords appear in SQL statements in plain text. As such, any program or script containing such statements needs appropriate protection with OS- and DBMS-provided mechanisms.
- A major database vulnerability (generic) is SQL injection. As such, use caution and validate any direct user input looking for SQL injection attacks—looking for SQL statements, special characters such as {},;, and quotes.

Note: This book does not provide information on how to install DB2 database servers. For installation information, See the *Version 9.7 Installing IBM DB2 Enterprise Server Edition*.

Some topics in book link to related topics, which are either included in Appendix A in order to resolve the links, or that are referenced outside of the Common Criteria certification documentation. These are for informational purposes only, and are not required for either installing or configuring a Common Criteria compliant environment.

Part 1. SQL Statements

Chapter 1. SQL Statements for Administrators

ALTER AUDIT POLICY

The ALTER AUDIT POLICY statement modifies the definition of an audit policy at the current server.

Invocation

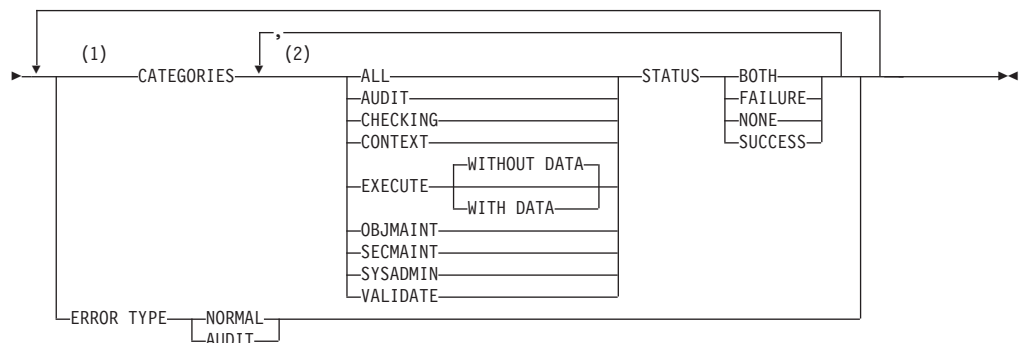
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax

►►—ALTER AUDIT POLICY—*policy-name*—►►



Notes:

- 1 Each of the CATEGORIES and ERROR TYPE clauses can be specified at most once (SQLSTATE 42614).
- 2 Each category can be specified at most once (SQLSTATE 42614), and no other category can be specified if ALL is specified (SQLSTATE 42601).

Description

policy-name

Identifies the audit policy that is to be altered. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The name must uniquely identify an existing audit policy at the current server (SQLSTATE 42704).

CATEGORIES

A list of one or more audit categories for which a new status value is specified. If ALL is not specified, the STATUS of any category that is not explicitly specified remains unchanged.

ALL

Sets all categories to the same status. The EXECUTE category is WITHOUT DATA.

AUDIT

Generates records when audit settings are changed or when the audit log is accessed.

CHECKING

Generates records during authorization checking of attempts to access or manipulate database objects or functions.

CONTEXT

Generates records to show the operation context when a database operation is performed.

EXECUTE

Generates records to show the execution of SQL statements.

WITHOUT DATA or WITH DATA

Specifies whether or not input data values provided for any host variables and parameter markers should be logged as part of the EXECUTE category.

WITHOUT DATA

Input data values provided for any host variables and parameter markers are not logged as part of the EXECUTE category.

WITH DATA

Input data values provided for any host variables and parameter markers are logged as part of the EXECUTE category. Not all input values are logged; specifically, LOB, LONG, XML, and structured type parameters appear as the null value. Date, time, and timestamp fields are logged in ISO format. The input data values are converted to the database code page before being logged. If code page conversion fails, no errors are returned and the unconverted data is logged.

OBJMAINT

Generates records when data objects are created or dropped.

SECMAINT

Generates records when object privileges, database privileges, or DBADM authority is granted or revoked. Records are also generated when the database manager security configuration parameters **sysadm_group**, **sysctrl_group**, or **sysmaint_group** are modified.

SYSADMIN

Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.

VALIDATE

Generates records when users are authenticated or when system security information related to a user is retrieved.

STATUS

Specifies a status for the specified category.

BOTH

Successful and failing events will be audited.

FAILURE

Only failing events will be audited.

SUCCESS

Only successful events will be audited.

NONE

No events in this category will be audited.

ERROR TYPE

Specifies whether audit errors are to be returned or ignored.

NORMAL

Any errors generated by the audit are ignored and only the SQLCODEs for errors associated with the operation being performed are returned to the application.

AUDIT

All errors, including errors occurring within the audit facility itself, are returned to the application.

Rules

- An AUDIT-exclusive SQL statement must be followed by a COMMIT or ROLLBACK statement (SQLSTATE 5U021). AUDIT-exclusive SQL statements are:
 - AUDIT
 - CREATE AUDIT POLICY, ALTER AUDIT POLICY, or DROP (AUDIT POLICY)
 - DROP (ROLE) or DROP (TRUSTED CONTEXT) if the role or trusted context is associated with an audit policy
- An AUDIT-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Only one uncommitted AUDIT-exclusive SQL statement is allowed at a time across all database partitions. If an uncommitted AUDIT-exclusive SQL statement is executing, subsequent AUDIT-exclusive SQL statements wait until the current AUDIT-exclusive SQL statement commits or rolls back.
- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.
- If the audit policy that is being altered is currently associated with a database object, the changes do not take effect until the next unit of work for the application that is affected by the change. For example, if the audit policy is in use for the database, no current units of work will see the change to the policy until after a COMMIT or a ROLLBACK statement for that unit of work completes.

Example

Alter the SECMAINT, CHECKING, and VALIDATE categories of an audit policy named DBAUDPRF to audit both successes and failures.

```
ALTER AUDIT POLICY DBAUDPRF
  CATEGORIES SECMAINT STATUS BOTH,
             CHECKING STATUS BOTH,
             VALIDATE STATUS BOTH
```

ALTER DATABASE PARTITION GROUP

The ALTER DATABASE PARTITION GROUP statement is used to:

- add one or more database partitions to a database partition group
- drop one or more database partitions from a database partition group.

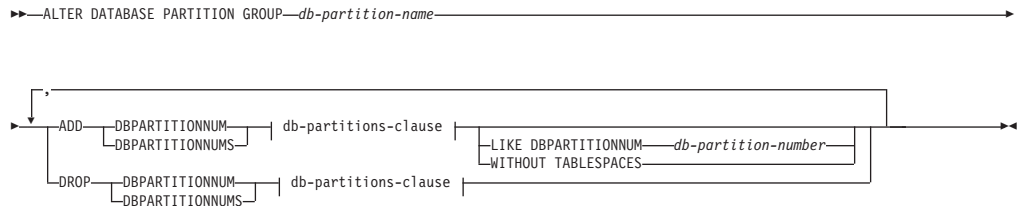
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

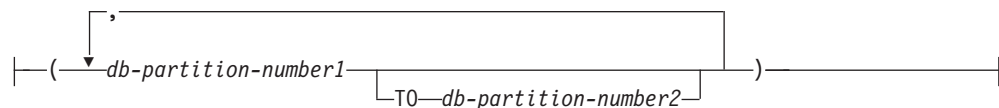
Authorization

The authorization ID of the statement must have SYSCTRL or SYSADM authority.

Syntax



db-partitions-clause:



Description

db-partition-name

Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). It must be a database partition group described in the catalog. IBMCATGROUP and IBMTEMPGROUP cannot be specified (SQLSTATE 42832).

ADD DBPARTITIONNUM

Specifies the specific database partition or partitions to add to the database partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must not already be defined in the database partition group (SQLSTATE 42728).

DROP DBPARTITIONNUM

Specifies the specific database partition or partitions to drop from the database partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must already be defined in the database partition group (SQLSTATE 42729).

db-partitions-clause

Specifies the database partition or partitions to be added or dropped.

db-partition-number1

Specify a specific database partition number.

TO *db-partition-number2*

Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9).

LIKE DBPARTITIONNUM *db-partition-number*

Specifies that the containers for the existing table spaces in the database partition group will be the same as the containers on the specified *db-partition-number*. The specified database partition must be a partition that existed in the database partition group prior to this statement, and that is not included in a DROP DBPARTITIONNUM clause of the same statement.

For table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all), the containers will not necessarily match those from the specified partition. Instead, containers will automatically be assigned by the database manager based on the storage paths that are associated with the database, and this might or might not result in the same containers being used. The size of each table space is based on the initial size that was specified when the table space was created, and might not match the current size of the table space on the specified partition.

WITHOUT TABLESPACES

Specifies that the containers for existing table spaces in the database partition group are not created on the newly added database partition or partitions. The ALTER TABLESPACE statement using the *db-partitions-clause* must be used to define containers for use with the table spaces that are defined on this database partition group. If this option is not specified, the default containers are specified on newly added database partitions for each table space defined on the database partition group.

This option is ignored for table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all). There is no way to defer container creation for these table spaces. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database. The size of each table space will be based on the initial size that was specified when the table space was created.

Rules

- Each database partition specified by number must be defined in the `db2nodes.cfg` file (SQLSTATE 42729).
- Each *db-partition-number* listed in the *db-partitions-clause* must be for a unique database partition (SQLSTATE 42728).
- A valid database partition number is between 0 and 999 inclusive (SQLSTATE 42729).
- A database partition cannot appear in both the ADD and DROP clauses (SQLSTATE 42728).
- There must be at least one database partition remaining in the database partition group. The last database partition cannot be dropped from a database partition group (SQLSTATE 428C0).

- If neither the LIKE DBPARTITIONNUM clause nor the WITHOUT TABLESPACES clause is specified when adding a database partition, the default is to use the lowest database partition number of the existing database partitions in the database partition group (say it is 2) and proceed as if LIKE DBPARTITIONNUM 2 had been specified. For an existing database partition to be used as the default, it must have containers defined for all the table spaces in the database partition group (column IN_USE of SYSCAT.DBPARTITIONGROUPDEF is not 'T').
- The ALTER DATABASE PARTITION GROUP statement might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This statement might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Notes

- When a database partition is added to a database partition group, a catalog entry is made for the database partition (see SYSCAT.DBPARTITIONGROUPDEF). The distribution map is changed immediately to include the new database partition, along with an indicator (IN_USE) that the database partition is in the distribution map if either:
 - no table spaces are defined in the database partition group or
 - no tables are defined in the table spaces defined in the database partition group and the WITHOUT TABLESPACES clause was not specified.
 The distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is not included in the distribution map if either:
 - Tables exist in table spaces in the database partition group or
 - Table spaces exist in the database partition group and the WITHOUT TABLESPACES clause was specified (unless all of the table spaces are defined to use automatic storage, in which case the WITHOUT TABLESPACES clause is ignored)
 To change the distribution map, the REDISTRIBUTE DATABASE PARTITION GROUP command must be used. This redistributes any data, changes the distribution map, and changes the indicator. Table space containers need to be added before attempting to redistribute data if the WITHOUT TABLESPACES clause was specified.
- When a database partition is dropped from a database partition group, the catalog entry for the database partition (see SYSCAT.DBPARTITIONGROUPDEF) is updated. If there are no tables defined in the table spaces defined in the database partition group, the distribution map is changed immediately to exclude the dropped database partition and the entry for the database partition in the database partition group is dropped. If tables exist, the distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is waiting to be dropped. The REDISTRIBUTE DATABASE PARTITION GROUP command must be used to redistribute the data and drop the entry for the database partition from the database partition group.
- **Compatibilities:** For compatibility with previous versions of DB2 products:
 - NODE can be specified in place of DBPARTITIONNUM
 - NODES can be specified in place of DBPARTITIONNUMS
 - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Example

Assume that you have a six-partition database that has the following database partitions: 0, 1, 2, 5, 7, and 8. Two database partitions (3 and 6) are added to the system.

- Assume that you want to add database partitions 3 and 6 to a database partition group called MAXGROUP, and have table space containers like those on database partition 2. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MAXGROUP
ADD DBPARTITIONNUMS (3,6)LIKE DBPARTITIONNUM 2
```

- Assume that you want to drop database partition 1 and add database partition 6 to database partition group MEDGROUP. You will define the table space containers separately for database partition 6 using ALTER TABLESPACE. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MEDGROUP
ADD DBPARTITIONNUM(6)WITHOUT TABLESPACES
DROP DBPARTITIONNUM(1)
```

ALTER FUNCTION

The ALTER FUNCTION statement modifies the properties of an existing function.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- ALTERIN privilege on the schema of the function
- Owner of the function, as recorded in the OWNER column of the SYSCAT.ROUTINES catalog view
- DBADM authority

To alter the EXTERNAL NAME of a function, the privileges held by the authorization ID of the statement must also include at least one of the following:

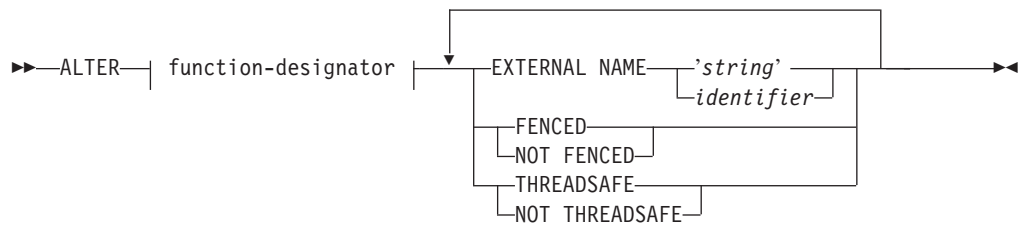
- CREATE_EXTERNAL_ROUTINE authority on the database
- DBADM authority

To alter a function to be not fenced, the privileges held by the authorization ID of the statement must also include at least one of the following:

- CREATE_NOT_FENCED_ROUTINE authority on the database
- DBADM authority

To alter a function to be fenced, no additional authorities or privileges are required.

Syntax



Description

function-designator

Uniquely identifies the function to be altered. For more information, see “Function, method, and procedure designators” on page 497.

EXTERNAL NAME 'string' or identifier

Identifies the name of the user-written code that implements the function. This option can only be specified when altering external functions (SQLSTATE 42849).

FENCED or NOT FENCED

Specifies whether the function is considered safe to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED). Most functions have the option of running as FENCED or NOT FENCED.

If a function is altered to be FENCED, the database manager insulates its internal resources (for example, data buffers) from access by the function. In general, a function running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:

Use of NOT FENCED for functions that were not adequately coded, reviewed, and tested can compromise the integrity of a DB2 database. DB2 databases take some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED user-defined functions are used.

A function declared as NOT THREADSAFE cannot be altered to be NOT FENCED (SQLSTATE 42613).

If a function has any parameters defined AS LOCATOR, and was defined with the NO SQL option, the function cannot be altered to be FENCED (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE, OLEDB, or CLR functions (SQLSTATE 42849).

THREADSAFE or NOT THREADSAFE

Specifies whether the function is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the function is defined with LANGUAGE other than OLE and OLEDB:

- If the function is defined as THREADSAFE, the database manager can invoke the function in the same process as other routines. In general, to be threadsafe, a function should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED functions can be THREADSAFE.

- If the function is defined as NOT THREADSAFE, the database manager will never simultaneously invoke the function in the same process as another routine. Only a fenced function can be NOT THREADSAFE (SQLSTATE 42613).

This option may not be altered for LANGUAGE OLE or OLEDB functions (SQLSTATE 42849).

Notes

- It is not possible to alter a function that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).
- Functions declared as LANGUAGE SQL, sourced functions, or template functions cannot be altered (SQLSTATE 42917).

Example

The function MAIL() has been thoroughly tested. To improve its performance, alter the function to be not fenced.

```
ALTER FUNCTION MAIL() NOT FENCED
```

ALTER METHOD

The ALTER METHOD statement modifies an existing method by changing the method body associated with the method.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CREATE_EXTERNAL_ROUTINE authority on the database, and at least one of:
 - ALTERIN privilege on the schema of the type
 - Owner of the type, as recorded in the OWNER column of the SYSCAT.DATATYPES catalog view
- DBADM authority

Syntax

```
➤➤ ALTER | method-designator | EXTERNAL NAME 'string' | identifier ➤➤
```

Description

method-designator

Uniquely identifies the method to be altered. For more information, see “Function, method, and procedure designators” on page 497.

EXTERNAL NAME 'string' or identifier

Identifies the name of the user-written code that implements the method. This option can only be specified when altering external methods (SQLSTATE 42849).

Notes

- It is not possible to alter a method that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).
- Methods declared as LANGUAGE SQL cannot be altered (SQLSTATE 42917).
- Methods declared as LANGUAGE CLR cannot be altered (SQLSTATE 42849).
- The specified method must have a body before it can be altered (SQLSTATE 42704).

Example

Alter the method DISTANCE() in the structured type ADDRESS_T to use the library newaddresslib.

```
ALTER METHOD DISTANCE()  
FOR TYPE ADDRESS_T  
EXTERNAL NAME 'newaddresslib!distance2'
```

ALTER PROCEDURE (External)

The ALTER PROCEDURE (External) statement modifies an existing external procedure by changing the properties of the procedure.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- ALTERIN privilege on the schema of the procedure
- Owner of the procedure, as recorded in the OWNER column of the SYSCAT.ROUTINES catalog view
- DBADM authority

To alter the EXTERNAL NAME of a procedure, the privileges held by the authorization ID of the statement must also include at least one of the following:

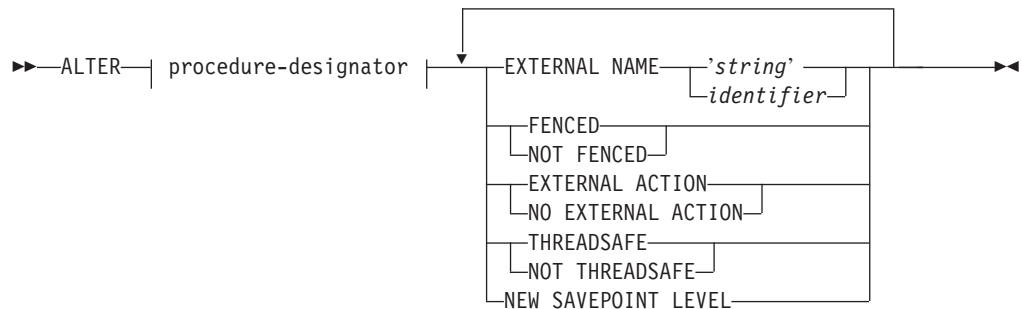
- CREATE_EXTERNAL_ROUTINE authority on the database
- DBADM authority

To alter a procedure to be not fenced, the privileges held by the authorization ID of the statement must also include at least one of the following:

- CREATE_NOT_FENCED_ROUTINE authority on the database
- DBADM authority

To alter a procedure to be fenced, no additional authorities or privileges are required.

Syntax



Description

procedure-designator

Identifies the procedure to alter. The *procedure-designator* must identify a procedure that exists at the current server. The owner of the procedure and all privileges on the procedure are preserved. For more information, see “Function, method, and procedure designators” on page 497.

EXTERNAL NAME 'string' or identifier

Identifies the name of the user-written code that implements the procedure.

FENCED or NOT FENCED

Specifies whether the procedure is considered safe to run in the database manager operating environment’s process or address space (NOT FENCED), or not (FENCED). Most procedures have the option of running as FENCED or NOT FENCED.

If a procedure is altered to be FENCED, the database manager insulates its internal resources (for example, data buffers) from access by the procedure. In general, a procedure running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:

Use of NOT FENCED for procedures that were not adequately coded, reviewed, and tested can compromise the integrity of a DB2 database. DB2 databases take some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED stored procedures are used.

A procedure declared as NOT THREADSAFE cannot be altered to be NOT FENCED (SQLSTATE 42613).

If a procedure has any parameters defined AS LOCATOR, and was defined with the NO SQL option, the procedure cannot be altered to be FENCED (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE or CLR procedures (SQLSTATE 42849).

EXTERNAL ACTION or NO EXTERNAL ACTION

Specifies whether the procedure takes some action that changes the state of an object not managed by the database manager (EXTERNAL ACTION), or not

(NO EXTERNAL ACTION). If NO EXTERNAL ACTION is specified, the system can use certain optimizations that assume the procedure has no external impact.

THREADSAFE or NOT THREADSAFE

Specifies whether the procedure is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the procedure is defined with LANGUAGE other than OLE:

- If the procedure is defined as THREADSAFE, the database manager can invoke the procedure in the same process as other routines. In general, to be threadsafe, a procedure should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED procedures can be THREADSAFE.
- If the procedure is defined as NOT THREADSAFE, the database manager will never invoke the procedure in the same process as another routine. Only a fenced procedure can be NOT THREADSAFE (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE procedures (SQLSTATE 42849).

NEW SAVEPOINT LEVEL

Specifies that a new savepoint level is to be created for the procedure. A savepoint level refers to the scope of reference for any savepoint-related statement, as well as to the name space used for comparison and reference of any savepoint names.

The savepoint level for a procedure can only be altered to NEW SAVEPOINT LEVEL.

Rules

- It is not possible to alter a procedure that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

Example

Alter the procedure PARTS_ON_HAND() to be not fenced.

```
ALTER PROCEDURE PARTS_ON_HAND() NOT FENCED
```

ALTER SECURITY LABEL COMPONENT

The ALTER SECURITY LABEL COMPONENT statement modifies a security label component.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax

▶▶ALTER SECURITY LABEL COMPONENT—*component-name*—| add-element-clause |▶▶▶

add-element-clause:

|—ADD ELEMENT—*string-constant*—|
|
| array-element-clause |
| tree-element-clause |

array-element-clause:

|—BEFORE—|
|—AFTER—| *string-constant*—|

tree-element-clause:

|—ROOT—|
|—UNDER—*string-constant*—|
|
| OVER—*string-constant*—|

Description

component-name

Specifies the name of the security label component to be altered. The named component must exist at the current server (SQLSTATE 42704).

ADD ELEMENT

Specifies the element to be added to the security label component. If *array-element-clause* and *tree-element-clause* are not specified, the element is added to a set component.

string-constant

The string constant value to be added to the set of valid values for the security label component. The value cannot be the same as any other value in the set of valid values for the security label component (SQLSTATE 42713).

BEFORE or AFTER

For an array component, specifies where the element is to be added in the ordered set of element values for the security label component.

BEFORE

The element to be added is to be ranked immediately before the identified existing element.

AFTER

The element to be added is to be ranked immediately after the identified existing element.

string-constant

Specifies a string constant value of an existing element in the array component (SQLSTATE 42704).

ROOT or UNDER

For a tree component, specifies where the element is to be added in the tree structure of node element values for the security label component.

ROOT

The element to be added is to be considered the root node of the tree.

UNDER *string-constant*

The element to be added is an immediate child of the element identified by the *string-constant*. The *string-constant* value must be an existing element in the tree component (SQLSTATE 42704).

OVER *string-constant,...*

The element to be added is an immediate child of every element identified by the list of *string-constant* values. Each *string-constant* value must be an existing element in the tree component (SQLSTATE 42704).

Rules

- Element names cannot contain any of these characters (SQLSTATE 42601):
 - Opening parenthesis - (
 - Closing parenthesis -)
 - Comma - ,
 - Colon - :
- An element name can have no more than 32 bytes (SQLSTATE 42622).
- If a security label component is a set or a tree, no more than 64 elements can be part of that component.
- If the component is an array, it might or might not be possible to arrive at an array whose total number of elements matches the total number of elements that could be specified when creating a security label component of type array (65 535). DB2 assigns an encoded value to the new element from within the interval into which the new element is added. Depending on the pattern followed when adding elements to an array component, the number of possible values that can be assigned from within a particular interval might be quickly exhausted if several elements are inserted into that interval.
- BEFORE and AFTER must only be specified for a security label component that is an array (SQLSTATE 42613).
- ROOT and UNDER must only be specified for a security label component that is a tree (SQLSTATE 42613).

Notes

- For a set component, there is no order to the elements in the set.

Examples

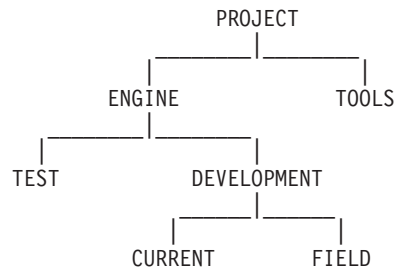
Example 1: Add the element 'High classified' to the LEVEL security label array component between the elements 'Secret' and 'Classified'.

```
ALTER SECURITY LABEL COMPONENT LEVEL
ADD ELEMENT 'High classified' BEFORE 'Classified'
```

Example 2: Add the element 'Funding' to the COMPARTMENTS security label set component.

```
ALTER SECURITY LABEL COMPONENT COMPARTMENTS
ADD ELEMENT 'Funding'
```

Example 3: Add the elements 'ENGINE' and 'TOOLS' to the GROUPS security label array component. The following diagram shows where these new elements are to be placed.



```
ALTER SECURITY LABEL COMPONENT GROUPS
ADD ELEMENT 'TOOLS' UNDER 'PROJECT'
```

```
ALTER SECURITY LABEL COMPONENT GROUPS
ADD ELEMENT 'ENGINE' UNDER 'PROJECT'
OVER 'TEST', 'DEVELOPMENT'
```

ALTER SECURITY POLICY

The ALTER SECURITY POLICY statement modifies a security policy.

Invocation

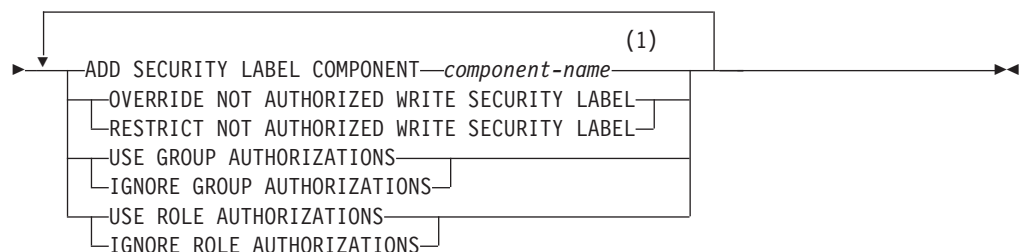
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax

►► ALTER SECURITY POLICY—*security-policy-name*—►►



Notes:

- 1 Only the ADD SECURITY LABEL COMPONENT clause can be specified more than once.

Description

security-policy-name

Specifies the name of the security policy to be altered. The name must identify an existing security policy at the current server (SQLSTATE 42710).

ADD SECURITY LABEL COMPONENT *component-name*

Adds a security label component to the security policy. The same security component must not be specified more than once for the security policy (SQLSTATE 42713). The security policy cannot currently be in use by a table (SQLSTATE 42893).

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL or RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL

Specifies the action taken when a user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement issued against a table that is protected with this security policy. A user's security label and exemption credentials determine the user's authorization to write an explicitly provided security label.

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL

Indicates that the value of the user's security label, rather than the explicitly specified security label, is used for write access during an insert or update operation.

RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL

Indicates that the insert or update operation will fail if the user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement (SQLSTATE 42519).

USE GROUP AUTHORIZATION or IGNORE GROUP AUTHORIZATION

Specifies whether or not security labels and exemptions granted to groups, directly or indirectly, are considered for any access attempt.

USE GROUP AUTHORIZATION

Indicates that any security labels or exemptions granted to groups, directly or indirectly, are considered.

IGNORE GROUP AUTHORIZATION

Indicates that any security labels or exemptions granted to groups are not considered.

USE ROLE AUTHORIZATION or IGNORE ROLE AUTHORIZATION

Specifies whether or not security labels and exemptions granted to roles, directly or indirectly, are considered for any access attempt.

USE ROLE AUTHORIZATION

Indicates that any security labels or exemptions granted to roles, directly or indirectly, are considered.

IGNORE ROLE AUTHORIZATION

Indicates that any security labels or exemptions granted to roles are not considered.

Rules

- If a user does not directly hold a security label for write access, an error is returned in the following situations (SQLSTATE 42519):
 - A value for the row security label column is not explicitly provided as part of the SQL statement

- The `OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL` option is in effect for the security policy, and the user is not allowed to write a data object with the provided security label

Notes

- New components are logically added at the end of the existing security label definition contained by the modified policy. Existing security labels defined for this security policy are modified to contain the new component as part of their definition with no element in their value for this component.
- *Cache invalidation when changing NOT AUTHORIZED WRITE SECURITY LABEL:* Changing the `NOT AUTHORIZED WRITE SECURITY LABEL` to a new value will cause the invalidation of any cached dynamic or static SQL statements that are dependent on any table that is protected by the security policy being altered.
- Because the session authorization ID is the focus authorization ID for label-based access control, security labels granted to groups or to roles that are accessible through groups are eligible for consideration for all types of SQL statements, including static SQL.
- If more than one security label or exemption is available to a user with associated groups or roles at the time of a read or write access attempt, those security labels and exemptions will be evaluated for eligibility based on the following rules:
 - If the security policy enables only role authorizations for consideration, all security labels and exemptions granted to roles of which the user authorization ID is a direct or indirect member will be considered. Security labels and exemptions granted to roles for which membership is only accessible through the groups associated with the user authorization ID will not be considered.
 - If the security policy enables only group authorizations for consideration, all security labels and exemptions granted to groups associated with the user authorization ID will be considered. Security labels and exemptions granted to roles for which membership is only accessible through the groups associated with the user authorization ID will not be considered.
 - If the security policy enables both group and role authorizations for consideration, any security labels and exemptions granted to roles accessible to the user indirectly through groups associated with the user authorization ID will be considered.
 - Role authorizations that are accessible to the user only through `PUBLIC` will not be considered at any time.
- If more than one security label is eligible for consideration during an access attempt, the values provided for each security label are merged at the individual component level to form a security label that reflects the combination of all available values at each component piece of the security policy. This is the security label value that will be used for the access attempt.
The mechanisms for combining security labels vary by component type. The components of the resultant security label are as follows:
 - Set components contain the union of all unique values encountered in the eligible security labels
 - Array components contain the highest order element encountered in the eligible security labels
 - Tree components contain the union of all unique values encountered in the eligible security labels

- If more than one exemption is eligible for consideration during an access attempt, all found exemptions are applied to the access attempt.

Examples

Example 1: Alter a security policy named DATA_ACCESS to add a new component named REGION.

```
ALTER SECURITY POLICY DATA_ACCESS
ADD COMPONENT REGION
```

Example 2: Alter a security policy named DATA_ACCESS to allow access through security labels granted to roles.

```
ALTER SECURITY POLICY DATA_ACCESS
USE ROLE AUTHORIZATIONS
```

Example 3: Show the eligible security labels that would be considered depending on the settings for group or role authorizations in a security policy. The security policy SECUR_POL has an array component and a set component, consisting of the following elements:

Array = {TS, S, C, U}
Set = {A, B, X, Y}

The following security labels are defined for SECUR_POL:

Security label L1 = C:A
Security label L2 = S:B
Security label L3 = TS:X
Security label L4 = U:Y

User Paul is a member of role R1 and group G1. Group G1 is a member of role R2. Security label L1 is granted to Paul. Security label L2 is granted to role R1. Security label L3 is granted to group G1. Security label L4 is granted to role R2. The following table shows what security labels would be considered for any access attempt by Paul, depending on the different possible settings of the security policy SECUR_POL.

Table 2. Security labels considered as a function of security policy settings

	Roles Enabled	Roles Disabled
Groups Enabled	L1, L2, L3, L4	L1, L3
Groups Disabled	L1, L2	L1

The following table shows the value of the combined security label for any access attempt by Paul, depending on the different settings of the security policy SECUR_POL.

Table 3. Combined security labels as a function of security policy settings

	Roles Enabled	Roles Disabled
Groups Enabled	TS:(A, B, X, Y)	TS:(A, X)
Groups Disabled	S:(A, B)	C:A

ALTER TABLE

The ALTER TABLE statement alters the definition of a table.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- ALTER privilege on the table to be altered
- CONTROL privilege on the table to be altered
- ALTERIN privilege on the schema of the table
- DBADM authority

To create or drop a foreign key, the privileges held by the authorization ID of the statement must include one of the following on the parent table:

- REFERENCES privilege on the table
- REFERENCES privilege on each column of the specified parent key
- CONTROL privilege on the table
- DBADM authority

To drop the primary key or a unique constraint on table T, the privileges held by the authorization ID of the statement must include at least one of the following on every table that is a dependent of this parent key of T:

- ALTER privilege on the table
- CONTROL privilege on the table
- ALTERIN privilege on the schema of the table
- DBADM authority

To alter a table to become a materialized query table (using a fullselect), the privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the table
- DBADM authority

and at least one of the following on each table or view identified in the fullselect (excluding group privileges):

- SELECT privilege and ALTER privilege (including group privileges) on the table or view
- CONTROL privilege on the table or view
- SELECT privilege on the table or view, and ALTERIN privilege (including group privileges) on the schema of the table or view
- DATAACCESS authority

To alter a table so that it is no longer a materialized query table, the privileges held by the authorization ID of the statement must include at least one of the following on each table or view identified in the fullselect used to define the materialized query table:

- ALTER privilege on the table or view

- CONTROL privilege on the table or view
- ALTERIN privilege on the schema of the table or view
- DBADM authority

To add a column of type DB2SECURITYLABEL to a table, the privileges held by the authorization ID of the statement must include at least a security label from the security policy associated with the table.

To remove the security policy from a table, the privileges held by the authorization ID of the statement must include SECADM authority.

To alter a table to attach a data partition, the privileges held by the authorization ID of the statement must also include at least one of the following on the source table:

- SELECT privilege on the table and DROPIN privilege on the schema of the table
- CONTROL privilege on the table
- DATAACCESS authority

and at least one of the following on the target table:

- ALTER and INSERT privileges on the table
- CONTROL privilege on the table
- DATAACCESS authority

To alter a table to detach a data partition, the privileges held by the authorization ID of the statement must also include at least one of the following on the target table of the detached partition:

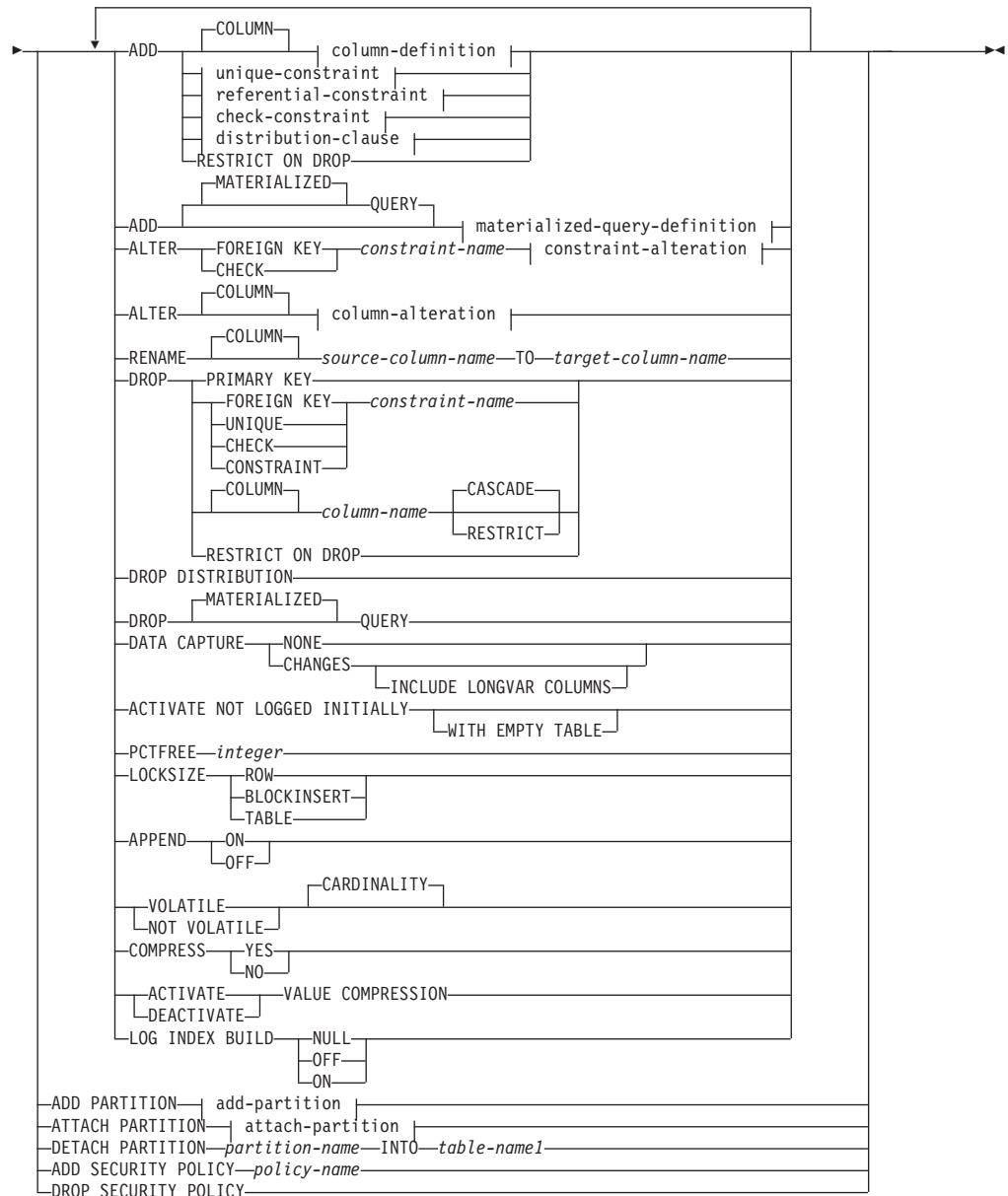
- CREATETAB authority on the database, and USE privilege on the table spaces used by the table, as well as one of:
 - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the new table does not exist
 - CREATEIN privilege on the schema, if the schema name of the new table refers to an existing schema
- DBADM authority

and at least one of the following on the source table:

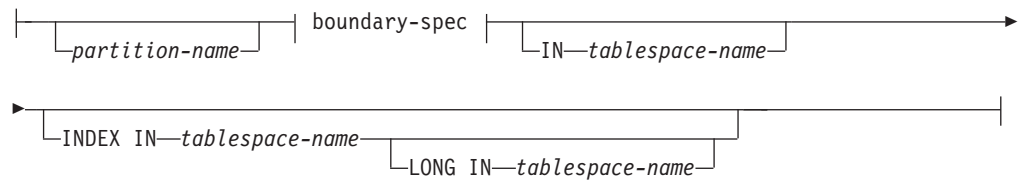
- SELECT, ALTER, and DELETE privileges on the table
- CONTROL privilege on the table
- DATAACCESS authority

Syntax

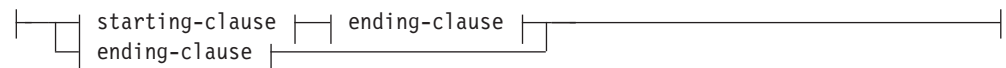
▶▶—ALTER TABLE—*table-name*—————▶



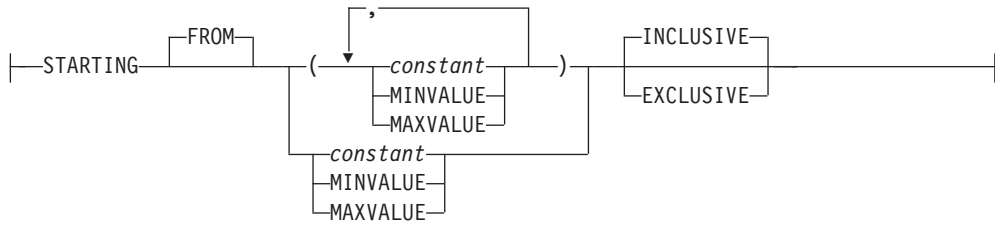
add-partition:



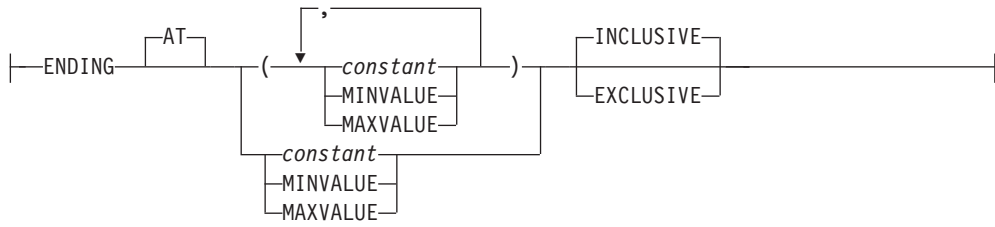
boundary-spec:



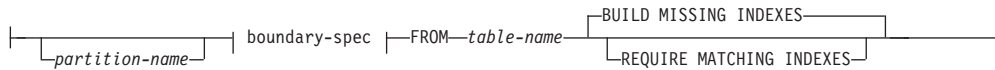
starting-clause:



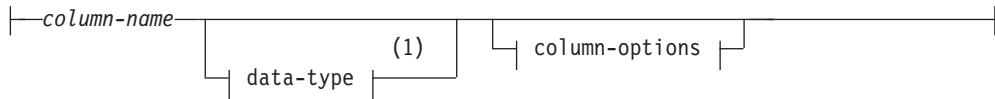
ending-clause:



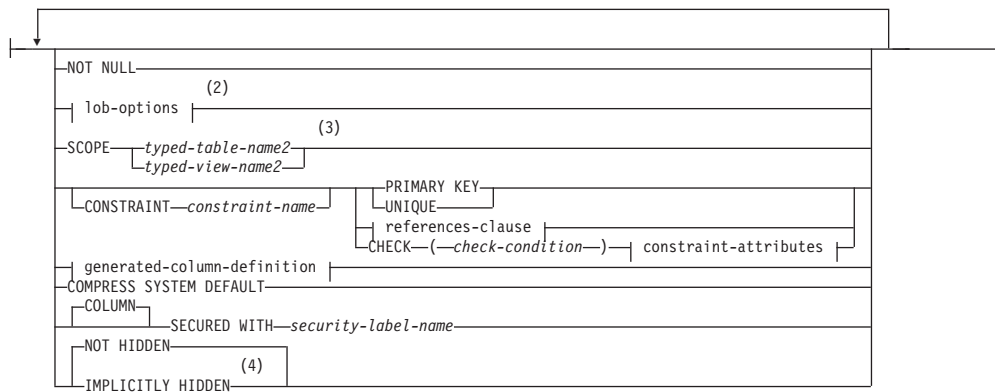
attach-partition:



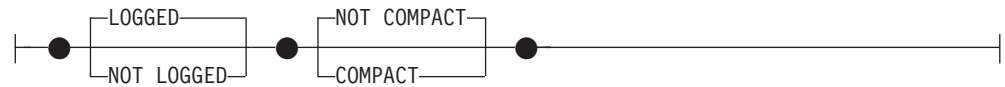
column-definition:



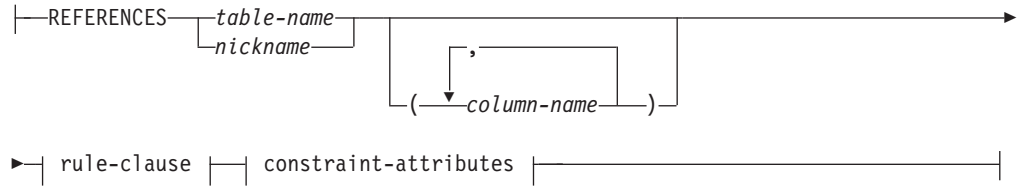
column-options:



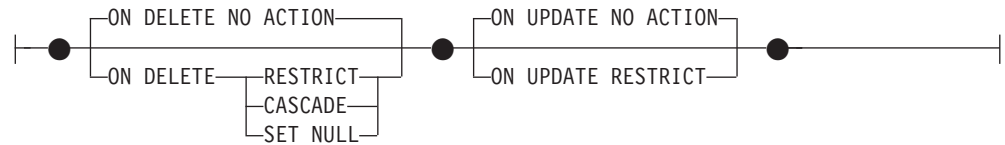
lob-options:



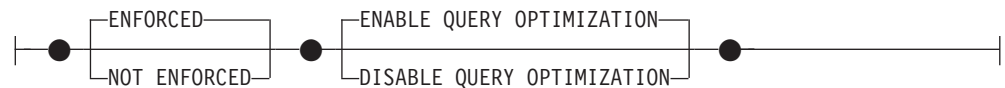
references-clause:



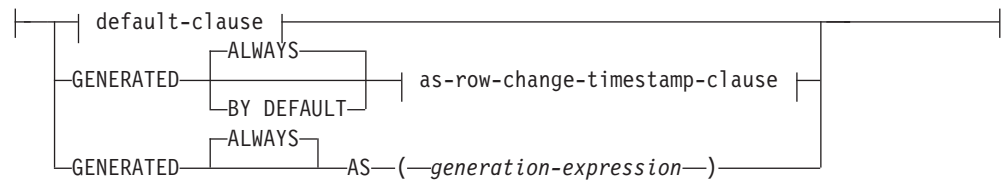
rule-clause:



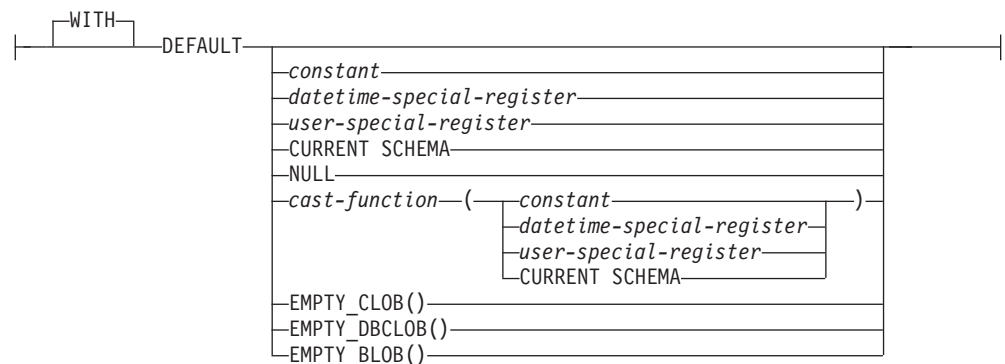
constraint-attributes:



generated-column-definition:



default-clause:



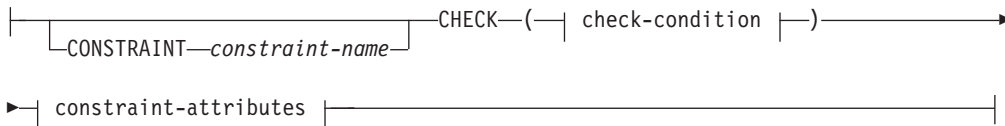
unique-constraint:



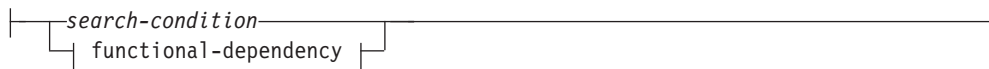
referential-constraint:



check-constraint:



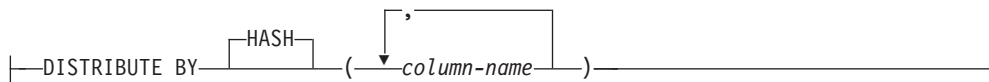
check-condition:



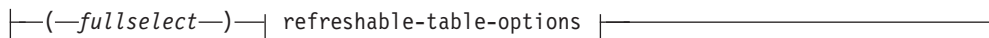
functional-dependency:



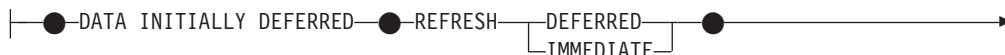
distribution-clause:



materialized-query-definition:

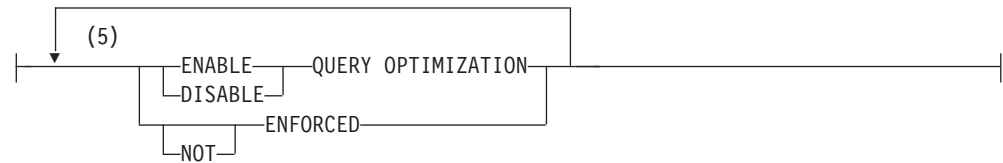


refreshable-table-options:

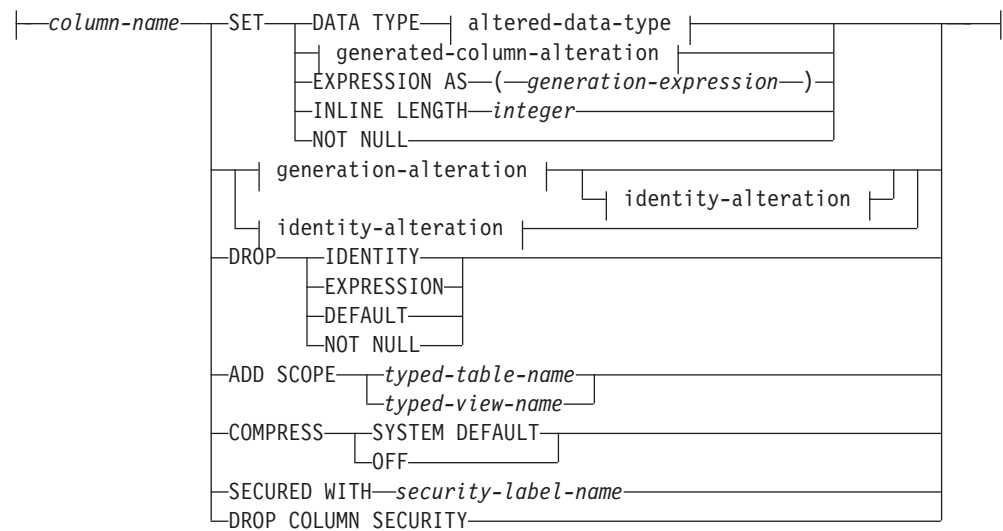




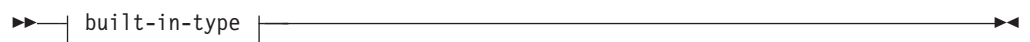
constraint-alteration:



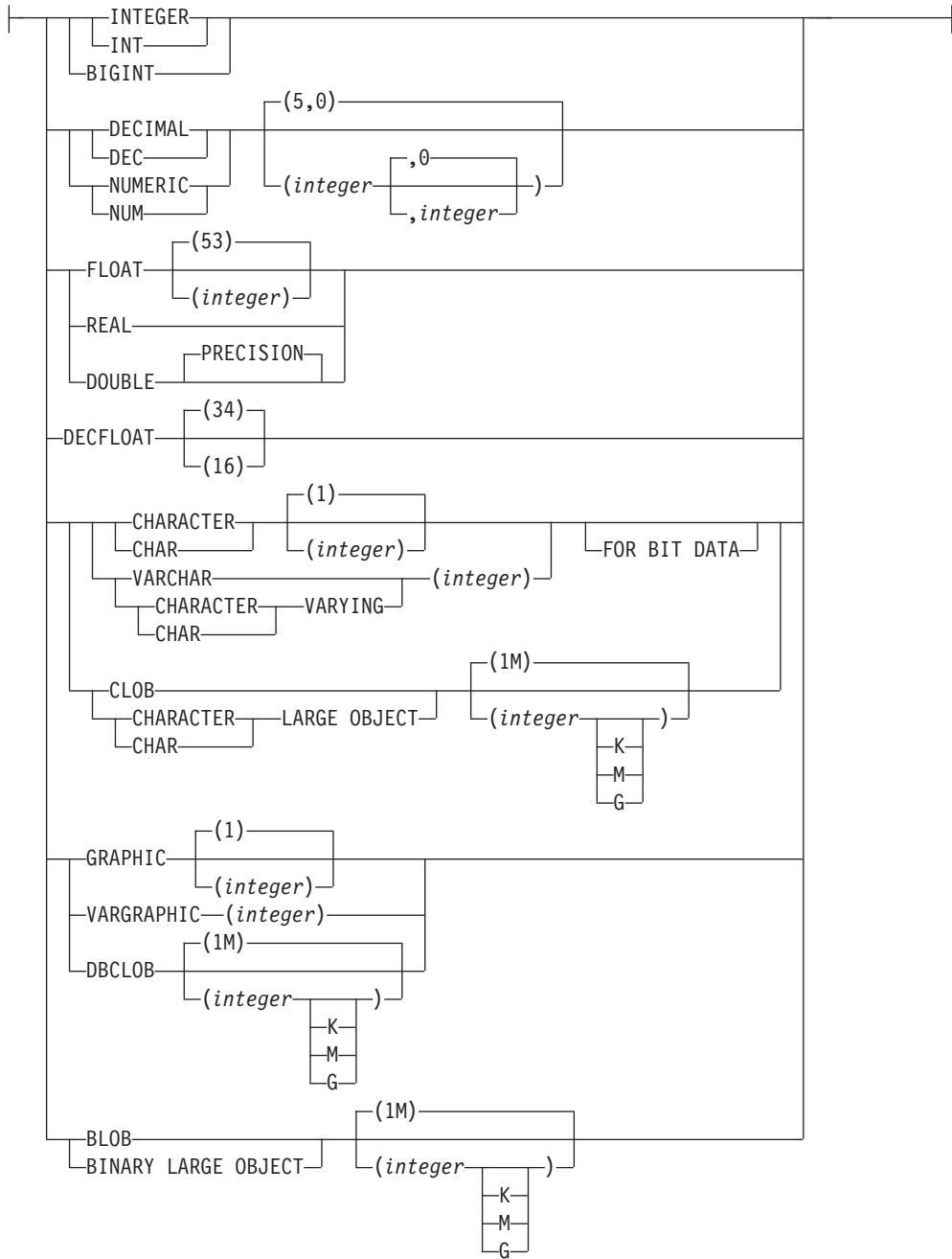
column-alteration:



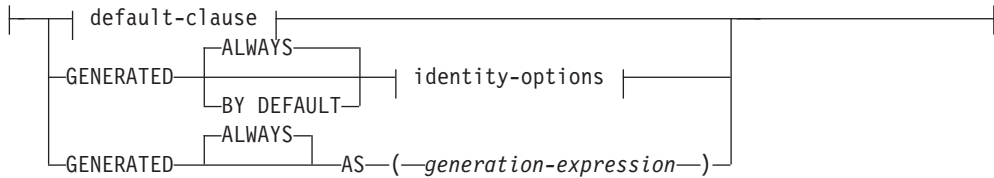
altered-data-type:



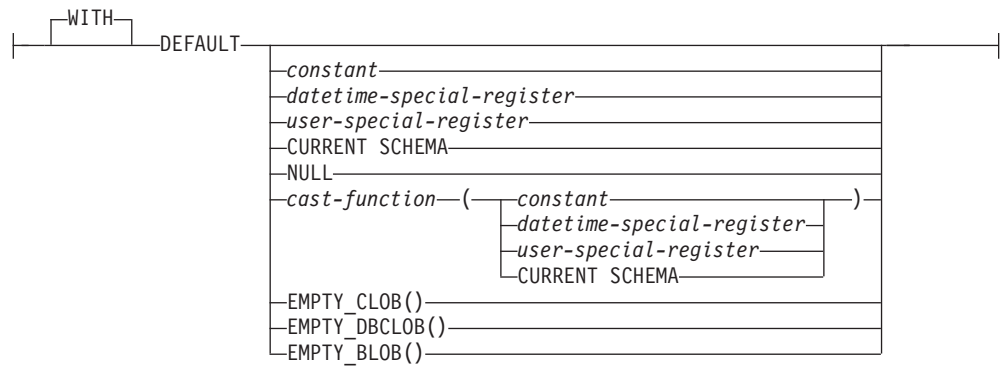
built-in-type:



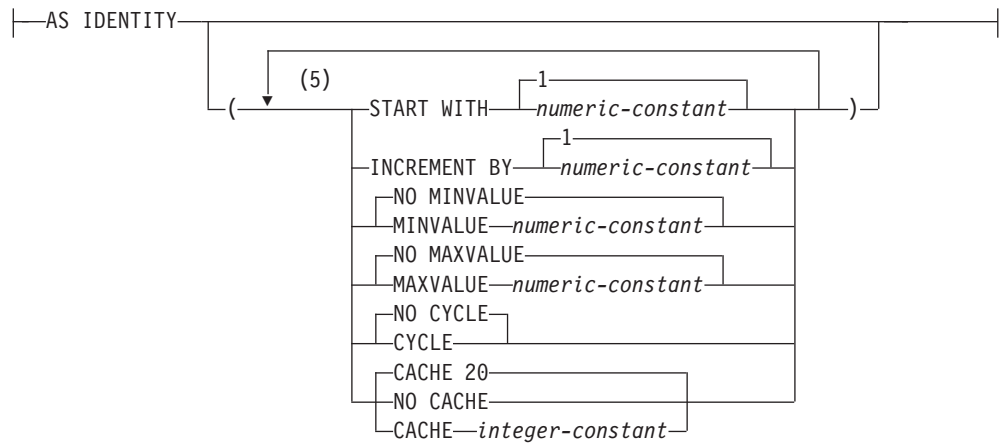
generated-column-alteration:



default-clause:



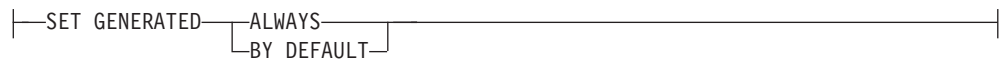
identity-options:



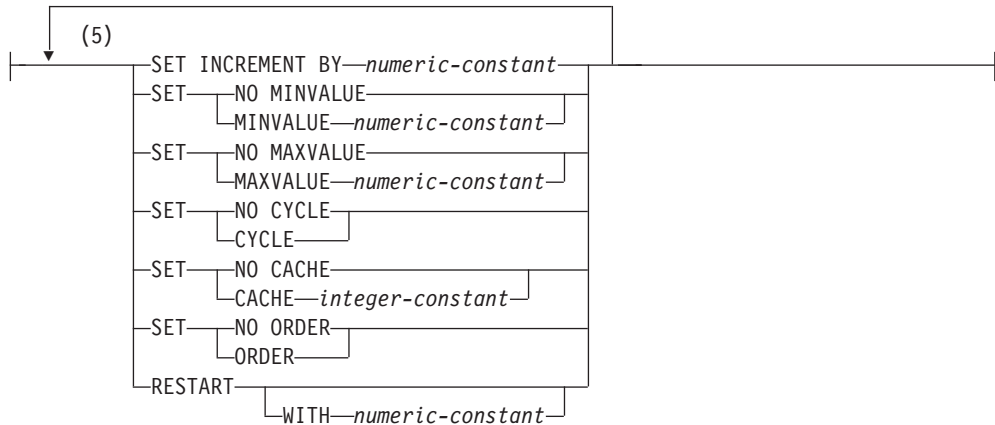
as-row-change-timestamp-clause:



generation-alteration:



identity-alteration:



Notes:

- 1 If the first column option chosen is *generated-column-definition*, *data-type* can be omitted; it will be computed by the generation expression.
- 2 The *lob-options* clause only applies to large object types (CLOB, DBCLOB, and BLOB), and to distinct types that are based on large object types.
- 3 The SCOPE clause only applies to the REF type.
- 4 IMPLICITLY HIDDEN can only be specified if ROW CHANGE TIMESTAMP is also specified.
- 5 The same clause must not be specified more than once.
- 6 Data type is optional for a row change timestamp column if the first column-option specified is a generated-column-definition, the data type default is TIMESTAMP(6).

Description

table-name

The *table-name* must identify a table that exists at the current server. It cannot be a nickname (SQLSTATE 42809) and must not be a view, a catalog table, a created temporary table, or a declared temporary table (SQLSTATE 42995).

If *table-name* identifies a materialized query table, alterations are limited to adding or dropping the materialized query table, activating not logged initially, adding or dropping RESTRICT ON DROP, and changing pctfree, locksize, append, or volatile.

If *table-name* identifies a range-clustered table, alterations are limited to adding, changing, or dropping constraints, activating not logged initially, adding or dropping RESTRICT ON DROP, changing locksize, data capture, or volatile, and setting column default values.

ADD PARTITION *add-partition*

Adds one or more data partitions to a partitioned table. If the specified table is not a partitioned table, an error is returned (SQLSTATE 428FT). The number of data partitions must not exceed 32 767.

partition-name

Names the data partition. The name must not be the same as any other data partition for the table (SQLSTATE 42710). If this clause is not specified, the name will be 'PART' followed by the character form of an integer value to make the name unique for the table.

boundary-spec

Specifies the range of values for the new data partition. This range must not overlap that of an existing data partition (SQLSTATE 56016). For a description of the starting-clause and the ending-clause, see “CREATE TABLE”.

If the starting-clause is omitted, the new data partition is assumed to be at the end of the table. If the ending-clause is omitted, the new data partition is assumed to be at the start of the table. If the first column of the partitioning key is DESC, these assumptions are reversed.

IN *tablespace-name*

Specifies the table space where the data partition is to be stored. The named table space must have the same page size, be in the same database partition group, and manage space in the same way as the other table spaces of the partitioned table (SQLSTATE 42838). This can be a table space that is already being used for another data partition of the same table, or a table space that is currently not being used by this table, but it must be a table space on which the authorization ID of the statement holds the USE privilege (SQLSTATE 42727). If this clause is not specified, the table space of the first visible or attached data partition of the table is used.

INDEX IN *tablespace-name*

Specifies the table space where partitioned indexes on the data partition are stored. If the INDEX IN clause is not specified, partitioned indexes on the data partition are stored in the same table space as the data partition.

The table space used by the new index partition, whether default or specified by the INDEX IN clause, must match the type (SMS or DMS), page size, and extent size of the table spaces used by all other index partitions (SQLSTATE 42838).

LONG IN *tablespace-name*

Specifies the table space where the data partition containing long column data is to be stored. The named table space must have the same page size, be in the same database partition group, and manage space in the same way as the other table spaces and data partitions of the partitioned table (SQLSTATE 42838); it must be a table space on which the authorization ID of the statement holds the USE privilege. The page size and extent size for the named table space can be different from the page size and extent size of the other data partitions of the partitioned table.

For rules governing the use of the LONG IN clause with partitioned tables, see “Large object behavior in partitioned tables”.

ATTACH PARTITION *attach-partition*

Attaches another table as a new data partition. The data object of the table being attached becomes a new partition of the table being attached to. There is no data movement involved. The table is placed in set integrity pending state, and referential integrity checking is deferred until execution of a SET INTEGRITY statement. The ALTER TABLE ATTACH operation does not allow the use of the IN or LONG IN clause. The placement of LOBs for that data partition is determined at the time the source table is created. For rules governing the use of the LONG IN clause with partitioned tables, see “Large object behavior in partitioned tables”.

partition-name

Names the data partition. The name must not be the same as any other data partition for the table (SQLSTATE 42710). If this clause is not

specified, the name will be 'PART' followed by the character form of an integer value to make the name unique for the table.

boundary-spec

Specifies the range of values for the new data partition. This range must not overlap that of an existing data partition (SQLSTATE 56016). For a description of the starting-clause and the ending-clause, see "CREATE TABLE".

If the starting-clause is omitted, the new data partition is assumed to be at the end of the table. If the ending-clause is omitted, the new data partition is assumed to be at the start of the table.

FROM *table-name1*

Specifies the table that is to be used as the source of data for the new partition. The table definition of *table-name1* cannot have multiple data partitions, and it must match the altered table in the following ways (SQLSTATE 428G3):

- The number of columns must be the same.
- The data types of the columns in the same ordinal position in the table must be the same.
- The nullability characteristic of the columns in the same ordinal position in the table must be the same.
- If the data is also distributed, it must be distributed over the same database partition group using the same distribution method.
- If the data in either table is organized, the organization must match.
- For structured, XML, or LOB data type, the value for `INLINE LENGTH` must be the same.

After the data from *table-name1* is successfully attached, an operation equivalent to `DROP TABLE table-name1` is performed to remove this table, which no longer has data, from the database.

BUILD MISSING INDEXES

Specifies that if the source table does not have indexes that correspond to the partitioned indexes on the target table, a `SET INTEGRITY` operation builds partitioned indexes on the new data partition to correspond to the partitioned indexes on the existing data partitions. Indexes on the source table that do not match the partitioned indexes on the target table are dropped during attach processing.

REQUIRE MATCHING INDEXES

Specifies that the source table must have indexes to match the partitioned indexes on the target table; otherwise, an error is returned (SQLSTATE 428GE) and information is written to the administration log about the indexes that do not match.

If the `REQUIRE MATCHING INDEXES` clause is not specified and the indexes on the source table do not match all the partitioned indexes on the target table, the following behavior occurs:

1. For indexes on the target table that do not have a match on the source table and are either unique indexes or XML indexes that are defined with `REJECT INVALID VALUES`, the `ATTACH` operation fails (SQLSTATE 428GE).
2. For all other indexes on the target table that do not have a match on the source table, the index object on the source table is marked invalid during the attach operation. If the source table does not have any

indexes, an empty index object is created and marked as invalid. The ATTACH operation will succeed, but the index object on the new data partition is marked as invalid. Typically, SET INTEGRITY is the next operation to run against the data partition. SET INTEGRITY will force a rebuild, if required, of the index object on data partitions that were recently attached. The index rebuild can increase the time required to bring the new data online.

3. Information is written to the administration log about the indexes that do not match

DETACH PARTITION *partition-name* **INTO** *table-name1*

Detaches the data partition *partition-name* from the altered table, and uses the data partition to create a new table named *table-name1*. The data partition is logically attached to the new table without any data movement. The specified data partition cannot be the last remaining partition of the table being altered (SQLSTATE 428G2).

ADD SECURITY POLICY *policy-name*

Adds a security policy to the table. The security policy must exist at the current server (SQLSTATE 42704). The table must not already have a security policy (SQLSTATE 55065), and must not be a typed table (SQLSTATE 428DH), materialized query table (MQT), or staging table (SQLSTATE 428FG).

DROP SECURITY POLICY

Removes the security policy and all LBAC protection from the table. The table specified by *table-name* must be protected by a security policy (SQLSTATE 428GT). If the table has a column with data type DB2SECURITYLABEL, the data type is changed to VARCHAR (128) FOR BIT DATA. If the table has one or more protected columns, those columns become unprotected.

ADD *column-definition*

Adds a column to the table. The table must not be a typed table (SQLSTATE 428DH). For all existing rows in the table, the value of the new column is set to its default value. The new column is the last column of the table; that is, if initially there are *n* columns, the added column is column *n+1*.

Adding the new column must not make the total byte count of all columns exceed the maximum record size.

column-name

Is the name of the column to be added to the table. The name cannot be qualified. Existing column names in the table cannot be used (SQLSTATE 42711).

data-type

Is one of the data types listed under "CREATE TABLE".

NOT NULL

Prevents the column from containing null values. The *default-clause* must also be specified (SQLSTATE 42601).

NOT HIDDEN or IMPLICITLY HIDDEN

Specifies whether or not the column is to be defined as hidden. The hidden attribute determines whether the column is included in an implicit reference to the table, or whether it can be explicitly referenced in SQL statements. The default is NOT HIDDEN.

NOT HIDDEN

Specifies that the column is included in implicit references to the table, and that the column can be explicitly referenced.

IMPLICITLY HIDDEN

Specifies that the column is not visible in SQL statements unless the column is explicitly referenced by name. For example, assuming that a table includes a column defined with the IMPLICITLY HIDDEN clause, the result of a SELECT * does not include the implicitly hidden column. However, the result of a SELECT that explicitly refers to the name of an implicitly hidden column will include that column in the result table.

IMPLICITLY HIDDEN must only be specified for a ROW CHANGE TIMESTAMP column (SQLSTATE 42867). The ROW CHANGE TIMESTAMP FOR *table-designator* expression will resolve to an IMPLICITLY HIDDEN ROW CHANGE TIMESTAMP column. Therefore, a ROW CHANGE TIMESTAMP column can be added to a table as IMPLICITLY HIDDEN, and existing applications that do a SELECT * from this table will not need to be modified to handle the column. Using the expression, new applications can always access the column without knowing the column name.

lob-options

Specifies options for LOB data types. See *lob-options* in "CREATE TABLE".

SCOPE

Specify a scope for a reference type column.

typed-table-name2

The name of a typed table. The data type of *column-name* must be REF(S), where S is the type of *typed-table-name2* (SQLSTATE 428DM). No checking is done of the default value for *column-name* to ensure that the value actually references an existing row in *typed-table-name2*.

typed-view-name2

The name of a typed view. The data type of *column-name* must be REF(S), where S is the type of *typed-view-name2* (SQLSTATE 428DM). No checking is done of the default value for *column-name* to ensure that the values actually references an existing row in *typed-view-name2*.

CONSTRAINT *constraint-name*

Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same ALTER TABLE statement, or as the name of any other existing constraint on the table (SQLSTATE 42710).

If the constraint name is not specified by the user, an 18 byte long identifier unique within the identifiers of the existing constraints defined on the table is generated by the system. (The identifier consists of "SQL" followed by a sequence of 15 numeric characters that are generated by a timestamp-based function.)

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* may be used as the name of an index that is created to support the constraint. See Notes for details on index names associated with unique constraints.

PRIMARY KEY

This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause were specified as a separate clause. The column cannot contain null values, so the NOT NULL attribute must also be specified (SQLSTATE 42831).

See PRIMARY KEY within the description of the *unique-constraint* below.

UNIQUE

This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause were specified as a separate clause.

See UNIQUE within the description of the *unique-constraint* below.

references-clause

This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* in "CREATE TABLE".

CHECK (*check-condition*)

This provides a shorthand method of defining a check constraint that applies to a single column. See *check-condition* in "CREATE TABLE".

generated-column-definition

For details on column generation, see "CREATE TABLE".

default-clause

Specifies a default value for the column.

WITH

An optional keyword.

DEFAULT

Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a specific default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in Table 4. If a column is defined as an XML or structured type, then a DEFAULT clause cannot be specified.

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

Table 4. Default Values (when no value specified)

Data Type	Default Value
Numeric	0
Fixed-length character string	Blanks
Varying-length character string	A string of length 0
Fixed-length graphic string	Double-byte blanks
Varying-length graphic string	A string of length 0
Date	For existing rows, a date corresponding to January 1, 0001. For added rows, the current date.
Time	For existing rows, a time corresponding to 0 hours, 0 minutes, and 0 seconds. For added rows, the current time.

Table 4. Default Values (when no value specified) (continued)

Data Type	Default Value
Timestamp	For existing rows, a date corresponding to January 1, 0001, and a time corresponding to 0 hours, 0 minutes, 0 seconds and 0 microseconds. For added rows, the current timestamp.
Binary string (blob)	A string of length 0

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column.

Specific types of values that can be specified with the DEFAULT keyword are as follows.

constant

Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment as described in Chapter 3
- not be a floating-point constant unless the column is defined with a floating-point data type
- be a numeric constant or a decimal floating-point special value if the data type of the column is decimal floating-point. Floating-point constants are first interpreted as DOUBLE and then converted to decimal floating-point. For DECFLOAT(16) columns, decimal constants must have a precision less than or equal to 16.
- not have nonzero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 bytes including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*.

datetime-special-register

Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified). For existing rows, the value is the current date, current time or current timestamp when the ALTER TABLE statement is processed.

user-special-register

Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and

CURRENT_USER can be specified in place of CURRENT_USER. For existing rows, the value is the CURRENT_USER, SESSION_USER, or SYSTEM_USER of the ALTER TABLE statement.

CURRENT SCHEMA

Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. For existing rows, the value of the CURRENT SCHEMA special register at the time the ALTER TABLE statement is processed.

NULL

Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL must not be specified within the same column definition.

cast-function

This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant

Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the *cast-function* is BLOB, the constant must be a string constant.

datetime-special-register

Specifies CURRENT_DATE, CURRENT_TIME, or CURRENT_TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register

Specifies CURRENT_USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA

Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

EMPTY_CLOB(), EMPTY_DBCLOB(), or EMPTY_BLOB()

Specifies a zero-length string as the default for the column. The column must have the data type that corresponds to the result data type of the function.

If the value specified is not valid, an error (SQLSTATE 42894) is returned.

GENERATED

Specifies that DB2 generates values for the column.

ALWAYS

Specifies that DB2 will always generate a value for the column when a row is inserted into the table, or whenever the result value of the *generation-expression* might change. The result of the expression is stored in the table. GENERATED ALWAYS is the recommended option unless data propagation or unload and reload operations are being performed. GENERATED ALWAYS is the required option for generated columns.

BY DEFAULT

Specifies that DB2 will generate a value for the column when a row is inserted into the table, or updated, specifying DEFAULT for the column, unless an explicit value is specified. BY DEFAULT is the recommended option when using data propagation or performing unload and reload operations.

AS (*generation-expression*)

Specifies that the definition of the column is based on an expression. Requires that the table be put in set integrity pending state, using the SET INTEGRITY statement with the OFF option. After the ALTER TABLE statement, the SET INTEGRITY statement with the IMMEDIATE CHECKED and FORCE GENERATED options must be used to update and check all the values in that column against the new expression. For details on specifying a column with a *generation-expression*, see "CREATE TABLE".

COMPRESS SYSTEM DEFAULT

Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the VALUE COMPRESSION clause is not specified, a warning is returned (SQLSTATE 01648) and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of extra checking that is done.

The base data type must not be a DATE, TIME, TIMESTAMP, XML, or structured data type (SQLSTATE 42842). If the base data type is a

varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

COLUMN SECURED WITH *security-label-name*

Identifies a security label that exists for the security policy that is associated with the table. The name must not be qualified (SQLSTATE 42601). The table must have a security policy associated with it (SQLSTATE 55064).

ADD *unique-constraint*

Defines a unique or primary key constraint. A primary key or unique constraint cannot be added to a table that is a subtable (SQLSTATE 429B3). If the table is a supertable at the top of the hierarchy, the constraint applies to the table and all its subtables.

CONSTRAINT *constraint-name*

Names the primary key or unique constraint. For more information, see *constraint-name* in "CREATE TABLE".

UNIQUE (*column-name...*)

Defines a unique key composed of the identified columns. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified more than once. The name cannot be qualified. The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" in "CREATE TABLE". For key length limits, see "SQL limits". No LOB, distinct type based on any of these types, or structured type can be used as part of a unique key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008). The set of columns in the unique key cannot be the same as the set of columns of the primary key or another unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.) Any existing values in the set of identified columns must be unique (SQLSTATE 23515).

A check is performed to determine whether an existing index matches the unique key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (ASC/DESC) specifications. However, for partitioned tables, non-unique partitioned indexes whose columns are not a superset of the table-partitioning key columns are not considered matching indexes.

If a matching index definition is found, the description of the index is changed to indicate that it is required by the system and it is changed to unique (after ensuring uniqueness) if it was a non-unique index. If the table has more than one matching index, an existing unique index is selected. If there are multiple unique indexes, the selection is arbitrary with one exception:

- For partitioned tables, matching unique partitioned indexes are favored over matching unique nonpartitioned indexes or matching non-unique indexes (partitioned or nonpartitioned).

If no matching index is found, a unique bidirectional index will automatically be created for the columns, as described in CREATE TABLE. See Notes for details on index names associated with unique constraints.

PRIMARY KEY ...(*column-name*,)

Defines a primary key composed of the identified columns. Each *column-name* must identify a column of the table, and the same column must not be identified more than once. The name cannot be qualified. The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see “Byte Counts” in “CREATE TABLE”. For key length limits, see “SQL limits”. The table must not have a primary key and the identified columns must be defined as NOT NULL. No LOB, distinct type based on any of these types, or structured type may be used as part of a primary key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008). The set of columns in the primary key cannot be the same as the set of columns in a unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.) Any existing values in the set of identified columns must be unique (SQLSTATE 23515).

A check is performed to determine if an existing index matches the primary key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (ASC/DESC) specifications. However, for partitioned tables, non-unique partitioned indexes whose columns are not a superset of the table-partitioning key columns are not considered matching indexes.

If a matching index definition is found, the description of the index is changed to indicate that it is the primary index, as required by the system, and it is changed to unique (after ensuring uniqueness) if it was a non-unique index. If the table has more than one matching index, an existing unique index is selected. If there are multiple unique indexes, the selection is arbitrary with one exception:

- For partitioned tables, matching unique partitioned indexes are favored over matching unique nonpartitioned indexes or matching non-unique indexes (partitioned or nonpartitioned).

If no matching index is found, a unique bidirectional index will automatically be created for the columns, as described in CREATE TABLE. See Notes for details on index names associated with unique constraints.

Only one primary key can be defined on a table.

ADD referential-constraint

Defines a referential constraint. See *referential-constraint* in “CREATE TABLE”.

ADD check-constraint

Defines a check constraint or functional dependency. See *check-constraint* in “CREATE TABLE”.

ADD distribution-clause

Defines a distribution key. The table must be defined in a table space on a single-partition database partition group (SQLSTATE 55037) and must not already have a distribution key (SQLSTATE 42889). If a distribution key already exists for the table, the existing key must be dropped before adding the new distribution key. A distribution key cannot be added to a table that is a subtable (SQLSTATE 428DH).

DISTRIBUTE BY HASH (*column-name*...)

Defines a distribution key using the specified columns. Each *column-name* must identify a column of the table, and the same column must not be

identified more than once. The name cannot be qualified. A column cannot be used as part of a distribution key if the data type of the column is a BLOB, CLOB, DBCLOB, XML, distinct type on any of these types, or structured type.

ADD RESTRICT ON DROP

Specifies that the table cannot be dropped, and that the table space that contains the table cannot be dropped.

ADD MATERIALIZED QUERY

materialized-query-definition

Changes a regular table to a materialized query table for use during query optimization. The table specified by *table-name* must not:

- Be previously defined as a materialized query table
- Be a typed table
- Have any constraints, unique indexes, or triggers defined
- Reference a nickname that is marked with caching disabled
- Be referenced in the definition of another materialized query table
- Be referenced in the definition of a view that is enabled for query optimization

If *table-name* does not meet these criteria, an error is returned (SQLSTATE 428EW).

fullselect

Defines the query in which the table is based. The columns of the existing table must:

- have the same number of columns
- have exactly the same data types
- have the same column names in the same ordinal positions

as the result columns of *fullselect* (SQLSTATE 428EW). For details about specifying the *fullselect* for a materialized query table, see “CREATE TABLE”. One additional restriction is that *table-name* cannot be directly or indirectly referenced in the fullselect.

refreshable-table-options

Specifies the refreshable options for altering a materialized query table.

DATA INITIALLY DEFERRED

The data in the table must be validated using the REFRESH TABLE or SET INTEGRITY statement.

REFRESH

Indicates how the data in the table is maintained.

DEFERRED

The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH TABLE statement is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

IMMEDIATE

The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the materialized query table. In this case, the content of the table, at any

point-in-time, is the same as if the specified subselect is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

ENABLE QUERY OPTIMIZATION

The materialized query table can be used for query optimization.

DISABLE QUERY OPTIMIZATION

The materialized query table will not be used for query optimization. The table can still be queried directly.

MAINTAINED BY

Specifies whether the data in the materialized query table is maintained by the system, user, or replication tool.

SYSTEM

Specifies that the data in the materialized query table is maintained by the system.

USER

Specifies that the data in the materialized query table is maintained by the user. The user is allowed to perform update, delete, or insert operations against user-maintained materialized query tables. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against user-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY USER.

FEDERATED_TOOL

Specifies that the data in the materialized query table is maintained by the replication tool. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against federated_tool-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY FEDERATED_TOOL.

ALTER FOREIGN KEY *constraint-name*

Alters the constraint attributes of the referential constraint *constraint-name*. The *constraint-name* must identify an existing referential constraint (SQLSTATE 42704).

ALTER CHECK *constraint-name*

Alters the constraint attributes of the check constraint or functional dependency *constraint-name*. The *constraint-name* must identify an existing check constraint or functional dependency (SQLSTATE 42704).

constraint-alteration

Options for changing attributes associated with referential or check constraints.

ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION

Specifies whether the constraint or functional dependency can be used for query optimization under appropriate circumstances.

ENABLE QUERY OPTIMIZATION

The constraint is assumed to be true and can be used for query optimization.

DISABLE QUERY OPTIMIZATION

The constraint cannot be used for query optimization.

ENFORCED or NOT ENFORCED

Specifies whether the constraint is enforced by the database manager during normal operations such as insert, update, or delete.

ENFORCED

Change the constraint to ENFORCED. ENFORCED cannot be specified for a functional dependency (SQLSTATE 42621).

NOT ENFORCED

Change the constraint to NOT ENFORCED. This should only be specified if the table data is independently known to conform to the constraint. Query results might be unpredictable if the data does not actually conform to the constraint.

ALTER *column-alteration*

Alters the definition of a column. Only the specified attributes will be altered; others will remain unchanged. Columns of a typed table cannot be altered (SQLSTATE 428DH).

column-name

Specifies the name of the column that is to be altered. The *column-name* must identify an existing column of the table (SQLSTATE 42703). The name must not be qualified. The name must not identify a column that is otherwise being added, altered, or dropped in the same ALTER TABLE statement (SQLSTATE 42711).

SET DATA TYPE *altered-data-type*

Specifies the new data type of the column. The new data type must be castable to the existing data type of the column (SQLSTATE 42837). Altering the length of a VARCHAR or VARCHARIC column which does not truncate any existing data does not require a subsequent reorganization of the table. If only trailing blanks are truncated, then a table reorganization is required before the table can be fully accessed (SQLSTATE 57016). The administrative routine SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS can be called to do table reorganization as required. Truncation of non-blank characters is not allowed (SQLSTATE 42837).

A string data type cannot be altered if the column is a column of a data partitioning key.

If the column is a column of a distribution key, then the new data type must not be REAL, DOUBLE, DECFLOAT(16), DECFLOAT(34), or DECIMAL(p, m) if ($p - m > 4$), except if the alter is from DECFLOAT(16) to DECFLOAT(34).

The specified length cannot be less than the existing length if the data type is a LOB (SQLSTATE 42837).

The data type of an identity column cannot be altered (SQLSTATE 42997).

The table cannot have data capture enabled (SQLSTATE 42997).

Altering a column must not make the total byte count of all columns exceed the maximum record size (SQLSTATE 54010). If the column is used in a unique constraint or an index, the new length must not cause the sum of the stored lengths for the unique constraint or index to exceed the index key length limit for the page size (SQLSTATE 54008). For column stored lengths, see "Byte Counts" in "CREATE TABLE". For key length limits, see "SQL limits".

If `auto_reval` is set to `DISABLED`, the cascaded effects of altering a column is shown in Table 5.

Table 5. Cascaded Effects of Altering a Column

Operation	Effect
Altering a column that is referenced by a view or check constraint	The object is regenerated during alter processing. In the case of a view, function or method resolution for the object might be different after the alter operation, changing the semantics of the object. In the case of a check constraint, if the semantics of the object will change as a result of the alter operation, the operation fails.
Altering a column in a table that has a dependent package, trigger, or SQL routine	The object is marked invalid, and is revalidated on next use.
Altering the type of a column in a table that is referenced by an XSROBJECT enabled for decomposition	The XSROBJECT is marked inoperative for decomposition. Re-enabling the XSROBJECT might require readjustment of its mappings; following this, issue an ALTER XSROBJECT ENABLE DECOMPOSITION statement against the XSROBJECT.
Altering a column that is referenced in the default expression of a global variable	The default expression of the global variable is validated during alter processing. If a user-defined function used in the default expression cannot be resolved, the operation fails.

SET *generated-column-alteration*

Specifies the technique used to generate a value for the column. This can be in the form of a specific default value, an expression, or defining the column as an identity column. If an existing default for the column results from a different generation technique, that default must be dropped, which can be done in the same *column-alteration* using one of the DROP clauses.

default-clause

Specifies a new default value for the column that is to be altered. The column must not already be defined as the identity column or have a generation expression defined (SQLSTATE 42837). The specified default value must represent a value that could be assigned to the column in accordance with the rules for assignment as described in “Assignments and comparisons”. Altering the default value does not change the value that is associated with this column for existing rows.

GENERATED ALWAYS or GENERATED BY DEFAULT

Specifies when the database manager is to generate values for the column. `GENERATED BY DEFAULT` specifies that a value is only to be generated when a value is not provided, or the `DEFAULT` keyword is used in an assignment to the column. `GENERATED ALWAYS` specifies that the database manager is to always generate a value for the column. `GENERATED BY DEFAULT` cannot be specified with a *generation-expression*.

identity-options

Specifies that the column is the identity column for the table. The column must not already be defined as the identity column, cannot have a generation expression, or cannot have an explicit default (SQLSTATE 42837). A table can only have a single identity column

(SQLSTATE 428C1). The column must be specified as not nullable (SQLSTATE 42997), and the data type associated with the column must be an exact numeric data type with a scale of zero (SQLSTATE 42815). An exact numeric data type is one of: SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC with a scale of zero, or a distinct type based on one of these types. For details on identity options, see “CREATE TABLE”.

AS (*generation-expression*)

Specifies that the definition of the column is based on an expression. The column must not already be defined with a generation expression, cannot be the identity column, or cannot have an explicit default (SQLSTATE 42837). The *generation-expression* must conform to the same rules that apply when defining a generated column. The result data type of the *generation-expression* must be assignable to the data type of the column (SQLSTATE 42821). The column must not be referenced in the distribution key column or in the ORGANIZE BY clause (SQLSTATE 42997).

SET EXPRESSION AS (*generation-expression*)

Changes the expression for the column to the specified *generation-expression*. SET EXPRESSION AS requires the table to be put in set integrity pending state, using the SET INTEGRITY statement with the OFF option. After the ALTER TABLE statement, the SET INTEGRITY statement with the IMMEDIATE CHECKED and FORCE GENERATED options must be used to update and check all the values in that column against the new expression. The column must already be defined as a generated column based on an expression (SQLSTATE 42837), and must not have appeared in the PARTITIONING KEY, DIMENSIONS, or KEY SEQUENCE clauses of the table (SQLSTATE 42997). The *generation-expression* must conform to the same rules that apply when defining a generated column. The result data type of the *generation-expression* must be assignable to the data type of the column (SQLSTATE 42821).

SET INLINE LENGTH *integer*

Changes the inline length of an existing structured type, XML, or LOB data type column. The inline length indicates the maximum size in bytes of an instance of a structured type, XML, or LOB data type to store in the base table row. Instances of a structured type or XML data type that cannot be stored inline in the base table row are stored separately, similar to the way that LOB values are stored.

The data type of *column-name* must be a structured type, XML, or LOB data type (SQLSTATE 42842).

The default inline length for a structured type column is the inline length of its data type (specified explicitly or by default in the CREATE TYPE statement). If the inline length of a structured type is less than 292, the value 292 is used for the inline length of the column.

The explicit inline length value can only be increased (SQLSTATE 429B2); it cannot exceed 32673 (SQLSTATE 54010). For a structured type or XML data type column, it must be at least 292. For a LOB data type column, the INLINE LENGTH must not be less than the maximum LOB descriptor size.

Altering the column must not make the total byte count of all columns exceed the row size limit (SQLSTATE 54010).

Data that is already stored separately from the rest of the row will not be moved inline into the base table row by this statement. To take advantage of the altered inline length of a structured type column, invoke the REORG command against the specified table after altering the inline length of its column. To take advantage of the altered inline length of an XML data type column in an existing table, update all rows with an UPDATE statement. The REORG command has no effect on the row storage of XML documents. To take advantage of the altered inline length of a LOB data type column, use the REORG command with the LONGLOBDATA option or UPDATE the corresponding LOB column. For example:

```
UPDATE table-name SET lob-column = lob-column
WHERE LENGTH(lob-column) <= chosen-inline-length - 4
```

where *table-name* is the table that had the inline length of the LOB data type column altered, *lob-column* is the LOB data type column that was altered, and *chosen-inline-length* is the new value that was chosen for the INLINE LENGTH.

SET NOT NULL

Specifies that the column cannot contain null values. No value for this column in existing rows of the table can be the null value (SQLSTATE 23502). This clause is not allowed if the column is specified in the foreign key of a referential constraint with a DELETE rule of SET NULL, and no other nullable columns exist in the foreign key (SQLSTATE 42831). Altering this attribute for a column requires table reorganization before further table access is allowed (SQLSTATE 57016). Note that because this operation requires validation of table data, it cannot be performed when the table is in reorg pending state (SQLSTATE 57016). The table cannot have data capture enabled (SQLSTATE 42997).

FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

Specifies that the column is a timestamp column for the table. A value is generated for the column in each row that is inserted, and for any row in which any column is updated. The value that is generated for a ROW CHANGE TIMESTAMP column is a timestamp that corresponds to the insert or update time for that row. If multiple rows are inserted or updated with a single statement, the value of the ROW CHANGE TIMESTAMP column might be different for each row.

A table can only have one ROW CHANGE TIMESTAMP column (SQLSTATE 428C1). If *data-type* is specified, it must be TIMESTAMP or TIMESTAMP(6) (SQLSTATE 42842). A ROW CHANGE TIMESTAMP column cannot have a DEFAULT clause (SQLSTATE 42623). NOT NULL must be specified for a ROW CHANGE TIMESTAMP column (SQLSTATE 42831).

SET GENERATED ALWAYS or GENERATED BY DEFAULT

Specifies when the database manager is to generate values for the column. GENERATED BY DEFAULT specifies that a value is only to be generated when a value is not provided or the DEFAULT keyword is used in an assignment to the column. GENERATED ALWAYS specifies that the database manager is to always generate a value for the column. The column must already be defined as a generated column based on an identity column; that is, defined with the AS IDENTITY clause (SQLSTATE 42837).

identity-alteration

Alters the identity attributes of the column. The column must be an identity column.

SET INCREMENT BY *numeric-constant*

Specifies the interval between consecutive values of the identity column. The next value to be generated for the identity column will be determined from the last assigned value with the increment applied. The column must already be defined with the IDENTITY attribute (SQLSTATE 42837).

This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), and does not exceed the value of a large integer constant (SQLSTATE 42820), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, this is a descending sequence after the ALTER statement. If this value is 0 or positive, this is an ascending sequence after the ALTER statement.

SET NO MINVALUE or MINVALUE *numeric-constant*

Specifies the minimum value at which a descending identity column either cycles or stops generating values, or the value to which an ascending identity column cycles after reaching the maximum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO MINVALUE

For an ascending sequence, the value is the original starting value. For a descending sequence, the value is the minimum value of the data type of the column.

MINVALUE *numeric-constant*

Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

SET NO MAXVALUE or MAXVALUE *numeric-constant*

Specifies the maximum value at which an ascending identity column either cycles or stops generating values, or the value to which a descending identity column cycles after reaching the minimum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO MAXVALUE

For an ascending sequence, the value is the maximum value of the data type of the column. For a descending sequence, the value is the original starting value.

MAXVALUE *numeric-constant*

Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing

to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

SET NO CYCLE or CYCLE

Specifies whether this identity column should continue to generate values after generating either its maximum or minimum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO CYCLE

Specifies that values will not be generated for the identity column once the maximum or minimum value has been reached.

CYCLE

Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, then after an ascending identity column reaches the maximum value, it generates its minimum value; or after a descending sequence reaches the minimum value, it generates its maximum value. The maximum and minimum values for the identity column determine the range that is used for cycling.

When CYCLE is in effect, duplicate values can be generated for an identity column. Although not required, if unique values are desired, a single-column unique index defined using the identity column will ensure uniqueness. If a unique index exists on such an identity column and a non-unique value is generated, an error occurs (SQLSTATE 23505).

SET NO CACHE or CACHE *integer-constant*

Specifies whether to keep some pre-allocated values in memory for faster access. This is a performance and tuning option. The column must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO CACHE

Specifies that values for the identity column are not to be pre-allocated. In a data sharing environment, if the identity values *must* be generated in order of request, the NO CACHE option must be used.

When this option is specified, the values of the identity column are not stored in the cache. In this case, every request for a new identity value results in synchronous I/O to the log.

CACHE *integer-constant*

Specifies how many values of the identity sequence are pre-allocated and kept in memory. When values are generated for the identity column, pre-allocating and storing values in the cache reduces synchronous I/O to the log.

If a new value is needed for the identity column and there are no unused values available in the cache, the allocation of the value requires waiting for I/O to the log. However, when a new value is needed for the identity column and there is an unused value in the cache, the allocation of that identity value can happen more quickly by avoiding the I/O to the log.

In the event of a database deactivation, either normally or due to a system failure, all cached sequence values that have not been used

in committed statements are lost (that is, they will never be used). The value specified for the CACHE option is the maximum number of values for the identity column that could be lost in case of system failure.

The minimum value is 2 (SQLSTATE 42815).

SET NO ORDER or ORDER

Specifies whether the identity column values must be generated in order of request. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO ORDER

Specifies that the identity column values do not need to be generated in order of request.

ORDER

Specifies that the identity column values must be generated in order of request.

RESTART or RESTART WITH *numeric-constant*

Resets the state of the sequence associated with the identity column. If WITH *numeric-constant* is not specified, the sequence for the identity column is restarted at the value that was specified, either implicitly or explicitly, as the starting value when the identity column was originally created.

The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837). RESTART does *not* change the original START WITH value.

The *numeric-constant* is an exact numeric constant that can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA). The *numeric-constant* will be used as the next value for the column.

DROP IDENTITY

Drops the identity attributes of the column, making the column a simple numeric data type column. DROP IDENTITY is not allowed if the column is not an identity column (SQLSTATE 42837).

DROP EXPRESSION

Drops the generated expression attributes of the column, making the column a non-generated column. DROP EXPRESSION is not allowed if the column is not a generated expression column (SQLSTATE 42837).

DROP DEFAULT

Drops the current default for the column. The specified column must have a default value (SQLSTATE 42837).

DROP NOT NULL

Drops the NOT NULL attribute of the column, allowing the column to have the null value. This clause is not allowed if the column is specified in the primary key, or in a unique constraint of the table (SQLSTATE 42831). Altering this attribute for a column requires table reorganization before further table access is allowed (SQLSTATE 57016). The table cannot have data capture enabled (SQLSTATE 42997).

ADD SCOPE

Add a scope to an existing reference type column that does not already

have a scope defined (SQLSTATE 428DK). If the table being altered is a typed table, the column must not be inherited from a supertable (SQLSTATE 428DJ).

typed-table-name

The name of a typed table. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name

The name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

COMPRESS

Specifies whether or not default values for this column are to be stored more efficiently.

SYSTEM DEFAULT

Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the table is not already set with the VALUE COMPRESSION attribute activated, a warning is returned (SQLSTATE 01648), and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of the extra checking that is done.

Existing data in the column is not changed. Consider offline table reorganization to enable existing data to take advantage of storing system default values using minimal space.

OFF

Specifies that system default values are to be stored in the column as regular values. Existing data in the column is not changed. Offline reorganization is recommended to change existing data.

The base data type must not be DATE, TIME or TIMESTAMP (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

If the table being altered is a typed table, the column must not be inherited from a supertable (SQLSTATE 428DJ).

SECURED WITH *security-label-name*

Identifies a security label that exists for the security policy that is associated with the table. The name must not be qualified (SQLSTATE 42601). The table must have a security policy associated with it (SQLSTATE 55064).

DROP COLUMN SECURITY

Alters a column to make it a non-protected column.

RENAME COLUMN *source-column-name* **TO** *target-column-name*

Renames the column that is specified in *source-column-name* to the name that is specified in *target-column-name*. If the **auto_reval** database configuration

parameter is set to DISABLED, the RENAME COLUMN option of the ALTER TABLE statement behaves like it is under the control of revalidation immediate semantics.

source-column-name

Specifies the name of the column that is to be renamed. The *source-column-name* must identify an existing column of the table (SQLSTATE 42703). The name must not be qualified. The name must not identify a column that is otherwise being added, altered, or dropped in the same ALTER TABLE statement (SQLSTATE 42711).

target-column-name

The new name for the column. The name must not be qualified. Existing column names in the table must not be used (SQLSTATE 42711).

DROP PRIMARY KEY

Drops the definition of the primary key and all referential constraints dependent on this primary key. The table must have a primary key (SQLSTATE 42888).

DROP FOREIGN KEY *constraint-name*

Drops the referential constraint *constraint-name*. The *constraint-name* must identify a referential constraint (SQLSTATE 42704). For information on implications of dropping a referential constraint see Notes.

DROP UNIQUE *constraint-name*

Drops the definition of the unique constraint *constraint-name* and all referential constraints dependent on this unique constraint. The *constraint-name* must identify an existing UNIQUE constraint (SQLSTATE 42704). For information on implications of dropping a unique constraint, see Notes.

DROP CHECK *constraint-name*

Drops the check constraint *constraint-name*. The *constraint-name* must identify an existing check constraint defined on the table (SQLSTATE 42704).

DROP CONSTRAINT *constraint-name*

Drops the constraint *constraint-name*. The *constraint-name* must identify an existing check constraint, referential constraint, primary key, or unique constraint defined on the table (SQLSTATE 42704). For information on implications of dropping a constraint, see Notes.

DROP COLUMN

Drops the identified column from the table. The table must not be a typed table (SQLSTATE 428DH). The table cannot have data capture enabled (SQLSTATE 42997). If a column is dropped, the table must be reorganized before an update, insert, or delete operation or an index scan can be performed on the table (SQLSTATE 57016). An XML column can only be dropped only if all of the other XML columns in the table are dropped at the same time.

column-name

Identifies the column that is to be dropped. The column name must not be qualified. The name must identify a column of the specified table (SQLSTATE 42703). The name must not identify the only column of the table (SQLSTATE 42814). The name must not identify a column that is part of the table's distribution key, table partitioning key, or organizing dimensions (SQLSTATE 42997).

CASCADE

Specifies that any views, indexes, triggers, SQL functions, constraints, or global variables that are dependent on the column being dropped are also dropped, or that any decomposition-enabled XSROBJECTs that are

dependent on the table containing the column are made inoperative for decomposition. A trigger is dependent on the column if it is referenced in the UPDATE OF column list, or anywhere in the triggered action. A decomposition-enabled XSROBJECT is dependent on a table if it contains a mapping of an XML element or attribute to the table. If an SQL function or global variable is dependent on another database object, it might not be possible to drop the function or global variable by means of the CASCADE option. CASCADE is the default.

RESTRICT

Specifies that the column cannot be dropped if any views, indexes, triggers, constraints, or global variables are dependent on the column, or if any decomposition-enabled XSROBJECT is dependent on the table that contains the column (SQLSTATE 42893). A trigger is dependent on the column if it is referenced in the UPDATE OF column list, or anywhere in the triggered action. A decomposition-enabled XSROBJECT is dependent on a table if it contains a mapping of an XML element or attribute to the table. The first dependent object that is detected is identified in the administration log.

Table 6. Cascaded Effects of Dropping a Column

Operation	RESTRICT Effect	CASCADE Effect
Dropping a column that is referenced by a view or a trigger	Dropping the column is not allowed.	The object and all objects that are dependent on that object are dropped.
Dropping a column that is referenced in the key of an index	If all columns that are referenced in the index are dropped in the same ALTER TABLE statement, dropping the index is allowed. Otherwise, dropping the column is not allowed.	The index is dropped.
Dropping a column that is referenced in a unique constraint	If all columns that are referenced in the unique constraint are dropped in the same ALTER TABLE statement, and the unique constraint is not referenced by a referential constraint, the columns and the constraint are dropped. (The index that is used to satisfy the constraint is also dropped.) Otherwise, dropping the column is not allowed.	The unique constraint and any referential constraints that reference that unique constraint are dropped. (Any indexes that are used by those constraints are also dropped).
Dropping a column that is referenced in a referential constraint	If all columns that are referenced in the referential constraint are dropped in the same ALTER TABLE statement, the columns and the constraint are dropped. Otherwise, dropping the column is not allowed.	The referential constraint is dropped.
Dropping a column that is referenced by a system-generated column that is not being dropped.	Dropping the column is not allowed.	Dropping the column is not allowed.

Table 6. Cascaded Effects of Dropping a Column (continued)

Operation	RESTRICT Effect	CASCADE Effect
Dropping a column that is referenced in a check constraint	Dropping the column is not allowed.	The check constraint is dropped.
Dropping a column that is referenced in a decomposition-enabled XSROBJECT	Dropping the column is not allowed.	The XSROBJECT is marked inoperative for decomposition. Re-enabling the XSROBJECT might require readjustment of its mappings; following this, issue an ALTER XSROBJECT ENABLE DECOMPOSITION statement against the XSROBJECT.
Dropping a column that is referenced in the default expression of a global variable	Dropping the column is not allowed.	The global variable is dropped, unless the dropping of the global variable is disallowed because there are other objects, which do not allow the cascade, that depend on the global variable.

DROP RESTRICT ON DROP

Removes the restriction, if there is one, on dropping the table and the table space that contains the table.

DROP DISTRIBUTION

Drops the distribution definition for the table. The table must have a distribution definition (SQLSTATE 428FT). The table space for the table must be defined on a single partition database partition group.

DROP MATERIALIZED QUERY

Changes a materialized query table so that it is no longer considered to be a materialized query table. The table specified by *table-name* must be defined as a materialized query table that is not replicated (SQLSTATE 428EW). The definition of the columns of *table-name* is not changed, but the table can no longer be used for query optimization, and the REFRESH TABLE statement can no longer be used.

DATA CAPTURE

Indicates whether extra information for data replication is to be written to the log.

If the table is a typed table, then this option is not supported (SQLSTATE 428DH for root tables or 428DR for other subtables).

NONE

Indicates that no extra information will be logged.

CHANGES

Indicates that extra information regarding SQL changes to this table will be written to the log. This option is required if this table will be replicated and the Capture program is used to capture changes for this table from the log.

If the table is defined to allow data on a database partition other than the catalog partition (multiple partition database partition group, or database

partition group with database partitions other than the catalog partition), then this option is not supported (SQLSTATE 42997).

If the schema name (implicit or explicit) of the table is longer than 18 bytes, this option is not supported (SQLSTATE 42997).

INCLUDE LONGVAR COLUMNS

Allows data replication utilities to capture changes made to LONG VARCHAR or LONG VARGRAPHIC columns. The clause may be specified for tables that do not have any LONG VARCHAR or LONG VARGRAPHIC columns since it is possible to ALTER the table to include such columns.

ACTIVATE NOT LOGGED INITIALLY

Activates the NOT LOGGED INITIALLY attribute of the table for this current unit of work.

Any changes made to the table by an INSERT, DELETE, UPDATE, CREATE INDEX, DROP INDEX, or ALTER TABLE in the same unit of work after the table is altered by this statement are not logged. Any changes made to the system catalog by the ALTER statement in which the NOT LOGGED INITIALLY attribute is activated are logged. Any subsequent changes made in the same unit of work to the system catalog information are logged.

At the completion of the current unit of work, the NOT LOGGED INITIALLY attribute is deactivated and all operations that are done on the table in subsequent units of work are logged.

If using this feature to avoid locks on the catalog tables while inserting data, it is important that only this clause be specified on the ALTER TABLE statement. Use of any other clause in the ALTER TABLE statement will result in catalog locks. If no other clauses are specified for the ALTER TABLE statement, then only a SHARE lock will be acquired on the system catalog tables. This can greatly reduce the possibility of concurrency conflicts for the duration of time between when this statement is executed and when the unit of work in which it was executed is ended.

If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

For more information about the NOT LOGGED INITIALLY attribute, see the description of this attribute in "CREATE TABLE".

Note: If non-logged activity occurs against a table that has the NOT LOGGED INITIALLY attribute activated, and if a statement fails (causing a rollback), or a ROLLBACK TO SAVEPOINT is executed, the entire unit of work is rolled back (SQL1476N). Furthermore, the table for which the NOT LOGGED INITIALLY attribute was activated is marked inaccessible after the rollback has occurred and can only be dropped. Therefore, the opportunity for errors within the unit of work in which the NOT LOGGED INITIALLY attribute is activated should be minimized.

WITH EMPTY TABLE

Causes all data currently in table to be removed. Once the data has been removed, it cannot be recovered except through use of the RESTORE facility. If the unit of work in which this alter statement was issued is rolled back, the table data will not be returned to its original state.

When this action is requested, no DELETE triggers defined on the affected table are fired. Any indexes that exist on the table are also deleted.

A partitioned table with attached data partitions cannot be emptied (SQLSTATE 42928).

PCTFREE *integer*

Specifies the percentage of each page that is to be left as free space during a load or a table reorganization operation. The first row on each page is added without restriction. When additional rows are added to a page, at least *integer* percent of the page is left as free space. The PCTFREE value is considered only by the load and table reorg utilities. The value of *integer* can range from 0 to 99. A PCTFREE value of -1 in the system catalog (SYSCAT.TABLES) is interpreted as the default value. The default PCTFREE value for a table page is 0. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

LOCKSIZE

Indicates the size (granularity) of locks used when the table is accessed. Use of this option in the table definition will not prevent normal lock escalation from occurring. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ROW

Indicates the use of row locks. This is the default lock size when a table is created.

BLOCKINSERT

Indicates the use of block locks during insert operations. This means that the appropriate exclusive lock is acquired on the block before insertion, and row locking is not done on the inserted row. This option is useful when separate transactions are inserting into separate cells in the table. Transactions inserting into the same cells can still do so concurrently, but will insert into distinct blocks, and this can impact the size of the cell if more blocks are needed. This option is only valid for MDC tables (SQLSTATE 42613).

TABLE

Indicates the use of table locks. This means that the appropriate share or exclusive lock is acquired on the table, and that intent locks (except intent none) are not used. For partitioned tables, this lock strategy is applied to both the table lock and the data partition locks for any data partitions that are accessed. Use of this value can improve the performance of queries by limiting the number of locks that need to be acquired. However, concurrency is also reduced, because all locks are held over the complete table.

APPEND

Indicates whether data is appended to the end of the table data or placed where free space is available in data pages. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ON

Indicates that table data will be appended and information about free space on pages will not be kept. The table must not have a clustered index (SQLSTATE 428CA).

OFF

Indicates that table data will be placed where there is available space. This is the default when a table is created.

The table should be reorganized after setting APPEND OFF since the information about available free space is not accurate and may result in poor performance during insert.

VOLATILE CARDINALITY or NOT VOLATILE CARDINALITY

Indicates to the optimizer whether or not the cardinality of table *table-name* can vary significantly at run time. Volatility applies to the number of rows in the table, not to the table itself. CARDINALITY is an optional keyword. The default is NOT VOLATILE.

VOLATILE

Specifies that the cardinality of table *table-name* can vary significantly at run time, from empty to large. To access the table, the optimizer will use an index scan (rather than a table scan, regardless of the statistics) if that index is index-only (all referenced columns are in the index), or that index is able to apply a predicate in the index scan. The list prefetch access method will not be used to access the table. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

NOT VOLATILE

Specifies that the cardinality of *table-name* is not volatile. Access plans to this table will continue to be based on existing statistics and on the current optimization level.

COMPRESS

Specifies whether or not data compression applies to the rows of the table.

YES

Specifies that data row compression is enabled. Insert and update operations on the table will be subject to compression. If no compression dictionary for the table exists, a compression dictionary is automatically created and rows are subject to compression after the table is sufficiently populated with data. If there is an existing compression dictionary for the table, compression is reactivated to use this dictionary, and rows are subject to compression. This also applies to the data in the XML storage object. If there is sufficient data in the XML storage object, a compression dictionary is automatically created and XML documents are subject to compression. Index compression will be enabled for new indexes unless explicitly disabled in the CREATE INDEX statement. Existing indexes can be compressed by using the ALTER INDEX statement.

NO

Specifies that data row compression is disabled. Insert and update operations on the table will no longer be subject to compression. Any rows in the table that are in compressed format remain in compressed format until they are converted to non-compressed format when they are updated. A non-inplace reorganization of the table decompresses all rows that are compressed. If a compression dictionary exists, it is discarded during table reinitialization or truncation (such as, for example, a replace operation). Index compression will be disabled for new indexes unless explicitly enabled in the CREATE INDEX statement. Index compression for existing indexes can be explicitly disabled by using the ALTER INDEX statement.

VALUE COMPRESSION

This determines the row format that is to be used. Each data type has a different byte count depending on the row format that is used. For more information, see "Byte Counts" in "CREATE TABLE". An update operation causes an existing row to be changed to the new row format. Offline table

reorganization is recommended to improve the performance of update operations on existing rows. This can also result in the table taking up less space. If the row size, calculated using the appropriate column in the table named "Byte Counts of Columns by Data Type" (see "CREATE TABLE"), would no longer fit within the row size limit, as indicated in the table named "Limits for Number of Columns and Row Size In Each Table Space Page Size", an error is returned (SQLSTATE 54010). If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ACTIVATE

The NULL value is stored using three bytes. This is the same or less space than when VALUE COMPRESSION is not active for columns of all data types, with the exception of CHAR(1). Whether or not a column is defined as nullable has no effect on the row size calculation. The zero-length data values for columns whose data type is VARCHAR, VARGRAPHIC, LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, or BLOB are to be stored using two bytes only, which is less than the storage required when VALUE COMPRESSION is not active. When a column is defined using the COMPRESS SYSTEM DEFAULT option, this also allows the system default value for the column to be stored using three bytes of total storage. The row format that is used to support this determines the byte counts for each data type, and tends to cause data fragmentation when updating to or from NULL, a zero-length value, or the system default value.

DEACTIVATE

The NULL value is stored with space set aside for possible future updates. This space is not set aside for varying-length columns. It also does not support efficient storage of system default values for a column. If columns already exist with the COMPRESS SYSTEM DEFAULT attribute, a warning is returned (SQLSTATE 01648).

LOG INDEX BUILD

Specifies the level of logging that is to be performed during create, recreate, or reorganize index operations on this table.

NULL

Specifies that the value of the **logindexbuild** database configuration parameter will be used to determine whether or not index build operations are to be completely logged. This is the default when the table is created.

OFF

Specifies that any index build operations on this table will be logged minimally. This value overrides the setting of the **logindexbuild** database configuration parameter.

ON

Specifies that any index build operations on this table will be logged completely. This value overrides the setting of the **logindexbuild** database configuration parameter.

Rules

- Any unique or primary key constraint defined on the table must be a superset of the distribution key, if there is one (SQLSTATE 42997).
- Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE).
- A column can only be referenced in one ADD, ALTER, or DROP COLUMN clause in a single ALTER TABLE statement (SQLSTATE 42711).

- A column length or data type cannot be altered, nor can the column be dropped, if the table has any materialized query tables that are dependent on the table (SQLSTATE 42997).
- VARCHAR and VARGRAPHIC columns that have been altered to be greater than 4000 and 2000, respectively, must not be used as input parameters in functions in the SYSFUN schema (SQLSTATE 22001).
- A column length cannot be altered if the table has any views enabled for query optimization that are dependent on the table (SQLSTATE 42997).
- The table must be put in set integrity pending state, using the SET INTEGRITY statement with the OFF option (SQLSTATE 55019), before:
 - Adding a column with a generation expression
 - Altering the generated expression of a column
 - Changing a column to have a generated expression
- An existing column cannot be altered to become of type DB2SECURITYLABEL (SQLSTATE 42837).
- Defining a column of type DB2SECURITYLABEL fails if the table does not have a security policy associated with it (SQLSTATE 55064).
- A column of type DB2SECURITYLABEL cannot be altered or dropped (SQLSTATE 42817).
- An ALTER TABLE operation to mark a table as protected fails if there exists an MQT that depends on that table (SQLSTATE 55067).
- Attaching a partition to a protected partitioned table fails if the source table and the target table are not protected using the same security policy, have the same row security label column, and have the same set of protected columns (SQLSTATE 428GE).
- If a generated column is referenced in a table partitioning key, the generated column expression cannot be altered (SQLSTATE 42837).
- The *isolation-clause* cannot be specified in the *fullselect* of the *materialized-query-definition* (SQLSTATE 42601).

Notes

- A *REORG-recommended operation* has occurred when changes resulting from an ALTER TABLE statement affect the row format of the data. When this occurs, most subsequent operations on the table are restricted until a table reorganization operation completes successfully. Up to three ALTER TABLE statements of this type can execute against a table before reorganization must be done (SQLSTATE 57016). Multiple alterations that would constitute a REORG-recommended operation can be made as part of a single ALTER TABLE statement (one per column); this is considered to be a single REORG-recommended operation. For example, dropping two columns in a single ALTER TABLE statement is not considered to be two REORG-recommended operations. Dropping two columns in two separate ALTER TABLE statements, however, would be regarded as two statements that contain REORG-recommended operations.
- The following table operations are allowed after a successful REORG-recommended operation has occurred:
 - ALTER TABLE, where no row data validation is required. However, the following operations are not allowed (SQLSTATE 57007):
 - ADD CHECK CONSTRAINT
 - ADD REFERENTIAL CONSTRAINT
 - ADD UNIQUE CONSTRAINT

- ALTER COLUMN SET NOT NULL
 - DROP TABLE
 - RENAME TABLE
 - REORG TABLE
 - TRUNCATE TABLE
 - Table scan access of table data
- Altering a table to make it a materialized query table will put the table in set integrity pending state. If the table is defined as REFRESH IMMEDIATE, the table must be taken out of set integrity pending state before INSERT, DELETE, or UPDATE commands can be invoked on the table referenced by the fullselect. The table can be taken out of set integrity pending state by using REFRESH TABLE or SET INTEGRITY, with the IMMEDIATE CHECKED option, to completely refresh the data in the table based on the fullselect. If the data in the table accurately reflects the result of the fullselect, the IMMEDIATE UNCHECKED option of SET INTEGRITY can be used to take the table out of set integrity pending state.
 - Altering a table to change it to a REFRESH IMMEDIATE materialized query table will cause any packages with INSERT, DELETE, or UPDATE usage on the table referenced by the fullselect to be invalidated.
 - Altering a table to change from a materialized query table to a regular table will cause any packages dependent on the table to be invalidated.
 - Altering a table to change from a MAINTAINED BY FEDERATED_TOOL materialized query table to a regular table will not cause any change in the subscription setup of the replication tool. Because a subsequent change to a MAINTAINED BY SYSTEM materialized query table will cause the replication tool to fail, you must change the subscription setting when changing a MAINTAINED BY FEDERATED_TOOL materialized query table.
 - If a deferred materialized query table is associated with a staging table, the staging table will be dropped if the materialized query table is altered to a regular table.
 - ADD column clauses are processed prior to all other clauses. Other clauses are processed in the order that they are specified.
 - Any columns added through an alter table operation will not automatically be added to any existing view of the table.
 - Adding or attaching a data partition to a partitioned table, or detaching a data partition from a partitioned table causes any packages that are dependent on that table to be invalidated.
 - To drop the partitioning for a table, the table must be dropped and then recreated.
 - To drop the organization for a table, the table must be dropped and then recreated.
 - When an index is automatically created for a unique or primary key constraint, the database manager will try to use the specified constraint name as the index name with a schema name that matches the schema name of the table. If this matches an existing index name or no name for the constraint was specified, the index is created in the SYSIBM schema with a system-generated name formed of "SQL" followed by a sequence of 15 numeric characters generated by a timestamp based function.
 - When a nonpartitioned index is created on a partitioned table with attached data partitions, the index will not include the data in the attached data partitions. Use the SET INTEGRITY statement to maintain all indexes for all attached data partitions.

- When creating a partitioned index in the presence of attached partitions (STATUS of 'A' in SYSDATAPARTITIONS), an index partition for each attached partition will also be created. If the partitioned index is being created as unique, or is an XML index being created with REJECT INVALID VALUES, then the index creation can fail if the attached partition contains any violations (duplicates for a unique index, or invalid values for the XML index).
- Any table that may be involved in a DELETE operation on table T is said to be *delete-connected* to T. Thus, a table is delete-connected to T if it is a dependent of T or it is a dependent of a table in which deletes from T cascade.
- A package has an insert (update/delete) usage on table T if records are inserted into (updated in/deleted from) T either directly by a statement in the package, or indirectly through constraints or triggers executed by the package on behalf of one of its statements. Similarly, a package has an update usage on a column if the column is modified directly by a statement in the package, or indirectly through constraints or triggers executed by the package on behalf of one of its statements.
- In a federated system, a remote base table that was created using transparent DDL can be altered. However, transparent DDL does impose some limitations on the modifications that can be made:
 - A remote base table can only be altered by adding new columns or specifying a primary key.
 - Specific clauses supported by transparent DDL include:
 - ADD COLUMN *column-definition*
 - NOT NULL and PRIMARY KEY in the *column-options* clause
 - ADD *unique-constraint* (PRIMARY KEY only)
 - You cannot specify a comment on an existing column in a remote base table.
 - An existing primary key in a remote base table cannot be altered or dropped.
 - Altering a remote base table invalidates any packages that are dependent on the nickname associated with that remote base table.
 - The remote data source must support the changes being requested through the ALTER TABLE statement. Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.
 - An attempt to alter a remote base table that was not created using transparent DDL returns an error.
- Any changes, whether implicit or explicit, to primary key, unique keys, or foreign keys might have the following effects on packages, indexes, and other foreign keys.
 - If a primary key or unique key is added:
 - There is no effect on packages, foreign keys, or existing unique keys. (If the primary or unique key uses an existing unique index that was created in a previous version and has not been converted to support deferred uniqueness, the index is converted, and packages with update usage on the associated table are invalidated.)
 - If a primary key or unique key is dropped:
 - The index is dropped if it was automatically created for the constraint. Any packages dependent on the index are invalidated.
 - The index is set back to non-unique if it was converted to unique for the constraint and it is no longer system-required. Any packages dependent on the index are invalidated.

- The index is set to no longer system required if it was an existing unique index used for the constraint. There is no effect on packages.
- All dependent foreign keys are dropped. Further action is taken for each dependent foreign key, as specified in the next item.
- If a foreign key is added, dropped, or altered from NOT ENFORCED to ENFORCED (or ENFORCED to NOT ENFORCED):
 - All packages with an insert usage on the object table are invalidated.
 - All packages with an update usage on at least one column in the foreign key are invalidated.
 - All packages with a delete usage on the parent table are invalidated.
 - All packages with an update usage on at least one column in the parent key are invalidated.
- If a foreign key or a functional dependency is altered from ENABLE QUERY OPTIMIZATION to DISABLE QUERY OPTIMIZATION:
 - All packages with dependencies on the constraint for optimization purposes are invalidated.
- Adding a column to a table will result in invalidation of all packages with insert usage on the altered table. If the added column is the first user-defined structured type column in the table, packages with DELETE usage on the altered table will also be invalidated.
- Adding a check or referential constraint to a table that already exists and that is not in set integrity pending state, or altering the existing check or referential constraint from NOT ENFORCED to ENFORCED on an existing table that is not in set integrity pending state will cause the existing rows in the table to be immediately evaluated against the constraint. If the verification fails, an error is returned (SQLSTATE 23512). If a table is in set integrity pending state, adding a check or referential constraint, or altering a constraint from NOT ENFORCED to ENFORCED will not immediately lead to the enforcement of the constraint. Issue the SET INTEGRITY statement with the IMMEDIATE CHECKED option to begin enforcing the constraint.
- Adding, altering, or dropping a check constraint will result in invalidation of all packages with either an insert usage on the object table, an update usage on at least one of the columns involved in the constraint, or a select usage exploiting the constraint to improve performance.
- Adding a distribution key invalidates all packages with an update usage on at least one of the columns of the distribution key.
- A distribution key that was defined by default as the first column of the primary key is not affected by dropping the primary key and adding a different primary key.
- Dropping a column or changing its data type removes all runstats information from the table being altered. Runstats should be performed on the table after it is again accessible. The statistical profile of the table is preserved if the table does not contain a column that was explicitly dropped.
- Altering a column (to increase its length or change its data type or nullability attribute) or dropping a column invalidates all packages that reference (directly or indirectly through a referential constraint or trigger) its table.
- Altering a column (to increase its length or change its data type or nullability attribute) regenerates views (except typed views) that are dependent on its table. If a problem occurs while regenerating such a view, an error is returned (SQLSTATE 56098). Any typed views that are dependent on the table are marked inoperative.

- Altering a column to increase its length or change its data type marks all dependent triggers and SQL functions as invalid; they are implicitly recompiled on next use. If a problem occurs while regenerating such an object, an error is returned (SQLSTATE 56098).
- Altering a column (to increase its length or change its data type or nullability attribute) might cause errors (SQLSTATE 54010) while processing a trigger or an SQL function when a statement involving the trigger or SQL function is prepared or bound. This can occur if the row length based on the sum of the lengths of the transition variables and transition table columns is too long. If such a trigger or SQL function is dropped, a subsequent attempt to recreate it returns an error (SQLSTATE 54040).
- Altering a structured or XML type column to increase the inline length will invalidate all packages that reference the table, either directly or indirectly through a referential constraint or trigger.
- Altering a structured or XML type column to increase the inline length will regenerate views that are dependent on the table.
- A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated the using an online table move.
- Changing the LOCKSIZE for a table will result in invalidation of all packages that have a dependency on the altered table.
- Changing VOLATILE or NOT VOLATILE CARDINALITY will result in invalidation of all packages that have a dependency on the altered table.
- **Replication:** Exercise caution when increasing the length or changing the data type of a column. The change data table that is associated with an application table might already be at or near the DB2 row size limit. The change data table should be altered before the application table, or the two tables should be altered within the same unit of work, to ensure that the alteration can be completed for both tables. Consideration should be given to copies, which might also be at or near the row size limit, or reside on platforms which lack the ability to increase the length of an existing column.

If the change data table is not altered before the Capture program processes log records with the altered attributes, the Capture program will likely fail. If a copy containing the altered column is not altered before the subscription maintaining the copy runs, the subscription will likely fail.

- When detaching a partition from a protected table, the target table automatically created by DB2 will be protected in exactly the same way the source table is protected.
- When a table is altered such that it becomes protected with row level granularity, any cached dynamic SQL sections that depend on such a table are invalidated. Similarly, any packages that depend on such a table are also invalidated.
- When a column of a table, T, is altered such that it becomes a protected column, any cached dynamic SQL sections that depend on table T are invalidated. Similarly, any packages that depend on table T are also invalidated.
- When a column of a table, T, is altered such that it becomes a non protected column, any cached dynamic SQL sections that depend on table T are invalidated. Similarly, any packages that depend on table T are also invalidated.
- For existing rows in the table, the value of the security label column defaults to the security label for write access of the session authorization ID at the time the ALTER statement that adds a row security label column is executed.

- **Considerations for implicitly hidden columns:** A column that is defined as implicitly hidden can be explicitly referenced in an ALTER TABLE statement. For example, an implicitly hidden column can be altered or specified as part of a referential constraint, check constraint, or materialized query table definition.
- **Compatibilities:** For compatibility with previous versions of DB2 products:
 - The ADD keyword is optional for:
 - Unnamed PRIMARY KEY constraints
 - Unnamed referential constraints
 - Referential constraints whose name follows the phrase FOREIGN KEY
 - The CONSTRAINT keyword can be omitted from a *column-definition* defining a references-clause
 - *constraint-name* can be specified following FOREIGN KEY (without the CONSTRAINT keyword)
 - SET SUMMARY AS can be specified in place of SET MATERIALIZED QUERY AS
 - SET MATERIALIZED QUERY AS DEFINITION ONLY can be specified in place of DROP MATERIALIZED QUERY
 - SET MATERIALIZED QUERY AS (fullselect) can be specified in place of ADD MATERIALIZED QUERY (fullselect)
 - ADD PARTITIONING KEY can be specified in place of ADD DISTRIBUTE BY HASH; the optional USING HASHING clause can also still be specified in this case
 - DROP PARTITIONING KEY can be specified in place of DROP DISTRIBUTION
 - The LONG VARCHAR and LONG VARGRAPHIC data types continue to be supported but are deprecated and not recommended, especially for portable applications

For compatibility with previous versions of DB2 products and for consistency:

- A comma can be used to separate multiple options in the *identity-alteration* clause

For compatibility with DB2 for z/OS:

- PART can be specified in place of PARTITION
- VALUES can be specified in place of ENDING AT

The following syntax is also supported:

- NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER

Examples

Example 1: Add a new column named RATING, which is one character long, to the DEPARTMENT table.

```
ALTER TABLE DEPARTMENT
  ADD RATING CHAR(1)
```

Example 2: Add a new column named SITE_NOTES to the PROJECT table. Create SITE_NOTES as a varying-length column with a maximum length of 1000 bytes. The values of the column do not have an associated character set and therefore should not be converted.

```
ALTER TABLE PROJECT
  ADD SITE_NOTES VARCHAR(1000) FOR BIT DATA
```

Example 3: Assume a table called EQUIPMENT exists defined with the following columns:

Column Name	Data Type
EQUIP_NO	INT
EQUIP_DESC	VARCHAR(50)
LOCATION	VARCHAR(50)
EQUIP_OWNER	CHAR(3)

Add a referential constraint to the EQUIPMENT table so that the owner (EQUIP_OWNER) must be a department number (DEPTNO) that is present in the DEPARTMENT table. DEPTNO is the primary key of the DEPARTMENT table. If a department is removed from the DEPARTMENT table, the owner (EQUIP_OWNER) values for all equipment owned by that department should become unassigned (or set to null). Give the constraint the name DEPTQUIP.

```
ALTER TABLE EQUIPMENT
ADD CONSTRAINT DEPTQUIP
FOREIGN KEY (EQUIP_OWNER)
REFERENCES DEPARTMENT
ON DELETE SET NULL
```

Also, an additional column is needed to allow the recording of the quantity associated with this equipment record. Unless otherwise specified, the EQUIP_QTY column should have a value of 1 and must never be null.

```
ALTER TABLE EQUIPMENT
ADD COLUMN EQUIP_QTY
SMALLINT NOT NULL DEFAULT 1
```

Example 4: Alter table EMPLOYEE. Add the check constraint named REVENUE defined so that each employee must make a total of salary and commission greater than \$30,000.

```
ALTER TABLE EMPLOYEE
ADD CONSTRAINT REVENUE
CHECK (SALARY + COMM > 30000)
```

Example 5: Alter table EMPLOYEE. Drop the constraint REVENUE which was previously defined.

```
ALTER TABLE EMPLOYEE
DROP CONSTRAINT REVENUE
```

Example 6: Alter a table to log SQL changes in the default format.

```
ALTER TABLE SALARY1
DATA CAPTURE NONE
```

Example 7: Alter a table to log SQL changes in an expanded format.

```
ALTER TABLE SALARY2
DATA CAPTURE CHANGES
```

Example 8: Alter the EMPLOYEE table to add 4 new columns with default values.

```
ALTER TABLE EMPLOYEE
ADD COLUMN HEIGHT MEASURE DEFAULT MEASURE(1)
ADD COLUMN BIRTHDAY BIRTHDATE DEFAULT DATE('01-01-1850')
ADD COLUMN FLAGS BLOB(1M) DEFAULT BLOB(X'01')
ADD COLUMN PHOTO PICTURE DEFAULT BLOB(X'00')
```

The default values use various function names when specifying the default. Since MEASURE is a distinct type based on INTEGER, the MEASURE function is used. The HEIGHT column default could have been specified without the function since the source type of MEASURE is not BLOB or a datetime data type. Since

BIRTHDATE is a distinct type based on DATE, the DATE function is used (BIRTHDATE cannot be used here). For the FLAGS and PHOTO columns the default is specified using the BLOB function even though PHOTO is a distinct type. To specify a default for BIRTHDAY, FLAGS and PHOTO columns, a function must be used because the type is a BLOB or a distinct type sourced on a BLOB or datetime data type.

Example 9: A table called CUSTOMERS is defined with the following columns:

Column Name	Data Type
BRANCH_NO	SMALLINT
CUSTOMER_NO	DECIMAL(7)
CUSTOMER_NAME	VARCHAR(50)

In this table, the primary key is made up of the BRANCH_NO and CUSTOMER_NO columns. To distribute the table, you will need to create a distribution key for the table. The table must be defined in a table space on a single-node database partition group. The primary key must be a superset of the distribution key columns: at least one of the columns of the primary key must be used as the distribution key. Make BRANCH_NO the distribution key as follows:

```
ALTER TABLE CUSTOMERS
ADD DISTRIBUTE BY HASH (BRANCH_NO)
```

Example 10: A remote table EMPLOYEE was created in a federated system using transparent DDL. Alter the remote table EMPLOYEE to add the columns PHONE_NO and WORK_DEPT; also add a primary key on the existing column EMP_NO and the new column WORK_DEPT.

```
ALTER TABLE EMPLOYEE
ADD COLUMN PHONE_NO CHAR(4) NOT NULL
ADD COLUMN WORK_DEPT CHAR(3)
ADD PRIMARY KEY (EMP_NO, WORK_DEPT)
```

Example 11: Alter the DEPARTMENT table to add a functional dependency FD1, then drop the functional dependency FD1 from the DEPARTMENT table.

```
ALTER TABLE DEPARTMENT
ADD CONSTRAINT FD1
CHECK ( DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED

ALTER TABLE DEPARTMENT
DROP CHECK FD1
```

Example 12: Change the default value for the WORKDEPT column in the EMPLOYEE table to 123.

```
ALTER TABLE EMPLOYEE
ALTER COLUMN WORKDEPT
SET DEFAULT '123'
```

Example 13: Associate the security policy DATA_ACCESS with the table EMPLOYEE.

```
ALTER TABLE EMPLOYEE
ADD SECURITY POLICY DATA_ACCESS
```

Example 14: Alter the table EMPLOYEE to protect the SALARY column.

```
ALTER TABLE EMPLOYEE
ALTER COLUMN SALARY
SECURED WITH EMPLOYEESECLABEL
```

Example 15: Assume that you have a table named SALARY_DATA that is defined with the following columns:

Column Name	Data Type
EMP_NAME	VARCHAR(50) NOT NULL
EMP_ID	SMALLINT NOT NULL
EMP_POSITION	VARCHAR(100) NOT NULL
SALARY	DECIMAL(5,2)
PROMOTION_DATE	DATE NOT NULL

Change this table to allow salaries to be stored in a DECIMAL(6,2) column, make PROMOTION_DATE an optional field that can be set to the null value, and remove the EMP_POSITION column.

```
ALTER TABLE SALARY_DATA
ALTER COLUMN SALARY SET DATA TYPE DECIMAL(6,2)
ALTER COLUMN PROMOTION_DATE DROP NOT NULL
DROP COLUMN EMP_POSITION
```

ALTER TABLESPACE

The ALTER TABLESPACE statement is used to modify an existing table space in the following ways:

- Add a container to, or drop a container from a DMS table space; that is, a table space created with the MANAGED BY DATABASE option.
- Modify the size of a container in a DMS table space.
- Lower the high water mark for a DMS table space through extent movement.
- Add a container to an SMS table space on a database partition that currently has no containers.
- Modify the PREFETCHSIZE setting for a table space.
- Modify the BUFFERPOOL used for tables in the table space.
- Modify the OVERHEAD setting for a table space.
- Modify the TRANSFERRATE setting for a table space.
- Modify the file system caching policy for a table space.
- Enable or disable auto-resize for a DMS or automatic storage table space.
- Rebalance a regular or large automatic storage table space.

Invocation

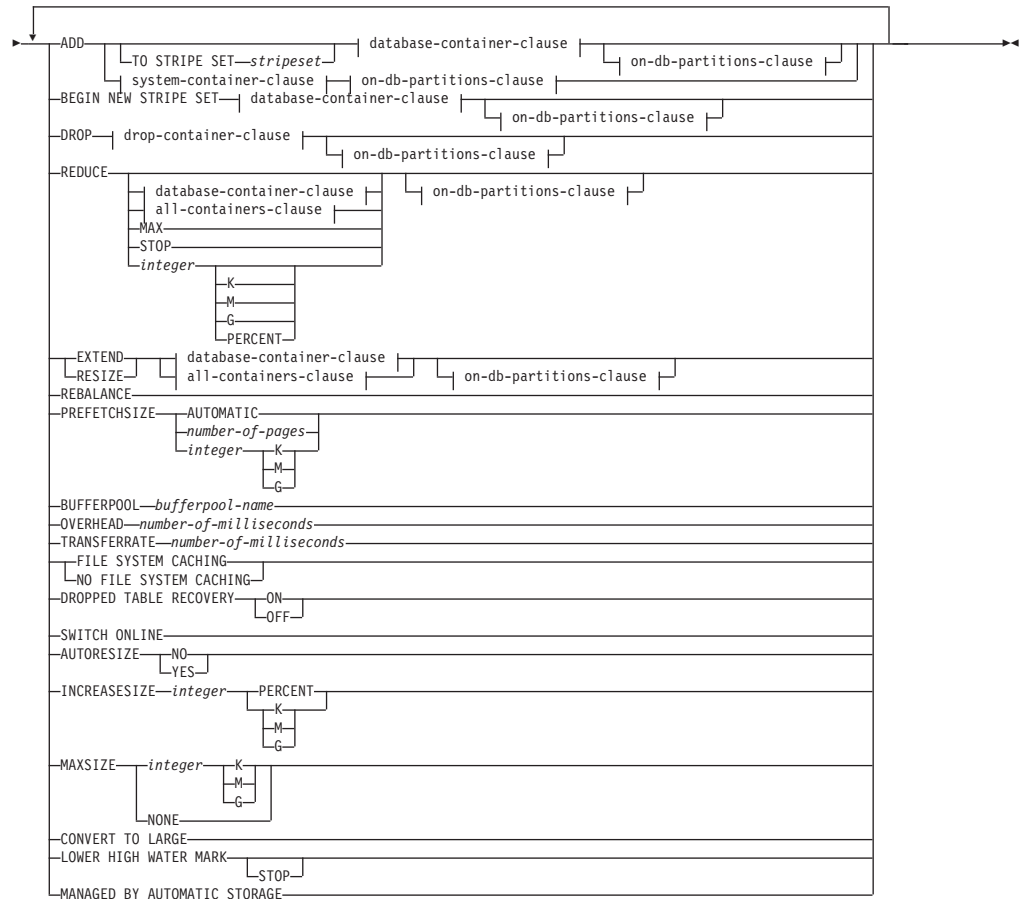
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

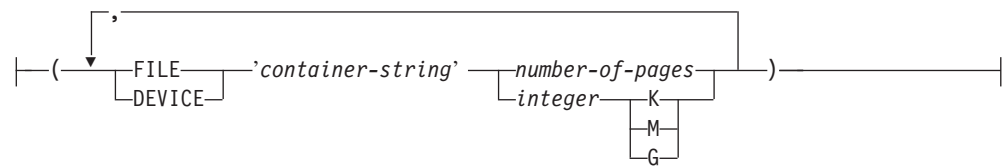
The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax

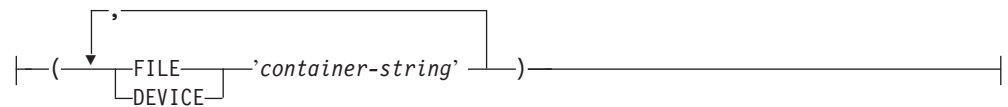
```
→ALTER TABLESPACE—tablespace-name→
```



database-container-clause:



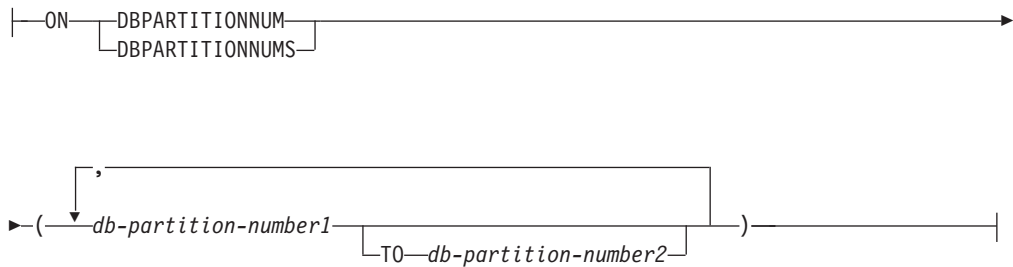
drop-container-clause:



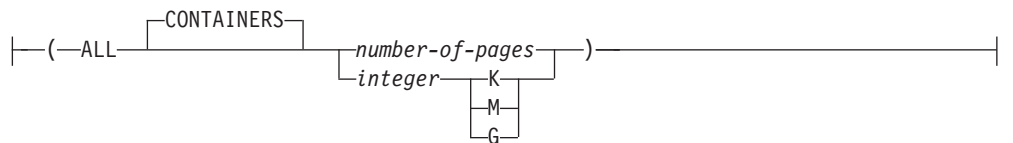
system-container-clause:



on-db-partitions-clause:



all-containers-clause:



Description

tablespace-name

Names the table space. This is a one-part name. It is a long SQL identifier (either ordinary or delimited).

ADD

Specifies that one or more new containers are to be added to the table space.

TO STRIPE SET *stripeset*

Specifies that one or more new containers are to be added to the table space, and that they will be placed into the given stripe set.

BEGIN NEW STRIPE SET

Specifies that a new stripe set is to be created in the table space, and that one or more containers are to be added to this new stripe set. Containers that are subsequently added using the `ADD` option will be added to this new stripe set unless `TO STRIPE SET` is specified.

DROP

Specifies that one or more containers are to be dropped from the table space.

REDUCE

For non-automatic storage table spaces, specifies that existing containers are to be reduced in size. The size specified is the size by which the existing container is decreased. If the *all-containers-clause* is specified, all containers in the table space will decrease by this size. If the reduction in size will result in a table space size that is smaller than the current high water mark, an attempt will be made to reduce the high water mark before attempting to reduce the containers. For non-automatic storage table spaces, the `REDUCE` clause must be followed by a *database-container-clause* or an *all-containers-clause*.

For automatic storage table spaces, specifies that the current high water mark is to be reduced, if possible, and that the size of the table space is to be reduced to the new high water mark. For automatic storage table spaces, the `REDUCE` clause must not be followed by a *database-container-clause* or an *all-containers-clause*.

Note: The REDUCE option with the MAX, numeric value, PERCENT, or STOP clauses, and the LOWER HIGH WATER MARK option including the STOP clause, are only available for database managed, and automatic storage managed, table spaces with the reclaimable storage attribute. Moreover, these options must be specified and run without any other options, including each other.

database-container-clause

Adds one or more containers to a DMS table space. The table space must identify a DMS table space that already exists at the application server.

all-containers-clause

Extends, reduces, or resizes all of the containers in a DMS table space. The table space must identify a DMS table space that already exists at the application server.

MAX

For automatic storage table spaces with reclaimable storage, specifies that the maximum number of extents should be moved to the beginning of the table space to lower the high water mark. Additionally, the size of the table space will be reduced to the new high water mark. This does not apply to non-automatic storage table spaces.

STOP

For automatic storage table spaces with reclaimable storage, interrupts the extent movement operation if in progress. This option is not available for non-automatic storage table spaces.

integer [K | M | G] or integer PERCENT

For automatic storage table spaces with reclaimable storage, specifies the numeric value by which the table space is to be reduced through extent movement. The value can be expressed in several ways:

- An integer specified without K, M, G, or PERCENT indicates that the numeric value is the number of pages by which the table space is to be reduced.
- An integer specified with K, M, or G indicates the reduction size in kilobytes, megabytes, or gigabytes, respectively. The value is first converted from bytes to number of pages based on the page size of the table space.
- An integer specified with PERCENT indicates the number of extents to move, as a percentage of the current size of the table space.

Once extent movement is complete, the table space size is reduced to the new high water mark. This option is not available for non-automatic storage table spaces.

on-db-partitions-clause

Specifies one or more database partitions for the corresponding container operations.

EXTEND

Specifies that existing containers are to be increased in size. The size specified is the size by which the existing container is increased. If the *all-containers-clause* is specified, all containers in the table space will increase by this size.

RESIZE

Specifies that the size of existing containers is to be changed. The size specified is the new size for the container. If the *all-containers-clause* is specified, all containers in the table space will be changed to this size. If the operation

affects more than one container, these containers must all either increase in size, or decrease in size. It is not possible to increase some while decreasing others (SQLSTATE 429BC).

database-container-clause

Adds one or more containers to a DMS table space. The table space must identify a DMS table space that already exists at the application server.

drop-container-clause

Drops one or more containers from a DMS table space. The table space must identify a DMS table space that already exists at the application server.

system-container-clause

Adds one or more containers to an SMS table space on the specified database partitions. The table space must identify an SMS table space that already exists at the application server. There must not be any containers on the specified database partitions for the table space (SQLSTATE 42921).

on-db-partitions-clause

Specifies one or more database partitions for the corresponding container operations.

all-containers-clause

Extends, reduces, or resizes all of the containers in a DMS table space. The table space must identify a DMS table space that already exists at the application server.

REBALANCE

For regular and large automatic storage table spaces, initiates the creation of containers on recently added storage paths, the drop of containers from storage paths that are in the "Drop Pending" state, or both. During the rebalance, data is moved into containers on new paths, and moved out of containers on dropped paths. The rebalance runs asynchronously in the background and does not affect the availability of data.

PREFETCHSIZE

Specifies to read in data needed by a query prior to it being referenced by the query, so that the query need not wait for I/O to be performed.

AUTOMATIC

Specifies that the prefetch size of a table space is to be updated automatically; that is, the prefetch size will be managed by DB2, using the following formula:

$$\text{Prefetch size} = (\text{number of containers}) * (\text{number of physical disks per container}) * (\text{extent size})$$

The number of physical disks per container defaults to 1, unless a value is specified through the DB2_PARALLEL_IO registry variable.

A DB2 database will update the prefetch size automatically whenever the number of containers in a table space changes (following successful execution of an ALTER TABLESPACE statement that adds or drops one or more containers). The prefetch size is updated at database start-up.

Automatic updating of the prefetch size can be turned off by specifying a numeric value in the PREFETCHSIZE clause.

number-of-pages

Specifies the number of PAGESIZE pages that will be read from the table

space when data prefetching is being performed. The prefetch size value can also be specified as an integer value followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the number of pages value for prefetch size.

BUFFERPOOL *bufferpool-name*

The name of the buffer pool used for tables in this table space. The buffer pool must currently exist in the database (SQLSTATE 42704). The database partition group of the table space must be defined for the bufferpool (SQLSTATE 42735).

OVERHEAD *number-of-milliseconds*

Any numeric literal (integer, decimal, or floating point) that specifies the I/O controller overhead and disk seek and latency time, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers. This value is used to determine the cost of I/O during query optimization.

TRANSFERRATE *number-of-milliseconds*

Any numeric literal (integer, decimal, or floating point) that specifies the time to read one page (4K or 8K) into memory, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers. This value is used to determine the cost of I/O during query optimization.

FILE SYSTEM CACHING or NO FILE SYSTEM CACHING

Specifies whether or not I/O operations will be cached at the file system level. Connections to the database must be terminated before a new caching policy takes effect. Note that I/O access to long or LOB data is buffered for both SMS and DMS containers.

FILE SYSTEM CACHING

All I/O operations in the target table space will be cached at the file system level.

NO FILE SYSTEM CACHING

All I/O operations will bypass the file system level cache.

DROPPED TABLE RECOVERY

Specifies whether or not tables that have been dropped from *tablespace-name* can be recovered using the RECOVER DROPPED TABLE ON option of the ROLLFORWARD DATABASE command. For partitioned tables, dropped table recovery is always on, even if dropped table recovery is turned off for non-partitioned tables in one or more table spaces.

ON

Specifies that dropped tables can be recovered.

OFF

Specifies that dropped tables cannot be recovered.

SWITCH ONLINE

Specifies that table spaces in OFFLINE state are to be brought online if their containers have become accessible. If the containers are not accessible, an error is returned (SQLSTATE 57048).

AUTORESIZE

Specifies whether or not the auto-resize capability of a database managed space (DMS) table space or an automatic storage table space is to be enabled. Auto-resizable table spaces automatically increase in size when they become full.

NO

Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be disabled. If the auto-resize capability is disabled, any values that have been previously specified for INCREASESIZE or MAXSIZE will not be kept.

YES

Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled.

INCREASESIZE *integer* PERCENT or INCREASESIZE *integer* K | M | G

Specifies the amount, per database partition, by which a table space that is enabled for auto-resize will automatically be increased when the table space is full, and a request for space has been made. The integer value must be followed by:

- PERCENT to specify the amount as a percentage of the table space size at the time that a request for space is made. When PERCENT is specified, the integer value must be between 0 and 100 (SQLSTATE 42615).
- K (for kilobytes), M (for megabytes), or G (for gigabytes) to specify the amount in bytes

Note that the actual value used might be slightly smaller or larger than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

MAXSIZE *integer* K | M | G or MAXSIZE NONE

Specifies the maximum size to which a table space that is enabled for auto-resize can automatically be increased.

integer

Specifies a hard limit on the size, per database partition, to which a DMS table space or an automatic storage table space can automatically be increased. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

NONE

Specifies that the table space is to be allowed to grow to file system capacity, or to the maximum table space size (described in “SQL limits”).

CONVERT TO LARGE

Modifies an existing regular DMS table space to be a large DMS table space. The table space and its contents are locked during conversion. This option can only be used on regular DMS table spaces. If an SMS table space, a temporary table space, or the system catalog table space is specified, an error is returned (SQLSTATE 560CF). You cannot convert a table space that contains a data partition of a partitioned table that has data partitions in another table space (SQLSTATE 560CF). Conversion cannot be reversed after being committed. If tables in the table space are defined with DATA CAPTURE CHANGES, consider the storage and capacity limits of the target table and table space.

LOWER HIGH WATER MARK

For both automatic storage and non-automatic storage table spaces with reclaimable storage, triggers the extent movement operation to move the maximum number of extents lower in the table space. Although the high water mark is lowered, the size of the table space is not reduced. This must be followed by an ALTER TABLESPACE REDUCE for automatic storage table

spaces or ALTER TABLESPACE REDUCE with the *database-container-clause* or *all-containers-clause* for non-automatic storage table spaces.

Note: The LOWER HIGH WATER MARK option including the STOP clause, and the REDUCE option with the MAX, numeric value, PERCENT, or STOP clauses, are only available for database managed and automatic storage managed table spaces with the reclaimable storage attribute. Moreover, these options must be specified and run without any other options, including each other.

STOP

For both automatic storage and non-automatic storage table spaces with reclaimable storage, interrupts the extent movement operation if in progress.

MANAGED BY AUTOMATIC STORAGE

Enables automatic storage for a database managed (DMS) table space. Once automatic storage is enabled, no further container operations can be executed on the table space. The table space being converted cannot be using RAW (DEVICE) containers.

Rules

- The BEGIN NEW STRIPE SET clause cannot be specified in the same statement as ADD, DROP, EXTEND, REDUCE, and RESIZE, unless those clauses are being directed to different database partitions (SQLSTATE 429BC).
- The stripe set value specified with the TO STRIPE SET clause must be within the valid range for the table space being altered (SQLSTATE 42615).
- When adding or removing space from the table space, the following rules must be followed:
 - EXTEND and RESIZE can be used in the same statement, provided that the size of each container is increasing (SQLSTATE 429BC).
 - REDUCE and RESIZE can be used in the same statement, provided that the size of each container is decreasing (SQLSTATE 429BC).
 - EXTEND and REDUCE cannot be used in the same statement, unless they are being directed to different database partitions (SQLSTATE 429BC).
 - ADD cannot be used with REDUCE or DROP in the same statement, unless they are being directed to different database partitions (SQLSTATE 429BC).
 - DROP cannot be used with EXTEND or ADD in the same statement, unless they are being directed to different database partitions (SQLSTATE 429BC).
- The AUTORESIZE, INCREASESIZE, or MAXSIZE clause cannot be specified for system managed space (SMS) table spaces, temporary table spaces that were created using automatic storage, or DMS table spaces that are defined to use raw device containers (SQLSTATE 42601).
- The INCREASESIZE or MAXSIZE clause cannot be specified if the table space is not auto-resizable (SQLSTATE 42601).
- When specifying a new maximum size for a table space, the value must be larger than the current size on each database partition (SQLSTATE 560B0).
- Container operations (ADD, EXTEND, RESIZE, DROP, or BEGIN NEW STRIPE SET) cannot be performed on automatic storage table spaces, because the database manager is controlling the space management of such table spaces (SQLSTATE 42858).
- Raw device containers cannot be added to an auto-resizable DMS table space (SQLSTATE 42601).

- The CONVERT TO LARGE clause cannot be specified in the same statement as any other clause (SQLSTATE 429BC).
- The REBALANCE clause cannot be specified with any other clause (SQLSTATE 429BC).
- The REBALANCE clause is only valid for regular and large automatic storage table spaces (SQLSTATE 42601). Temporary automatic storage table spaces should be dropped and recreated to take advantage of recently added storage paths or to have their containers removed from storage paths being dropped.
- Container operations and the REBALANCE clause cannot be specified if the table space is in the “DMS rebalancer is active” state (SQLSTATE 55041).

Notes

- Each container definition requires 53 bytes plus the number of bytes necessary to store the container name. The combined length of all container names for the table space cannot exceed 20 480 bytes (SQLSTATE 54034).
- Default container operations are container operations that are specified in the ALTER TABLESPACE statement, but that are not explicitly directed to a specific database partition. These container operations are sent to any database partition that is not listed in the statement. If these default container operations are not sent to any database partition, because all database partitions are explicitly mentioned for a container operation, a warning is returned (SQLSTATE 01589).
- Once space has been added or removed from a table space, and the transaction is committed, the contents of the table space may be rebalanced across the containers. Access to the table space is not restricted during rebalancing.
- If the table space is in OFFLINE state and the containers have become accessible, the user can disconnect all applications and connect to the database again to bring the table space out of OFFLINE state. Alternatively, SWITCH ONLINE option can bring the table space up (out of OFFLINE) while the rest of the database is still up and being used.
- If adding more than one container to a table space, it is recommended that they be added in the same statement so that the cost of rebalancing is incurred only once. An attempt to add containers to the same table space in separate ALTER TABLESPACE statements within a single transaction will result in an error (SQLSTATE 55041).
- Any attempts to extend, reduce, resize, or drop containers that do not exist will raise an error (SQLSTATE 428B2).
- When extending, reducing, or resizing a container, the container type must match the type that was used when the container was created (SQLSTATE 428B2).
- An attempt to change container sizes in the same table space, using separate ALTER TABLESPACE statements but within a single transaction, will raise an error (SQLSTATE 55041).
- In a partitioned database if more than one database partition resides on the same physical node, the same device or specific path cannot be specified for such database partitions (SQLSTATE 42730). For this environment, either specify a unique *container-string* for each database partition or use a relative path name.
- Although the table space definition is transactional and the changes to the table space definition are reflected in the catalog tables on commit, the buffer pool with the new definition cannot be used until the next time the database is started. The buffer pool in use, when the ALTER TABLESPACE statement was issued, will continue to be used in the interim.

- The REDUCE, RESIZE, or DROP option attempts to free unused extents, if necessary, for DMS table spaces, and the REDUCE option attempts to free unused extents for automatic storage table spaces. The removal of unused extents allows the table space high water mark to be reduced to a value that accurately represents the amount of space used, which, in turn, enables larger reductions in table space size.
- *Conversion to large DMS table spaces*:: After conversion, it is recommended that you issue the COMMIT statement and then increase the storage capacity of the table space.
 - If the table space is enabled for auto-resize, the MAXSIZE table space attribute should be increased, unless it is already set to NONE.
 - If the table space is not enabled for auto-resize:
 - Enable auto-resize by issuing the ALTER TABLESPACE statement with the AUTORESIZE YES option, or
 - Add more storage by adding stripe sets, extending the size of existing containers, or both

Indexes for tables in a converted table space must be reorganized or rebuilt before they can support large record identifiers (RIDs).

- The indexes can be reorganized using the REORG INDEXES ALL command (without the CLEANUP ONLY clause). Specify the ALLOW NO ACCESS option for partitioned tables.
- Alternatively, the tables can be reorganized (not INPLACE), which will rebuild all indexes and enable the tables to support more than 255 rows per page.

To determine which tables do not yet support large RIDs, use the ADMIN_GET_TAB_INFO table function.

- The rebalance of an automatic storage table space that has containers on a storage path in the “Drop Pending” state will drop those containers. New containers may need to be created to hold the data being moved off the dropped containers. There must be sufficient free space on the other storage paths in the database to allow those containers to be created, otherwise an error is returned SQLSTATE 57011. The actual amount of free space required depends on many factors, including the location of the high-water mark extent and the stripe sets being altered. However, to ensure that the operation will be successful, there should be at least enough free space on the remaining storage paths as there is space being consumed by the containers being dropped.
- If the REBALANCE clause is specified but the data server determines that there is no need to create new containers or drop existing ones, a rebalance does not occur and the statement succeeds with a warning (SQLSTATE 01690).
- In addition to adding containers on recently added paths, the REBALANCE operation may also be used to add containers on existing storage paths. Each stripe set in the table space is examined and storage paths that are not in use by a particular stripe set are identified. For each storage path identified, if there is sufficient free space on it then a new container will be created. The container will have the same size as the other containers in the stripe set. This would be beneficial if a given storage path ran out of space, table spaces stopped using it (by creating stripe sets on the other paths), and more storage was given to the path. In this case, no new paths have been added, but the rebalance will attempt to include that storage path in stripe sets where it wasn’t included before.
- Auto-resize can still occur while a rebalance of an automatic storage table space is in progress.

- When a DMS table space is enabled for automatic storage by the **MANAGED BY AUTOMATIC STORAGE** clause, that table space will have one or more stripe sets of user-defined (non-automatic storage) containers and one or more stripe sets of automatic storage containers. Rebalancing the table space (using the **REBALANCE** clause) removes all of the user-defined containers. The database manager might extend existing automatic storage containers or create new automatic storage containers to hold the data being moved from the user-defined containers.
- **Compatibilities:** For compatibility with versions earlier than Version 8, the keyword:
 - **NODE** can be substituted for **DBPARTITIONNUM**
 - **NODES** can be substituted for **DBPARTITIONNUMS**

Examples

Example 1: Add a device to the PAYROLL table space.

```
ALTER TABLESPACE PAYROLL
  ADD (DEVICE '/dev/rhdisk9' 10000)
```

Example 2: Change the prefetch size and I/O overhead for the ACCOUNTING table space.

```
ALTER TABLESPACE ACCOUNTING
  PREFETCHSIZE 64
  OVERHEAD 19.3
```

Example 3: Create a table space TS1, then resize the containers so that all of the containers have 2000 pages. (Three different ALTER TABLESPACE statements that will accomplish this resizing are shown.)

```
CREATE TABLESPACE TS1
  MANAGED BY DATABASE
  USING (FILE '/conts/cont0' 1000,
        DEVICE '/dev/rcont1' 500,
        FILE 'cont2' 700)
ALTER TABLESPACE TS1
  RESIZE (FILE '/conts/cont0' 2000,
        DEVICE '/dev/rcont1' 2000,
        FILE 'cont2' 2000)
```

OR

```
ALTER TABLESPACE TS1
  RESIZE (ALL 2000)
```

OR

```
ALTER TABLESPACE TS1
  EXTEND (FILE '/conts/cont0' 1000,
        DEVICE '/dev/rcont1' 1500,
        FILE 'cont2' 1300)
```

Example 4: Extend all of the containers in the DATA_TS table space by 1000 pages.

```
ALTER TABLESPACE DATA_TS
  EXTEND (ALL 1000)
```

Example 5: Resize all of the containers in the INDEX_TS table space to 100 megabytes (MB).

```
ALTER TABLESPACE INDEX_TS
  RESIZE (ALL 100 M)
```


Example 6: Add three new containers. Extend the first container, and resize the second.

```
ALTER TABLESPACE TS0
  ADD (FILE 'cont2' 2000, FILE 'cont3' 2000)
  ADD (FILE 'cont4' 2000)
  EXTEND (FILE 'cont0' 100)
  RESIZE (FILE 'cont1' 3000)
```

Example 7: Table space TSO exists on database partitions 0, 1 and 2. Add a new container to database partition 0. Extend all of the containers on database partition 1. Resize a container on all database partitions other than the ones that were explicitly specified (that is, database partitions 0 and 1).

```
ALTER TABLESPACE TS0
  ADD (FILE 'A' 200) ON DBPARTITIONNUM (0)
  EXTEND (ALL 200) ON DBPARTITIONNUM (1)
  RESIZE (FILE 'B' 500)
```

The RESIZE clause is the default container clause in this example, and will be executed on database partition 2, because other operations are being explicitly sent to database partitions 0 and 1. If, however, there had only been these two database partitions, the statement would have succeeded, but returned a warning (SQL1758W) that default containers had been specified but not used.

Example 8: Enable the auto-resize option for table space DMS_TS1, and set its maximum size to 256 megabytes.

```
ALTER TABLESPACE DMS_TS1
  AUTORESIZE YES MAXSIZE 256 M
```

Example 9: Enable the auto-resize option for table space AUTOSTORE1, and change its growth rate to 5%.

```
ALTER TABLESPACE AUTOSTORE1
  AUTORESIZE YES INCREASESIZE 5 PERCENT
```

Example 10: Change the growth rate for an auto-resizable table space named MY_TS to 512 kilobytes, and set its maximum size to be as large as possible.

```
ALTER TABLESPACE MY_TS
  INCREASESIZE 512 K MAXSIZE NONE
```

Example 11: Enable automatic storage for database managed table space DMS_TS10

```
ALTER TABLESPACE DMS_TS10
  MANAGED BY AUTOMATIC STORAGE
```

Example 12: An ALTER DATABASE statement removed the paths /db2/filesystem1 and /db2/filesystem2 from the currently connected database. The table spaces named PRODTS1, PRODTS2, and PRODTS3 were the only table spaces using the removed paths. Rebalance these table spaces. Three ALTER TABLESPACE statements must be used.

```
ALTER TABLESPACE PRODTS1 REBALANCE
ALTER TABLESPACE PRODTS2 REBALANCE
ALTER TABLESPACE PRODTS3 REBALANCE
```

Example 13: Enable automatic storage for database managed table space DATA1 and remove all of the existing non-automatic storage containers from the table space. The first statement must be committed before the second statement can be run.

```
ALTER TABLESPACE DATA1 MANAGED BY AUTOMATIC STORAGE
ALTER TABLESPACE DATA1 REBALANCE
```

Example 14: Trigger extent movement for an automatic storage table space with reclaimable storage attribute, to reduce the size of the containers by 10MB.

```
ALTER TABLESPACE REDUCE 10 M
```

Example 15: Trigger extent movement for a non-automatic storage table space with reclaimable storage attribute and subsequently reduce the size of each container by 10MB.

```
ALTER TABLESPACE LOWER HIGH WATER MARK  
ALTER TABLESPACE REDUCE (ALL CONTAINERS 10 M)
```

ALTER VIEW

The ALTER VIEW statement modifies an existing view by:

- Altering a reference type column to add a scope
- Enabling or disabling a view for use in query optimization

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

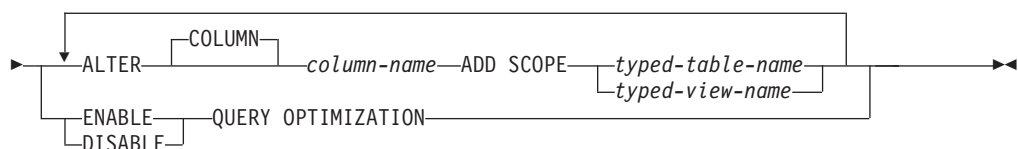
- ALTERIN privilege on the schema of the view
- Owner of the view to be altered
- CONTROL privilege on the view to be altered
- DBADM authority

To enable or disable a view for use in query optimization, the privileges held by the authorization ID of the statement must also include at least one of the following for each of the tables or underlying tables of views that are referenced in the FROM clause of the view fullselect:

- ALTER privilege on the table
- ALTERIN privilege on the schema of the table
- DBADM authority

Syntax

```
▶▶—ALTER VIEW—view-name—————▶▶
```



Description

view-name

Specifies the view that is to be changed. It must be a view that is described in the catalog.

ALTER COLUMN *column-name*

Specifies the name of the column that is to be altered. The *column-name* must identify an existing column of the view (SQLSTATE 42703). The name cannot be qualified.

ADD SCOPE

Adds a scope to an existing reference type column that does not already have a scope defined (SQLSTATE 428DK). The column must not be inherited from a superview (SQLSTATE 428DJ).

typed-table-name

Specifies the name of a typed table. The data type of *column-name* must be REF(S), where S is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name

Specifies the name of a typed view. The data type of *column-name* must be REF(S), where S is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION

Specifies whether or not the view and any associated statistics are to be used to improve the optimization of queries. DISABLE QUERY OPTIMIZATION is the default when a view is created.

ENABLE QUERY OPTIMIZATION

Specifies that the view includes statistics that can be used to improve the optimization of queries that involve this view or queries that include subqueries similar to the fullselect of this view.

DISABLE QUERY OPTIMIZATION

Specifies that the view and any associated statistics are not to be used to improve the optimization of queries.

Rules

- A view cannot be enabled for query optimization if:
 - The view directly or indirectly references a materialized query table (MQT). Note that an MQT or statistical view can reference a statistical view.
 - It is a typed view

Notes

- To be considered for optimizing a query, a view:
 - Cannot contain aggregation or distinct operations
 - Cannot contain union, except, or intersect operations
 - Cannot contain scalar aggregate (OLAP) functions
- If a view is altered to disable query optimization, cached query plans that used the view for query optimization are invalidated. If a view is altered to enable query optimization, cached query plans are invalidated if they reference the same tables as the newly enabled view references, either directly or indirectly

through other views. The invalidation of these cached query plans results in implicit revalidation that takes the view's changed query optimization property into account.

The query optimization property for a view has no impact on static embedded SQL statements.

AUDIT

The AUDIT statement determines the audit policy that is to be used for a particular database or database object at the current server. Whenever the object is in use, it is audited according to that policy.

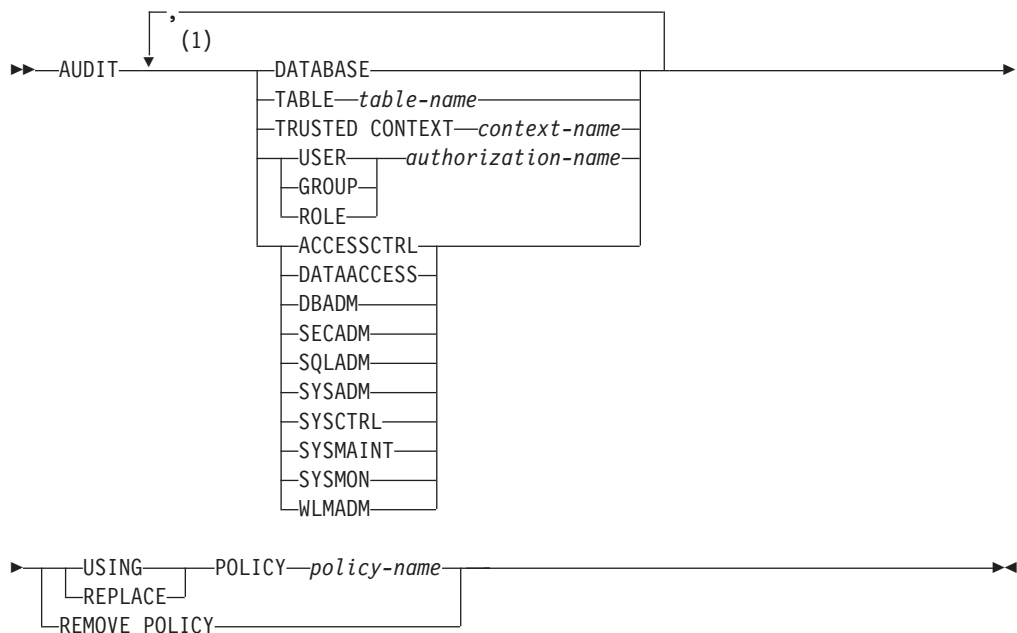
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Notes:

- 1 Each clause (with the same object name, if applicable) can be specified at most once (SQLSTATE 42713).

Description

ACCESSCTRL, DATAACCESS, DBADM, SECADM, SQLADM, SYSADM, SYSCTRL, SYSMANT, SYSMON, or WLMADM

Specifies that an audit policy is to be associated with or removed from the specified authority. All auditable events that are initiated by a user who holds the specified authority, even if that authority is not required for the event, will be audited according to the associated audit policy.

DATABASE

Specifies that an audit policy is to be associated with or removed from the database at the current server. All auditable events that occur within the database are audited according to the associated audit policy.

TABLE *table-name*

Specifies that an audit policy is to be associated with or removed from *table-name*. The *table-name* must identify a table, materialized query table (MQT), or nickname that exists at the current server (SQLSTATE 42704). It cannot be a view, a catalog table, a created temporary table, a declared temporary table (SQLSTATE 42995), or a typed table (SQLSTATE 42997). Only EXECUTE category audit events, with or without data, will be generated when the table is accessed, even if the policy indicates that other categories should be audited.

TRUSTED CONTEXT *context-name*

Specifies that an audit policy is to be associated with or removed from *context-name*. The *context-name* must identify a trusted context that exists at the current server (SQLSTATE 42704). All auditable events that happen within the trusted connection defined by the trusted context *context-name* will be audited according to the associated audit policy.

USER *authorization-name*

Specifies that an audit policy is to be associated with or removed from the user with authorization ID *authorization-name*. All auditable events that are initiated by *authorization-name* will be audited according to the associated audit policy.

GROUP *authorization-name*

Specifies that an audit policy is to be associated with or removed from the group with authorization ID *authorization-name*. All auditable events that are initiated by users who are members of *authorization-name* will be audited according to the associated audit policy. If user membership in a group cannot be determined, the policy will not apply to that user.

ROLE *authorization-name*

Specifies that an audit policy is to be associated with or removed from the role with authorization ID *authorization-name*. The *authorization-name* must identify a role that exists at the current server (SQLSTATE 42704). All auditable events that are initiated by users who are members of *authorization-name* will be audited according to the associated audit policy. Indirect role membership through other roles or groups is valid.

USING, REMOVE, or REPLACE

Specifies whether the audit policy should be used, removed, or replaced for the specified object.

USING

Specifies that the audit policy is to be used for the specified object. An existing audit policy must not already be defined for the object (SQLSTATE 5U041). If an audit policy already exists, it must be removed or replaced.

REMOVE

Specifies that the audit policy is to be removed from the specified object. Use of the object will no longer be audited according to the audit policy. The association is deleted from the catalog when the audit policy is removed from the object.

REPLACE

Specifies that the audit policy is to replace an existing audit policy for the specified object. This combines both REMOVE and USING options into one step to ensure that there is no period of time in which an audit policy does not apply to the specified object. If a policy was not in use for the specified object, REPLACE is equivalent to USING.

POLICY *policy-name*

Specifies the audit policy that is to be used to determine audit settings. The *policy-name* must identify an existing audit policy at the current server (SQLSTATE 42704).

Rules

- An AUDIT-exclusive SQL statement must be followed by a COMMIT or ROLLBACK statement (SQLSTATE 5U021). AUDIT-exclusive SQL statements are:
 - AUDIT
 - CREATE AUDIT POLICY, ALTER AUDIT POLICY, or DROP (AUDIT POLICY)
 - DROP (ROLE or TRUSTED CONTEXT if it is associated with an audit policy)
- An AUDIT-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.
- An object can be associated with no more than one policy (SQLSTATE 5U042).

Notes

- Changes are written to the catalog, but do not take effect until after a COMMIT statement executes.
- Changes do not take effect until the next unit of work that references the object to which the audit policy applies. For example, if the audit policy is in use for the database, no current units of work will begin auditing according to the policy until after a COMMIT or a ROLLBACK statement completes.
- Views accessing a table that is associated with an audit policy are audited according to the underlying table's policy.
- The audit policy that applies to a table does not apply to a materialized query table (MQT) based on that table. It is recommended that if you associate an audit policy with a table, you also associate that policy with any MQT based on that table. The compiler might automatically use an MQT, even though an SQL statement references the base table; however, the audit policy in use for the base table will still be in effect.
- When a switch user operation is performed within a trusted context, all audit policies are re-evaluated according to the new user, and no policies from the old user are used for the current session. This applies specifically to audit policies associated directly with the user, the user's group or role memberships, and the user's authorities. For example, if the current session was audited because the previous user was a member of an audited role, and the switched-to user is not a member of that role, that policy no longer applies to the session.
- When a SET SESSION USER statement is executed, the audit policies associated with the original user (and that user's group and role memberships and authorities) are combined with the policies that are associated with the user

specified in the SET SESSION USER statement. The audit policies associated with the original user are still in effect, as are the policies for the user specified in the SET SESSION USER statement. If multiple SET SESSION USER statements are issued within a session, only the audit policies associated with the original user and the current user are considered.

- If the object with which an audit policy is associated is dropped, the association to the audit policy is removed from the catalog and no longer exists. If that object is recreated at some later time, the object will not be audited according to the policy that was associated with it when the object was dropped.

Examples

Example 1: Use the audit policy DBAUDPRF to determine the audit settings for the database at the current server.

```
AUDIT DATABASE USING POLICY DBAUDPRF
```

Example 2: Remove the audit policy from the EMPLOYEE table.

```
AUDIT TABLE EMPLOYEE REMOVE POLICY
```

Example 3: Use the audit policy POWERUSERS to determine the audit settings for the authorities SYSADM, DBADM, and SECADM, as well as the group DBAS.

```
AUDIT SYSADM, DBADM, SECADM, GROUP DBAS USING POLICY POWERUSERS
```

Example 4: Replace the audit policy for the role TELLER with the new policy TELLERPRF.

```
AUDIT ROLE TELLER REPLACE POLICY TELLERPRF
```

COMMENT

The COMMENT statement adds or replaces comments in the catalog descriptions of various objects.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

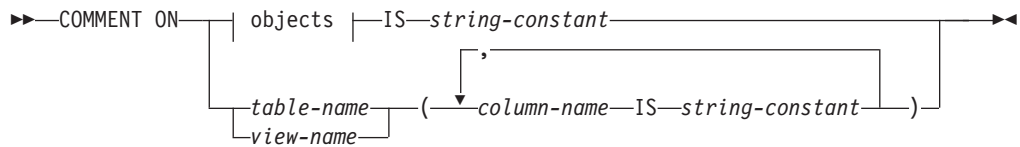
The privileges held by the authorization ID of the statement must include at least one of the following:

- Owner of the object (underlying table for column or constraint), as recorded in the OWNER column of the catalog view for the object
- ALTERIN privilege on the schema (applicable only to objects that allow more than one-part names)
- CONTROL privilege on the object (applicable only to index, package, table, or view objects)
- ALTER privilege on the object (applicable only to table objects)
- The WITH ADMIN OPTION (applicable only to roles)
- WLMADM authority (applicable only to workload manager objects)

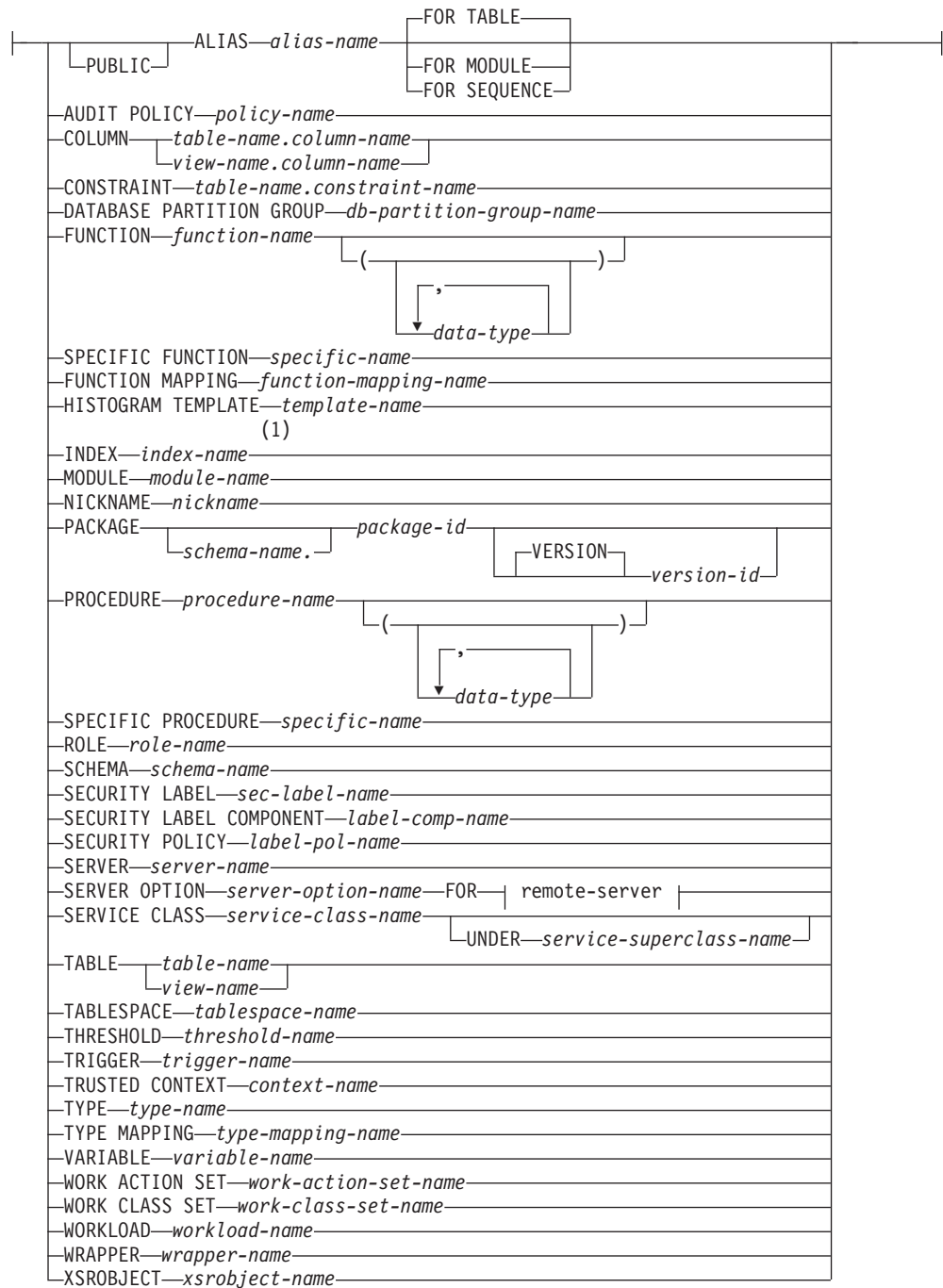
- SECADM authority (applicable only to audit policy, role, security label, security label component, security policy, or trusted context objects)
- DBADM authority (applicable to all objects except audit policy, role, security label, security label component, security policy, or trusted context objects)

Note that for table space or database partition group, and bufferpools, the authorization ID must have SYSCTRL or SYSADM authority.

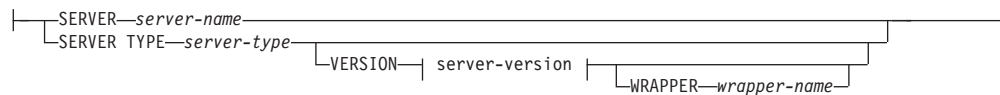
Syntax



objects:



remote-server:



DATABASE PARTITION GROUP *db-partition-group-name*

Indicates a comment will be added or replaced for a database partition group. The *db-partition-group-name* must identify a distinct database partition group that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.DBPARTITIONGROUPS catalog view for the row that describes the database partition group.

FUNCTION

Indicates a comment will be added or replaced for a function. The function instance specified must be a user-defined function or function template that exists at the current server. The user-defined function must not identify a module function (SQLSTATE 42883).

There are several different ways available to identify the function instance:

FUNCTION *function-name*

Identifies the particular function, and is valid only if there is exactly one function with the *function-name*. The function thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one specific instance of the function in the named or implied schema, an error (SQLSTATE 42725) is raised.

FUNCTION *function-name (data-type,...)*

Provides the function signature, which uniquely identifies the function to be commented upon. The function selection algorithm is *not* used.

function-name

Gives the function name of the function to be commented upon. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)

Must match the data types that were specified on the CREATE FUNCTION statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific function for which to add or replace the comment.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(n) does not need to match the defined value for n since 0 <n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

(Note that the FOR BIT DATA attribute is not considered part of the signature for matching purposes. So, for example, a CHAR FOR BIT DATA specified in the signature would match a function defined with CHAR only, and vice versa.)

If no function with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC FUNCTION *specific-name*

Indicates that comments will be added or replaced for a function (see FUNCTION for other methods of identifying a function). Identifies the particular user-defined function that is to be commented upon, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

It is not possible to comment on a function that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

The comment replaces the value of the REMARKS column of the SYSCAT.ROUTINES catalog view for the row that describes the function.

FUNCTION MAPPING *function-mapping-name*

Indicates a comment will be added or replaced for a function mapping. The *function-mapping-name* must identify a function mapping that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.FUNCMAPPINGS catalog view for the row that describes the function mapping.

HISTOGRAM TEMPLATE *template-name*

Indicates a comment will be added or replaced for a histogram template. The *template-name* must identify a histogram template that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.HISTOGRAMTEMPLATES catalog view for the row that describes the histogram template.

INDEX *index-name*

Indicates a comment will be added or replaced for an index or index specification. The *index-name* must identify either a distinct index or an index specification that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.INDEXES catalog view for the row that describes the index or index specification.

MODULE *module-name*

Indicates a comment will be added or replaced for a module. The *module-name* must identify a module that exists at the current server (SQLSTATE 42704).

NICKNAME *nickname*

Indicates a comment will be added or replaced for a nickname. The *nickname* must be a nickname that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the nickname.

PACKAGE *schema-name.package-id*

Indicates that a comment will be added or replaced for a package. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The schema name and package ID, together with the implicitly or explicitly specified version ID, must identify a package that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.PACKAGES catalog view for the row that describes the package.

VERSION *version-id*

Identifies which package version is to be commented on. If a value is not specified, the version defaults to the empty string. If multiple packages with the same package name but different versions exist, only one package version can be commented on in one invocation of the COMMENT statement. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

PROCEDURE

Indicates a comment will be added or replaced for a procedure. The procedure instance specified must be a procedure that exists at the current server. The procedure must not identify a module procedure (SQLSTATE 42883).

There are several different ways available to identify the procedure instance:

PROCEDURE *procedure-name*

Identifies the particular procedure, and is valid only if there is exactly one procedure with the *procedure-name* in the schema. The procedure thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no procedure by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one specific instance of the procedure in the named or implied schema, an error (SQLSTATE 42725) is raised.

PROCEDURE *procedure-name (data-type,...)*

This is used to provide the procedure signature, which uniquely identifies the procedure to be commented upon.

procedure-name

Gives the procedure name of the procedure to be commented upon. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)

Must match the data types that were specified on the CREATE PROCEDURE statement in the corresponding position. For federated procedures, the data type must match the local catalog information.

The number of data types, and the logical concatenation of the data types is used to identify the specific procedure for which to add or replace the comment.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement or the local catalog information, in the case of a federated procedure.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC PROCEDURE *specific-name*

Indicates that comments will be added or replaced for a procedure (see PROCEDURE for other methods of identifying a procedure). Identifies the particular procedure that is to be commented upon, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

It is not possible to comment on a procedure that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

The comment replaces the value of the REMARKS column of the SYSCAT.ROUTINES catalog view for the row that describes the procedure.

ROLE *role-name*

Indicates a comment will be added or replaced for a role. The *role-name* must identify a role that exists at the current server (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.ROLES catalog view for the row that describes the role.

SCHEMA *schema-name*

Indicates a comment will be added or replaced for a schema. The *schema-name* must identify a schema that exists at the current server (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.SCHEMATA catalog view for the row that describes the schema.

SECURITY LABEL *sec-label-name*

Indicates that a comment will be added or replaced for the security label named *sec-label-name*. The name must be qualified with a security policy and must identify a security label that exists at the current server (SQLSTATE

42704). The comment replaces the value for the REMARKS column of the SYSCAT.SECURITYLABELS catalog view for the row that describes the security label.

SECURITY LABEL COMPONENT *label-comp-name*

Indicates that a comment will be added or replaced for the security label component named *label-comp-name*. The *label-comp-name* must identify a security label component that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SECURITYLABELCOMPONENTS catalog view for the row that describes the security label component.

SECURITY POLICY *label-pol-name*

Indicates that a comment will be added or replaced for the security policy named *label-pol-name*. The *label-pol-name* must identify a security policy that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SECURITYPOLICIES catalog view for the row that describes the security policy.

SERVER *server-name*

Indicates a comment will be added or replaced for a data source. The *server-name* must identify a data source that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SERVERS catalog view for the row that describes the data source.

SERVER OPTION *server-option-name* **FOR** *remote-server*

Indicates a comment will be added or replaced for a server option.

server-option-name

Identifies a server option. This option must be one that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SERVEROPTIONS catalog view for the row that describes the server option.

remote-server

Describes the data source to which the *server-option* applies.

SERVER *server-name*

Names the data source to which the *server-option* applies. The *server-name* must identify a data source that exists at the current server.

TYPE *server-type*

Specifies the type of data source—for example, DB2 for z/OS or Oracle—to which the *server-option* applies. The *server-type* can be specified in either lower- or uppercase; it will be stored in uppercase in the catalog.

VERSION

Specifies the version of the data source identified by *server-name*.

version

Specifies the version number. *version* must be an integer.

release

Specifies the number of the release of the version denoted by *version*. *release* must be an integer.

mod

Specifies the number of the modification of the release denoted by *release*. *mod* must be an integer.

version-string-constant

Specifies the complete designation of the version. The *version-string-constant* can be a single value (for example, '8i'); or it can be the concatenated values of *version*, *release*, and, if applicable, *mod* (for example, '8.0.3').

WRAPPER *wrapper-name*

Identifies the wrapper that is used to access the data source referenced by *server-name*.

SERVICE CLASS *service-class-name*

Indicates a comment will be added or replaced for a service class. The *service-class-name* must identify a service class that exists at the current server (SQLSTATE 42704). To add or replace a comment for a service subclass, the *service-superclass-name* must be specified using the UNDER clause. The comment replaces the value for the REMARKS column of the SYSCAT.SERVICECLASSES catalog view for the row that describes the service class.

UNDER *service-superclass-name*

Specifies the service superclass of the service subclass when adding or replacing a comment for a service subclass. The *service-superclass-name* must identify a service superclass that exists at the current server (SQLSTATE 42704).

TABLE *table-name* or *view-name*

Indicates a comment will be added or replaced for a table or view. The *table-name* or *view-name* must identify a table or view (not an alias or nickname) that exists at the current server (SQLSTATE 42704) and must not identify a declared temporary table (SQLSTATE 42995). The comment replaces the value for the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the table or view.

TABLESPACE *tablespace-name*

Indicates a comment will be added or replaced for a table space. The *tablespace-name* must identify a distinct table space that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TABLESPACES catalog view for the row that describes the table space.

THRESHOLD *threshold-name*

Indicates a comment will be added or replaced for a threshold. The *threshold-name* must identify a threshold that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.THRESHOLDS catalog view for the row that describes the threshold.

TRIGGER *trigger-name*

Indicates a comment will be added or replaced for a trigger. The *trigger-name* must identify a distinct trigger that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TRIGGERS catalog view for the row that describes the trigger.

TRUSTED CONTEXT *context-name*

Indicates a comment will be added or replaced for a trusted context. The *context-name* must identify a trusted context that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.CONTEXTS catalog view for the row that describes the trusted context.

TYPE *type-name*

Indicates a comment will be added or replaced for a user-defined type. The *type-name* must identify a user-defined type that exists at the current server (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.DATATYPES catalog view for the row that describes the user-defined type.

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

TYPE MAPPING *type-mapping-name*

Indicates a comment will be added or replaced for a user-defined data type mapping. The *type-mapping-name* must identify a data type mapping that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TYPEMAPPINGS catalog view for the row that describes the mapping.

VARIABLE *variable-name*

Indicates a comment will be added or replaced for a global variable. The *variable-name* must identify a global variable that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.VARIABLES catalog view for the row that describes the variable.

WORK ACTION SET *work-action-set-name*

Indicates a comment will be added or replaced for a work action set. The *work-action-set-name* must identify a work action set that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WORKACTIONSETS catalog view for the row that describes the work action set.

WORK CLASS SET *work-class-set-name*

Indicates a comment will be added or replaced for a work class set. The *work-class-set-name* must identify a work class set that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WORKCLASSSETS catalog view for the row that describes the work class set.

WORKLOAD *workload-name*

Indicates that a comment will be added or replaced for a workload. The *workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WORKLOADS catalog view for the row that describes the workload.

WRAPPER *wrapper-name*

Indicates a comment will be added or replaced for a wrapper. The *wrapper-name* must identify a wrapper that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WRAPPERS catalog view for the row that describes the wrapper.

XSRBJECT *xsobject-name*

Indicates a comment will be added or replaced for an XSR object. The *xsobject-name* must identify an XSR object that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.XSRBJECTS catalog view for the row that describes the XSR object.

IS *string-constant*

Specifies the comment to be added or replaced. The *string-constant* can be any character string constant of up to 254 bytes. (Carriage return and line feed each count as 1 byte.)

table-name | view-name ({ *column-name IS string-constant* } ...)

This form of the COMMENT statement provides the ability to specify comments for multiple columns of a table or view. The column names must not be qualified, each name must identify a column of the specified table or view, and the table or view must exist at the current server. The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

A comment cannot be made on a column of an inoperative view (SQLSTATE 51024).

Notes

- **Compatibilities:** For compatibility with previous versions of DB2 products:
 - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
 - DISTINCT TYPE *type-name* can be specified in place of TYPE *type-name*
 - DATA TYPE *type-name* can be specified in place of TYPE *type-name*
 - SYNONYM can be specified in place of ALIAS

Examples

Example 1: Add a comment for the EMPLOYEE table.

```
COMMENT ON TABLE EMPLOYEE
  IS 'Reflects first quarter reorganization'
```

Example 2: Add a comment for the EMP_VIEW1 view.

```
COMMENT ON TABLE EMP_VIEW1
  IS 'View of the EMPLOYEE table without salary information'
```

Example 3: Add a comment for the EDLEVEL column of the EMPLOYEE table.

```
COMMENT ON COLUMN EMPLOYEE.EDLEVEL
  IS 'highest grade level passed in school'
```

Example 4: Add comments for two different columns of the EMPLOYEE table.

```
COMMENT ON EMPLOYEE
  (WORKDEPT IS 'see DEPARTMENT table for names',
  EDLEVEL IS 'highest grade level passed in school' )
```

Example 5: Pellow wants to comment on the CENTRE function, which he created in his PELLOW schema, using the signature to identify the specific function to be commented on.

```
COMMENT ON FUNCTION CENTRE (INT,FLOAT)
  IS 'Frank''s CENTRE fctn, uses Chebychev method'
```

Example 6: McBride wants to comment on another CENTRE function, which she created in the PELLOW schema, using the specific name to identify the function instance to be commented on:

```
COMMENT ON SPECIFIC FUNCTION PELLOW.FOCUS92 IS
  'Louise''s most triumphant CENTRE function, uses the
  Brownian fuzzy-focus technique'
```

Example 7: Comment on the function ATOMIC_WEIGHT in the CHEM schema, where it is known that there is only one function with that name:

```
COMMENT ON FUNCTION CHEM.ATOMIC_WEIGHT
IS 'takes atomic nbr, gives atomic weight'
```

Example 8: Eigler wants to comment on the SEARCH procedure, which he created in his EIGLER schema, using the signature to identify the specific procedure to be commented on.

```
COMMENT ON PROCEDURE SEARCH (CHAR,INT)
IS 'Frank''s mass search and replace algorithm'
```

Example 9: Macdonald wants to comment on another SEARCH function, which he created in the EIGLER schema, using the specific name to identify the procedure instance to be commented on:

```
COMMENT ON SPECIFIC PROCEDURE EIGLER.DESTROY IS
'Patrick''s mass search and destroy algorithm'
```

Example 10: Comment on the procedure OSMOSIS in the BIOLOGY schema, where it is known that there is only one procedure with that name:

```
COMMENT ON PROCEDURE BIOLOGY.OSMOSIS
IS 'Calculations modelling osmosis'
```

Example 11: Comment on an index specification named INDEXSPEC.

```
COMMENT ON INDEX INDEXSPEC
IS 'An index specification that indicates to the optimizer
that the table referenced by nickname NICK1 has an index.'
```

Example 12: Comment on the wrapper whose default name is NET8.

```
COMMENT ON WRAPPER NET8
IS 'The wrapper for data sources associated with
Oracle's Net8 client software.'
```

Example 13: Create a comment on the XML schema HR.EMPLOYEE.

```
COMMENT ON XSROBJECT HR.EMPLOYEE
IS 'This is the base XML Schema for employee data.'
```

Example 14: Create a comment for trusted context APPSERVER.

```
COMMENT ON TRUSTED CONTEXT APPSERVER
IS 'WebSphere Server'
```

CREATE AUDIT POLICY

The CREATE AUDIT POLICY statement defines an auditing policy at the current server. The policy determines what categories are to be audited; it can then be applied to other database objects to determine how the use of those objects is to be audited.

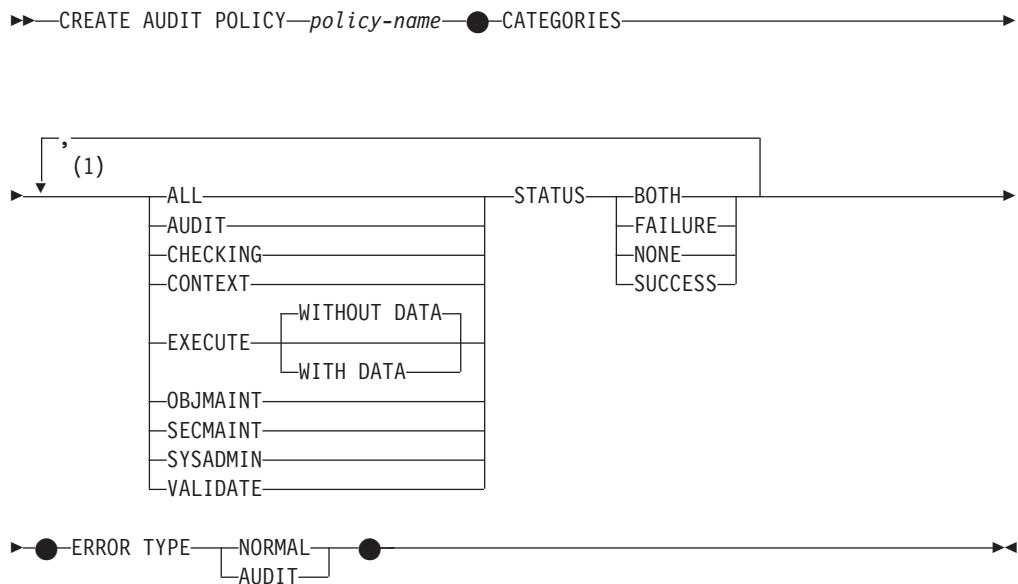
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Notes:

- 1 Each category can be specified at most once (SQLSTATE 42614), and no other category can be specified if ALL is specified (SQLSTATE 42601).

Description

policy-name

Names the audit policy. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *policy-name* must not identify an audit policy already described in the catalog (SQLSTATE 42710). The name must not begin with the characters 'SYS' (SQLSTATE 42939).

CATEGORIES

A list of one or more audit categories for which a status is specified. If ALL is not specified, the STATUS of any category that is not explicitly specified is set to NONE.

ALL

Sets all categories to the same status. The EXECUTE category is WITHOUT DATA.

AUDIT

Generates records when audit settings are changed or when the audit log is accessed.

CHECKING

Generates records during authorization checking of attempts to access or manipulate database objects or functions.

CONTEXT

Generates records to show the operation context when a database operation is performed.

EXECUTE

Generates records to show the execution of SQL statements.

WITHOUT DATA or WITH DATA

Specifies whether or not input data values provided for any host variables and parameter markers should be logged as part of the EXECUTE category.

WITHOUT DATA

Input data values provided for any host variables and parameter markers are not logged as part of the EXECUTE category. WITHOUT DATA is the default.

WITH DATA

Input data values provided for any host variables and parameter markers are logged as part of the EXECUTE category. Not all input values are logged; specifically, LOB, LONG, XML, and structured type parameters appear as the null value. Date, time, and timestamp fields are logged in ISO format. The input data values are converted to the database code page before being logged. If code page conversion fails, no errors are returned and the unconverted data is logged.

OBJMAINT

Generates records when data objects are created or dropped.

SECMAINT

Generates records when object privileges, database privileges, or DBADM authority is granted or revoked. Records are also generated when the database manager security configuration parameters **sysadm_group**, **sysctrl_group**, or **sysmaint_group** are modified.

SYSADMIN

Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.

VALIDATE

Generates records when users are authenticated or when system security information related to a user is retrieved.

STATUS

Specifies a status for the specified category.

BOTH

Successful and failing events will be audited.

FAILURE

Only failing events will be audited.

SUCCESS

Only successful events will be audited.

NONE

No events in this category will be audited.

ERROR TYPE

Specifies whether audit errors are to be returned or ignored.

NORMAL

Any errors generated by the audit are ignored and only the SQLCODEs for errors associated with the operation being performed are returned to the application.

AUDIT

All errors, including errors occurring within the audit facility itself, are returned to the application.

Rules

- An AUDIT-exclusive SQL statement must be followed by a COMMIT or ROLLBACK statement (SQLSTATE 5U021). AUDIT-exclusive SQL statements are:
 - AUDIT
 - CREATE AUDIT POLICY, ALTER AUDIT POLICY, or DROP (AUDIT POLICY)
 - DROP (ROLE or TRUSTED CONTEXT if it is associated with an audit policy)
- An AUDIT-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Only one uncommitted AUDIT-exclusive SQL statement is allowed at a time across all database partitions. If an uncommitted AUDIT-exclusive SQL statement is executing, subsequent AUDIT-exclusive SQL statements wait until the current AUDIT-exclusive SQL statement commits or rolls back.
- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.

Example

Create an audit policy to audit successes and failures for the AUDIT and OBJMAINT categories; only failures for the SECMAINT, CHECKING, and VALIDATE categories, and no events for the other categories.

```
CREATE AUDIT POLICY DBAUDPRF
  CATEGORIES AUDIT STATUS BOTH,
             SECMAINT STATUS FAILURE,
             OBJMAINT STATUS BOTH,
             CHECKING STATUS FAILURE,
             VALIDATE STATUS FAILURE
  ERROR TYPE NORMAL
```

CREATE DATABASE PARTITION GROUP

The CREATE DATABASE PARTITION GROUP statement defines a new database partition group within the database, assigns database partitions to the database partition group, and records the database partition group definition in the system catalog.

Invocation

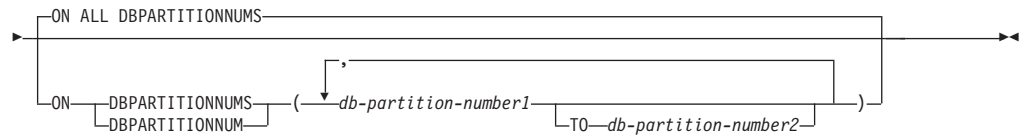
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax

```
►►—CREATE DATABASE PARTITION GROUP—db-partition-group-name—►►
```



Description

db-partition-group-name

Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *db-partition-group-name* must not identify a database partition group that already exists in the catalog (SQLSTATE 42710). The *db-partition-group-name* must not begin with the characters 'SYS' or 'IBM' (SQLSTATE 42939).

ON ALL DBPARTITIONNUMS

Specifies that the database partition group is defined over all database partitions defined to the database (db2nodes.cfg file) at the time the database partition group is created.

If a database partition is added to the database system, the ALTER DATABASE PARTITION GROUP statement should be issued to include this new database partition in a database partition group (including IBMDEFAULTGROUP). Furthermore, the REDISTRIBUTE DATABASE PARTITION GROUP command must be issued to move data to the database partition.

ON DBPARTITIONNUMS

Specifies the database partitions that are in the database partition group. DBPARTITIONNUM is a synonym for DBPARTITIONNUMS.

db-partition-number1

Specify a database partition number. (A *node-name* of the form NODEnnnnn can be specified for compatibility with the previous version.)

TO *db-partition-number2*

Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9). All database partitions between and including the specified database partition numbers are included in the database partition group.

Rules

- Each database partition specified by number must be defined in the db2nodes.cfg file (SQLSTATE 42729).
- Each *db-partition-number* listed in the ON DBPARTITIONNUMS clause must be appear at most once (SQLSTATE 42728).
- A valid *db-partition-number* is between 0 and 999 inclusive (SQLSTATE 42729).
- The CREATE DATABASE PARTITION GROUP statement might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This statement might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Notes

- This statement creates a distribution map for the database partition group. A distribution map identifier (PMAP_ID) is generated for each distribution map. This information is recorded in the catalog and can be retrieved from SYSCAT.DBPARTITIONGROUPS and SYSCAT.PARTITIONMAPS. Each entry in

the distribution map specifies the target database partition on which all rows that are hashed reside. For a single-partition database partition group, the corresponding distribution map has only one entry. For a multiple partition database partition group, the corresponding distribution map has 32768 entries, where the database partition numbers are assigned to the map entries in a round-robin fashion, by default.

- **Compatibilities:** For compatibility with previous versions of DB2 products:
 - NODE can be specified in place of DBPARTITIONNUM
 - NODES can be specified in place of DBPARTITIONNUMS
 - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Examples

Assume that you have a partitioned database with six database partitions defined as 0, 1, 2, 5, 7, and 8.

- Assume that you want to create a database partition group called MAXGROUP on all six database partitions. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MAXGROUP ON ALL DBPARTITIONNUMS
```

- Assume that you want to create a database partition group called MEDGROUP on database partitions 0, 1, 2, 5, and 8. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MEDGROUP  
ON DBPARTITIONNUMS( 0 TO 2, 5, 8)
```

- Assume that you want to create a single-partition database partition group MINGROUP on database partition 7. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MINGROUP  
ON DBPARTITIONNUM (7)
```

CREATE FUNCTION

The CREATE FUNCTION statement is used to register or define a user-defined function or function template at the current server.

There are five different types of functions that can be created using this statement. Each of these is described separately.

- **External Scalar.** The function is written in a programming language and returns a scalar value. The external executable is registered in the database, along with various attributes of the function.
- **External Table.** The function is written in a programming language and returns a complete table. The external executable is registered in the database along with various attributes of the function.
- **OLE DB External Table.** A user-defined OLE DB external table function is registered in the database to access data from an OLE DB provider.
- **Sourced or Template.** A source function is implemented by invoking another function (either built-in, external, SQL, or source) that is already registered in the database.

It is possible to create a partial function, called a *function template*, which defines what types of values are to be returned, but which contains no executable code. The user maps it to a data source function within a federated system, so that the data source function can be invoked from a federated database. A function template can be registered only with an application server that is designated as a federated server.

- SQL Scalar, Table or Row. The function body is written in SQL and defined together with the registration in the database. It returns a scalar value, a table, or a single row.

CREATE INDEX

The CREATE INDEX statement is used to:

- Define an index on a DB2 table. An index can be defined on XML data, or on relational data.
- Create an index specification (metadata that indicates to the optimizer that a data source table has an index)

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

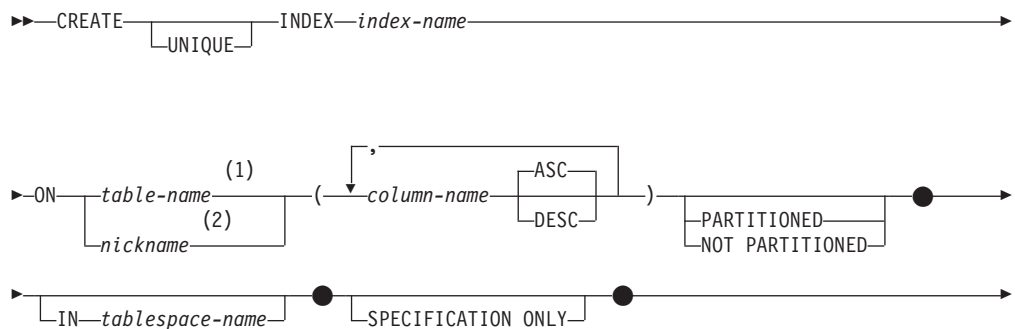
Authorization

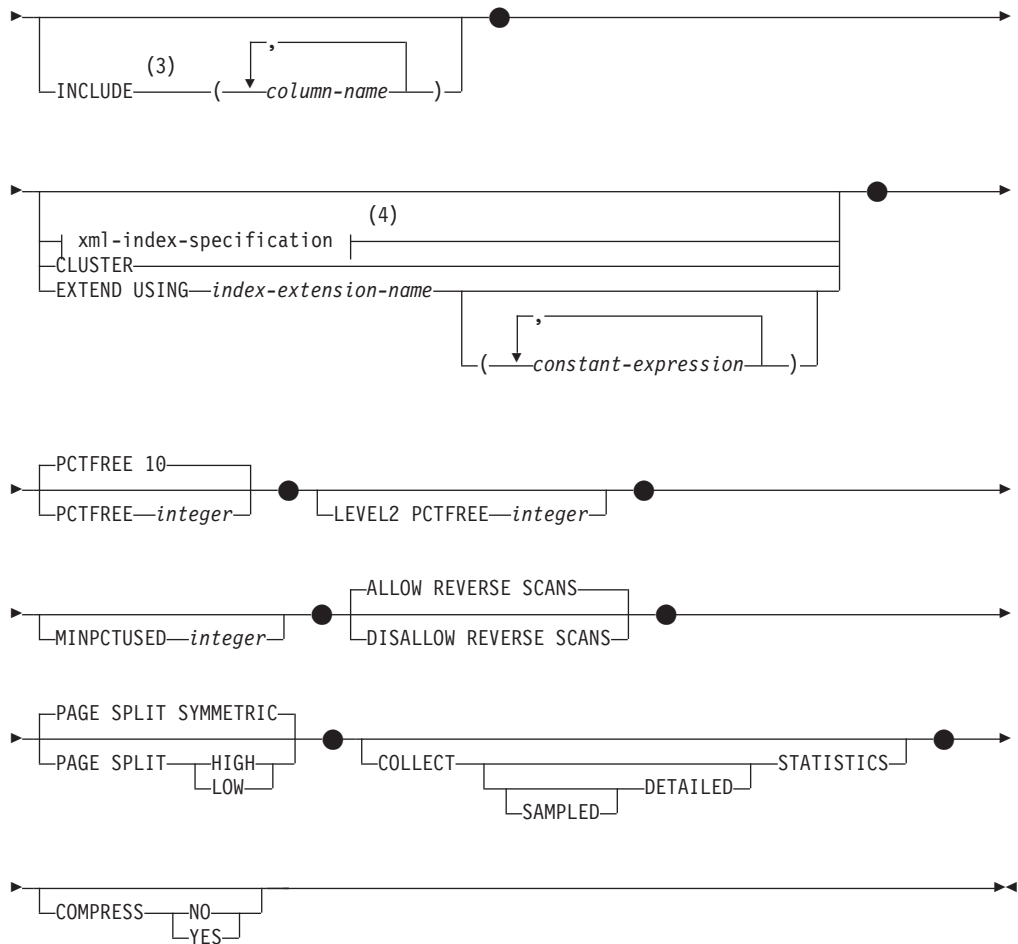
The privileges held by the authorization ID of the statement must include at least one of the following:

- One of:
 - CONTROL privilege on the table or nickname on which the index is defined
 - INDEX privilege on the table or nickname on which the index is defined
 and one of:
 - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the index does not exist
 - CREATEIN privilege on the schema, if the schema name of the index refers to an existing schema
- DBADM authority

No explicit privilege is required to create an index on a declared temporary table.

Syntax

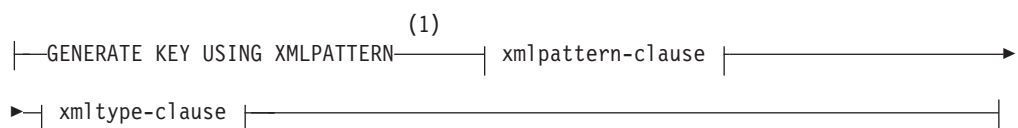




Notes:

- 1 In a federated system, *table-name* must identify a table in the federated database. It cannot identify a data source table.
- 2 If *nickname* is specified, the CREATE INDEX statement creates an index specification. In this case, INCLUDE, *xml-index-specification*, CLUSTER, EXTEND USING, PCTFREE, MINPCTUSED, DISALLOW REVERSE SCANS, ALLOW REVERSE SCANS, PAGE SPLIT, or COLLECT STATISTICS cannot be specified.
- 3 The INCLUDE clause can only be specified if UNIQUE is specified.
- 4 If *xml-index-specification* is specified, *column-name* DESC, INCLUDE, or CLUSTER cannot be specified.

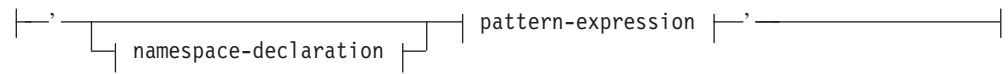
xml-index-specification:



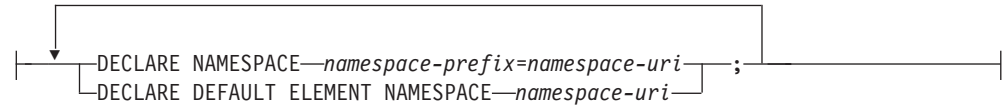
Notes:

1 The alternative syntax GENERATE KEYS USING XMLPATTERN can be used.

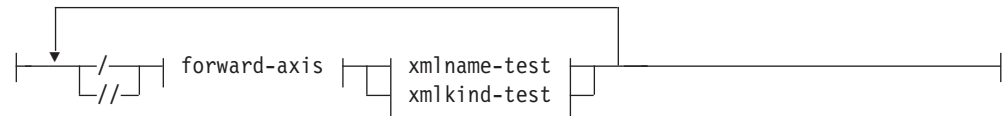
xmlpattern-clause:



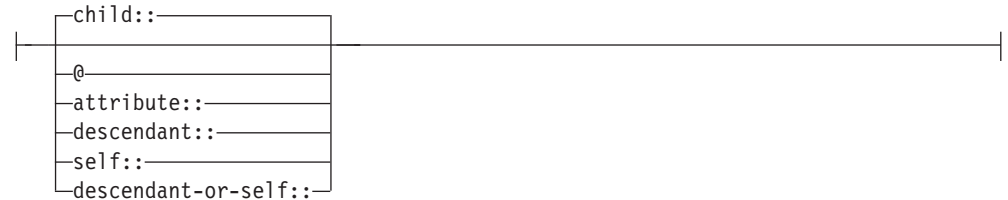
namespace-declaration:



pattern-expression:



forward-axis:



xmlname-test:



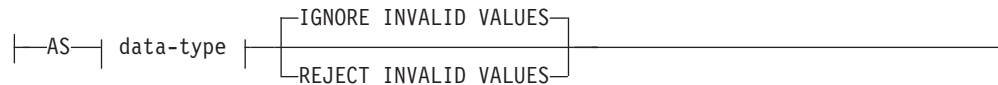
xml-wildcard:



xmlkind-test:



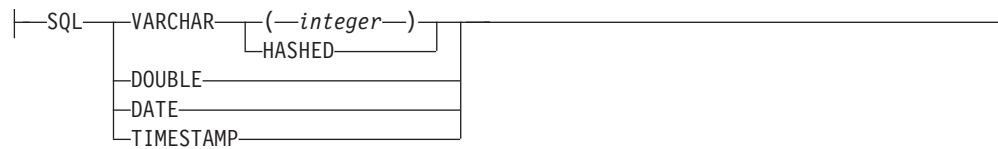
xmltype-clause:



data-type:



sql-data-type:



Description

UNIQUE

If `ON table-name` is specified, `UNIQUE` prevents the table from containing two or more rows with the same value of the index key. The uniqueness is enforced at the end of the `SQL` statement that updates rows or inserts new rows.

The uniqueness is also checked during the execution of the `CREATE INDEX` statement. If the table already contains rows with duplicate key values, the index is not created.

If the index is on an XML column (the index is an index over XML data), the uniqueness applies to values with the specified *pattern-expression* for all rows of the table. Uniqueness is enforced on each value after the value has been converted to the specified *sql-data-type*. Because converting to the specified *sql-data-type* might result in a loss of precision or range, or different values might be hashed to the same key value, multiple values that appear to be unique in the XML document might result in duplicate key errors. The uniqueness of character strings depends on XQuery semantics where trailing blanks are significant. Therefore, values that would be duplicates in `SQL` but differ in trailing blanks are considered unique values in an index over XML data.

When `UNIQUE` is used, null values are treated as any other values. For example, if the key is a single column that may contain null values, that column may contain no more than one null value.

If the `UNIQUE` option is specified, and the table has a distribution key, the columns in the index key must be a superset of the distribution key. That is, the columns specified for a unique index key must include all the columns of the distribution key (SQLSTATE 42997).

If the `UNIQUE` option is specified and the table has a table partitioning key, the columns of the index key must be a superset of the table partitioning key. That is, the columns specified for the unique index key must include all the columns of the table partitioning key (SQLSTATE 42990).

Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE).

If *ON nickname* is specified, UNIQUE should be specified only if the data for the index key contains unique values for every row of the data source table. The uniqueness will not be checked.

For an index over XML data, UNIQUE can be specified only if the specified *pattern-expression* specifies a single complete path and does not contain a descendant or descendant-or-self axis, *"/ /"*, an *xml-wildcard*, *node()*, or *processing-instruction()* (SQLSTATE 429BS).

In a partitioned database environment, the following rules apply to a table with one or more XML columns:

- A distributed table cannot have a unique index over XML data.
- A unique index over XML data is supported only on a table that does not have a distribution key and that is on a single node multi-partition database.
- If a unique index over XML data exists on a table, the table cannot be altered to add a distribution key.

INDEX *index-name*

Names the index or index specification. The name, including the implicit or explicit qualifier, must not identify an index or index specification that is described in the catalog, or an existing index on a declared temporary table (SQLSTATE 42704). The qualifier must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

The implicit or explicit qualifier for indexes on declared global temporary tables must be SESSION (SQLSTATE 428EK).

ON *table-name* **or** *nickname*

The *table-name* identifies a table on which an index is to be created. The table must be a base table (not a view), a created temporary table, a declared temporary table, a materialized query table that exists at the current server, or a declared temporary table. The name of a declared temporary table must be qualified with SESSION. The *table-name* must not identify a catalog table (SQLSTATE 42832). If UNIQUE is specified and *table-name* is a typed table, it must not be a subtable (SQLSTATE 429B3).

nickname is the nickname on which an index specification is to be created. The *nickname* references either a data source table whose index is described by the index specification, or a data source view that is based on such a table. The *nickname* must be listed in the catalog.

column-name

For an index, *column-name* identifies a column that is to be part of the index key. For an index specification, *column-name* is the name by which the federated server references a column of a data source table.

Each *column-name* must be an unqualified name that identifies a column of the table. Up to 64 columns can be specified. If *table-name* is a typed table, up to 63 columns can be specified. If *table-name* is a subtable, at least one *column-name* must be introduced in the subtable; that is, not inherited from a supertable (SQLSTATE 428DS). No *column-name* can be repeated (SQLSTATE 42711).

The sum of the stored lengths of the specified columns must not be greater than the index key length limit for the page size. For key length limits, see "SQL limits". If *table-name* is a typed table, the index key length limit is further reduced by 4 bytes. Note that this length limit can be reduced even more by system overhead, which varies according to the data type of the column and whether or not the column is nullable. For more information on overhead affecting this limit, see "Byte Counts" in "CREATE TABLE".

Note that this length can be reduced by system overhead, which varies according to the data type of the column and whether it is nullable. For more information on overhead affecting this limit, see “Byte Counts” in “CREATE TABLE”.

No LOB column or distinct type column based on a LOB can be used as part of an index, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008). A structured type column can only be specified if the EXTEND USING clause is also specified (SQLSTATE 42962). If the EXTEND USING clause is specified, only one column can be specified, and the type of the column must be a structured type or a distinct type that is not based on a LOB (SQLSTATE 42997).

If an index has only one column, and that column has the XML data type, and the GENERATE KEY USING XMLPATTERN clause is also specified, the index is an index over XML data. A column with the XML data type can be specified only if the GENERATE KEY USING XMLPATTERN clause is also specified (SQLSTATE 42962). If the GENERATE KEY USING XMLPATTERN clause is specified, only one column can be specified, and the type of the column must be XML.

ASC

Specifies that index entries are to be kept in ascending order of the column values; this is the default setting. ASC cannot be specified for indexes that are defined with EXTEND USING (SQLSTATE 42601).

DESC

Specifies that index entries are to be kept in descending order of the column values. DESC cannot be specified for indexes that are defined with EXTEND USING, or if the index is an index over XML data (SQLSTATE 42601).

PARTITIONED

Indicates that a partitioned index should be created. The *table-name* must identify a table defined with data partitions (SQLSTATE 42601).

If the table is partitioned and neither PARTITIONED nor NOT PARTITIONED is specified, the index is created as partitioned (with a few exceptions). A nonpartitioned index is created instead of partitioned index if any of the following situations apply:

- UNIQUE is specified and the index key does not include all the table partitioning key columns.
- A spatial index is created.
- The index is defined over XML data.

A partitioned index with a definition that duplicates the definition of a nonpartitioned index is not considered to be a duplicate index. For more details, see the “Rules” on page 115 section in this topic.

The PARTITIONED keyword cannot be specified for the following indexes:

- An index on a nonpartitioned table (SQLSTATE 42601)
- An index defined over XML data (SQLSTATE 42613)
- A unique index where the index key does not include all the table partitioning key columns (SQLSTATE 42990)
- A spatial index (SQLSTATE 42997)

A partitioned index cannot be created on a partitioned table that has detached dependent tables, for example, MQTs (SQLSTATE 55019).

The table space placement for an index partition of the partitioned index is determined by the following rules:

- If the table being indexed was created using the partition-tablespace-options INDEX IN clause of the CREATE TABLE statement, the index partition is created in the table space specified in that INDEX IN clause.
- If the CREATE TABLE statement for the table being indexed did not specify the partition-tablespace-options INDEX IN clause, the index partition of the partitioned index is created in the same table space as the corresponding data partition that it indexes.

The IN clause of the CREATE INDEX statement is not supported for partitioned indexes (SQLSTATE 42601). The tablespace-clauses INDEX IN clause of the CREATE TABLE statement is ignored for partitioned indexes.

NOT PARTITIONED

Indicates that a nonpartitioned index should be created that spans all of the data partitions defined for the table. The *table-name* must identify a table defined with data partitions (SQLSTATE 42601).

A nonpartitioned index with a definition that duplicates the definition of a partitioned index is not considered to be a duplicate index. For more details, see the “Rules” on page 115 section in this topic.

The table space placement for a the nonpartitioned index is determined by the following rules:

- If you specify the IN clause of the CREATE INDEX statement, the nonpartitioned index is placed in the table space specified in that IN clause.
- If you do not specify the IN clause of the CREATE INDEX statement, the following rules determine the table space placement of the nonpartitioned index:
 - If the table being indexed was created using the tablespace-clauses INDEX IN clause of the CREATE TABLE statement, the nonpartitioned index is placed in the table space specified in that INDEX IN clause.
 - If the table being indexed was created without using the tablespace-clauses INDEX IN clause of the CREATE TABLE statement, the nonpartitioned index is created in the table space of the first visible or attached data partition of the table. The first visible or attached data partition of the table is the first partition in the list of data partitions that are sorted on the basis of range specifications. Also, the authorization ID of the statement is not required to have the USE privilege on the default table space.

IN *tablespace-name*

The IN clause is supported only for nonpartitioned indexes. Specifying the IN clause for partitioned indexes results in SQLSTATE 42601.

Specifies the table space in which the index is to be created. This clause is not supported for indexes on a created temporary table or a declared temporary table (SQLSTATE 42601). You can specify this clause even if the INDEX IN clause was specified when the table was created. This will override that clause.

The table space specified by *tablespace-name* must be in the same database partition group as the data table spaces for the table and manage space in the same way as the other table spaces of the partitioned table (SQLSTATE 42838); it must be a table space on which the authorization ID of the statement holds the USE privilege.

If the IN clause is not specified, the index is created in the table space that was specified by the INDEX IN clause on the CREATE TABLE statement. If no INDEX IN clause was specified, the table space of the first visible or attached data partition of the table is used. This is the first partition in the list of data partitions that are sorted on the basis of range specifications. If the IN clause is not specified, the authorization ID of the statement is not required to have the USE privilege on the default table space.

SPECIFICATION ONLY

Indicates that this statement will be used to create an index specification that applies to the data source table referenced by *nickname*. SPECIFICATION ONLY must be specified if *nickname* is specified (SQLSTATE 42601). It cannot be specified if *table-name* is specified (SQLSTATE 42601).

If the index specification applies to an index that is unique, DB2 does not verify that the column values in the remote table are unique. If the remote column values are not unique, queries against the nickname that include the index column might return incorrect data or errors.

This clause cannot be used when creating an index on a created temporary table or declared temporary table (SQLSTATE 42995).

INCLUDE

This keyword introduces a clause that specifies additional columns to be appended to the set of index key columns. Any columns included with this clause are not used to enforce uniqueness. These included columns might improve the performance of some queries through index only access. The columns must be distinct from the columns used to enforce uniqueness (SQLSTATE 42711). UNIQUE must be specified when INCLUDE is specified (SQLSTATE 42613). The limits for the number of columns and sum of the length attributes apply to all of the columns in the unique key and in the index.

This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

column-name

Identifies a column that is included in the index but not part of the unique index key. The same rules apply as defined for columns of the unique index key. The keywords ASC or DESC may be specified following the column-name but have no effect on the order.

INCLUDE cannot be specified for indexes that are defined with EXTEND USING, if *nickname* is specified, or if the index is an XML values index (SQLSTATE 42601).

xml-index-specification

Specifies how index keys are generated from XML documents that are stored in an XML column. *xml-index-specification* cannot be specified if there is more than one index column, or if the column does not have the XML data type.

This clause only applies to XML columns (SQLSTATE 429BS).

GENERATE KEY USING XMLPATTERN *xmlpattern-clause*

Specifies the parts of an XML document that are to be indexed. XML pattern values are the indexed values generated by the *xmlpattern-clause*. List data type nodes are not supported in the index. If a node is qualified by the *xmlpattern-clause* and an XML schema exists that specifies that the

node is a list data type, then the list data type node cannot be indexed (SQLSTATE 23526 for CREATE INDEX statements, or SQLSTATE 23525 for INSERT and UPDATE statements).

xmlpattern-clause

Contains a pattern expression that identifies the nodes that are to be indexed. It consists of an optional *namespace-declaration* and a required *pattern-expression*.

namespace-declaration

If the pattern expression contains qualified names, a *namespace-declaration* must be specified to define namespace prefixes. A default namespace can be defined for unqualified names.

DECLARE NAMESPACE *namespace-prefix=namespace-uri*

Maps *namespace-prefix*, which is an NCName, to *namespace-uri*, which is a string literal. The *namespace-declaration* can contain multiple *namespace-prefix-to-namespace-uri* mappings. The *namespace-prefix* must be unique within the list of *namespace-declaration* (SQLSTATE 10503).

DECLARE DEFAULT ELEMENT NAMESPACE *namespace-uri*

Declares the default namespace URI for unqualified element names or types. If no default namespace is declared, unqualified names of elements and types are in no namespace. Only one default namespace can be declared (SQLSTATE 10502).

pattern-expression

Specifies the nodes in an XML document that are indexed. The *pattern-expression* can contain pattern-matching characters (*). It is similar to a path expression in XQuery, but supports a subset of the XQuery language that is supported by DB2.

/ (forward slash)

Separates path expression steps.

// (double forward slash)

This is the abbreviated syntax for */descendant-or-self::node()/*. You cannot use *// (double forward slash)* if you also specify UNIQUE.

forward-axis

child::

Specifies children of the context node. This is the default, if no other forward axis is specified.

@ Specifies attributes of the context node. This is the abbreviated syntax for *attribute::*.

attribute::

Specifies attributes of the context node.

descendant::

Specifies the descendants of the context node. You cannot use *descendant::* if you also specify UNIQUE.

self::

Specifies just the context node itself.

descendant-or-self::

Specifies the context node and the descendants of the context node. You cannot use `descendant-or-self::` if you also specify `UNIQUE`.

xmlname-test

Specifies the node name for the step in the path using a qualified XML name (`xml-qname`) or a wildcard (`xml-wildcard`).

xml-ncname

An XML name as defined by XML 1.0. It cannot include a colon character.

xml-qname

Specifies a qualified XML name (also known as a `QName`) that can have two possible forms:

- `xml-namespace:xml-ncname`, where the `xml-namespace` is an `xml-ncname` that identifies an in-scope namespace
- `xml-ncname`, which indicates that the default namespace should be applied as the implicit `xml-namespace`

xml-wildcard

Specifies an `xml-qname` as a wildcard that can have three possible forms:

- `*` (a single asterisk character) indicates any `xml-qname`
- `xml-namespace:*` indicates any `xml-ncname` within the specified namespace
- `*:xml-ncname` indicates a specific XML name in any in-scope namespace

You cannot use *xml-wildcard* if you also specify `UNIQUE`.

xmlkind-test

Use these options to specify what types of nodes you pattern match. The following options are available to you:

node()

Matches any node. You cannot use `node()` if you also specify `UNIQUE`.

text()

Matches any text node.

comment()

Matches any comment node.

processing-instruction()

Matches any processing instruction node. You cannot use `processing-instruction()` if you also specify `UNIQUE`.

*xmltype-clause***AS data-type**

Specifies the data type to which indexed values are converted before they are stored. Values are converted to the index XML data type that corresponds to the specified index SQL data type.

Table 7. Corresponding index data types

Index XML data type	Index SQL data type
xs:string	VARCHAR(<i>integer</i>), VARCHAR HASHED
xs:double	DOUBLE
xs:date	DATE
xs:dateTime	TIMESTAMP

For VARCHAR(*integer*) and VARCHAR HASHED, the value is converted to an xs:string value using the XQuery function fn:string. The length attribute of VARCHAR(*integer*) is applied as a constraint to the resulting xs:string value. An index SQL data type of VARCHAR HASHED applies a hash algorithm to the resulting xs:string value to generate a hash code that is inserted into the index.

For indexes using the data types DOUBLE, DATE, and TIMESTAMP, the value is converted to the index XML data type using the XQuery cast expression.

If the index is unique, the uniqueness of the value is enforced after the value is converted to the indexed type.

data-type

The following data type is supported:

sql-data-type

Supported SQL data types are:

VARCHAR(*integer*)

If this form of VARCHAR is specified, DB2 uses *integer* as a constraint. If document nodes that are to be indexed have values that are longer than *integer*, the documents are not inserted into the table if the index already exists. If the index does not exist, the index is not created. *integer* is a value between 1 and a page size-dependent maximum. Table 8 shows the maximum value for each page size.

Table 8. Maximum length of document nodes by page size

Page size	Maximum length of document node (bytes)
4KB	817
8KB	1841
16KB	3889
32KB	7985

XQuery semantics are used for string comparisons, where trailing blanks are significant. This differs from SQL semantics, where trailing blanks are insignificant during comparisons.

VARCHAR HASHED

Specify VARCHAR HASHED to handle indexing of arbitrary length character strings. The length of an indexed string has no limit. DB2 generates an eight-byte hash code over the entire string. Indexes that

use these hashed character strings can be used only for equality lookups. XQuery semantics are used for string equality comparisons, where trailing blanks are significant. This differs from SQL semantics, where trailing blanks are insignificant during comparisons. The hash on the string preserves XQuery semantics for equality and not SQL semantics.

DOUBLE

Specifies that the data type DOUBLE is used for indexing numeric values. Unbounded decimal types and 64 bit integers may lose precision when they are stored as a DOUBLE value. The values for DOUBLE may include the special numeric values *NaN*, *INF*, *-INF*, *+0*, and *-0*, even though the SQL data type DOUBLE itself does not support these values.

DATE

Specifies that the data type DATE is used for indexing XML values. Note that the XML schema data type for `xs:date` allows greater range of values than the DB2 pureXML `xs:date` data type that corresponds to the SQL data type. If an out-of-range value is encountered, an error is returned.

TIMESTAMP

Specifies that the data type TIMESTAMP is used for indexing XML values. Note that the XML schema data type for `xs:dateTime` allows greater range of values and fractional seconds precision than the DB2 pureXML `xs:dateTime` data type that corresponds to the SQL data type. If an out-of-range value is encountered, an error is returned.

IGNORE INVALID VALUES

Specifies that XML pattern values that are invalid for the target index XML data type are ignored and that the corresponding values in the stored XML documents are not indexed by the CREATE INDEX statement. By default, invalid values are ignored. During insert and update operations, the invalid XML pattern values are not indexed, but XML documents are still inserted into the table. No error or warning is raised, because specifying these data types is not a constraint on the XML pattern values (XQuery expressions that search for the specific XML index data type will not consider these values).

The index can ignore only invalid XML pattern values for the index XML data type. Valid values must conform to the DB2 representation of the value for the index XML data type, or an error is returned. An XML pattern value associated with the index XML data type `xs:string` is always valid. However, the additional length constraint of the associated index SQL data type `VARCHAR(integer)` data type can still raise an error, if the maximum length is exceeded. If an error is returned, XML data is not inserted or updated in the table if the index already exists (SQLSTATE 23525). If the index does not exist, the index is not created (SQLSTATE 23526).

REJECT INVALID VALUES

Specifies that all XML pattern values must be valid for the index XML data type. If any XML pattern value cannot be cast to the index XML data type, an error is returned. XML data is not inserted or updated in the table if the index already exists (SQLSTATE 23525). If the index does not exist, the index is not created (SQLSTATE 23526).

CLUSTER

Specifies that the index is the clustering index of the table. The cluster factor of a clustering index is maintained or improved dynamically as data is inserted into the associated table, by attempting to insert new rows physically close to the rows for which the key values of this index are in the same range. Only one clustering index may exist for a table so CLUSTER may not be specified if it was used in the definition of any existing index on the table (SQLSTATE 55012). A clustering index may not be created on a table that is defined to use append mode (SQLSTATE 428D8).

CLUSTER is disallowed if *nickname* is specified, or if the index is an index over XML data (SQLSTATE 42601). This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995) or range-clustered tables (SQLSTATE 429BG).

EXTEND USING *index-extension-name*

Names the *index-extension* used to manage this index. If this clause is specified, then there must be only one *column-name* specified and that column must be a structured type or a distinct type (SQLSTATE 42997). The *index-extension-name* must name an index extension described in the catalog (SQLSTATE 42704). For a distinct type, the column must exactly match the type of the corresponding source key parameter in the index extension. For a structured type column, the type of the corresponding source key parameter must be the same type or a supertype of the column type (SQLSTATE 428E0).

This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

constant-expression

Identifies values for any required arguments for the index extension. Each expression must be a constant value with a data type that exactly matches the defined data type of the corresponding index extension parameters, including length or precision, and scale (SQLSTATE 428E0). This clause must not exceed 32 768 bytes in length in the database code page (SQLSTATE 22001).

PCTFREE *integer*

Specifies what percentage of each index page to leave as free space when building the index. The first entry in a page is added without restriction. When additional entries are placed in an index page at least *integer* percent of free space is left on each page. The value of *integer* can range from 0 to 99. If a value greater than 10 is specified, only 10 percent free space will be left in non-leaf pages. The default is 10.

PCTFREE is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

LEVEL2 PCTFREE *integer*

Specifies what percentage of each index level 2 page to leave as free space when building the index. The value of *integer* can range from 0 to 99. If LEVEL2 PCTFREE is not set, a minimum of 10 or PCTFREE percent of free

space is left on all non-leaf pages. If LEVEL2 PCTFREE is set, *integer* percent of free space is left on level 2 intermediate pages, and a minimum of 10 or *integer* percent of free space is left on level 3 and higher intermediate pages.

LEVEL2 PCTFREE is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

MINPCTUSED *integer*

Indicates whether index leaf pages are merged online, and the threshold for the minimum percentage of space used on an index leaf page. If, after a key is removed from an index leaf page, the percentage of space used on the page is at or below *integer* percent, an attempt is made to merge the remaining keys on this page with those of a neighboring page. If there is sufficient space on one of these pages, the merge is performed and one of the pages is deleted. The value of *integer* can be from 0 to 99. A value of 50 or below is recommended for performance reasons. Specifying this option will have an impact on update and delete performance. Merging is only done during update and delete operations when an exclusive table lock is held. If an exclusive table lock does not exist, keys are marked as pseudo deleted during update and delete operations, and no merging is done. Consider using the CLEANUP ONLY ALL option of REORG INDEXES to merge leaf pages instead of using the MINPCTUSED option of CREATE INDEX.

MINPCTUSED is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

DISALLOW REVERSE SCANS

Specifies that an index only supports forward scans or scanning of the index in the order that was defined at index creation time.

DISALLOW REVERSE SCANS cannot be specified together with *nickname* (SQLSTATE 42601).

ALLOW REVERSE SCANS

Specifies that an index can support both forward and reverse scans; that is, scanning of the index in the order that was defined at index creation time, and scanning in the opposite order.

ALLOW REVERSE SCANS cannot be specified together with *nickname* (SQLSTATE 42601).

PAGE SPLIT

Specifies an index split behavior. The default is SYMMETRIC.

SYMMETRIC

Specifies that pages are to be split roughly in the middle.

HIGH

Specifies an index page split behavior that uses the space on index pages efficiently when the values of the index keys being inserted follow a particular pattern. For a subset of index key values, the leftmost column or columns of the index must contain the same value, and the rightmost column or columns of the index must contain values that increase with each insertion. For details, see "Options on the CREATE INDEX statement".

LOW

Specifies an index page split behavior that uses the space on index pages efficiently when the values of the index keys being inserted follow a

particular pattern. For a subset of index key values, the leftmost column or columns of the index must contain the same value, and the rightmost column or columns of the index must contain values that decrease with each insertion. For details, see “Options on the CREATE INDEX statement”.

COLLECT STATISTICS

Specifies that basic index statistics are to be collected during index creation.

DETAILED

Specifies that extended index statistics (CLUSTERFACTOR and PAGE_FETCH_PAIRS) are also to be collected during index creation.

SAMPLED

Specifies that sampling can be used when compiling extended index statistics.

COMPRESS

Specifies whether index compression is enabled. By default, index compression will be enabled if data row compression is enabled; index compression will be disabled if data row compression is disabled. This option can be used to override the default behavior. COMPRESS is disallowed if *nickname* is specified (SQLSTATE 42601).

YES

Specifies that index compression is enabled. Insert and update operations on the index will be subject to compression.

NO

Specifies that index compression is disabled.

Rules

- The CREATE INDEX statement fails (SQLSTATE 01550) when attempting to create an index that matches an existing index.

A number of factors are used to determine if two indexes match. These factors are combined in various different ways into the rules that determine if two indexes match. The following factors are used to determine if two indexes match:

1. The sets of index columns, including any INCLUDE columns, are the same in both indexes.
2. The ordering of index key columns, including any INCLUDE columns, is the same in both indexes.
3. The key columns of the new index are the same or a superset of the key columns in the existing index.
4. The ordering attributes of the columns are the same in both indexes.
5. The existing index is unique.
6. Both indexes are non-unique.

The following combinations of these factors form the rules that determine when two indexes are considered duplicates:

- 1 + 2 + 4 + 5
- 1 + 2 + 4 + 6
- 1 + 2 + 3 + 5

Exceptions:

- If one of the compared indexes is partitioned and the other of the compared indexes is nonpartitioned, the indexes are not considered duplicates if the indexes have different names, even if other matching index conditions are met.
- For indexes over XML data, the index descriptions are not considered duplicates if the index names are different, even if the indexed XML column, the XML patterns, and the data type, including its options, are identical.
- Unique indexes on system-maintained MQTs are not supported (SQLSTATE 42809).
- The COLLECT STATISTICS options are not supported if a nickname is specified (SQLSTATE 42601).

Notes

- Concurrent read/write access to the table is permitted while an index is being created. However, the default index creation behavior differs for indexes on nonpartitioned tables, nonpartitioned indexes, and partitioned indexes:
 - For indexes on nonpartitioned tables, once the index has been built, changes that were made to the table during index creation time are forward-fitted to the new index. Write access to the table is then briefly blocked while index creation completes, after which the new index becomes available.
 - For nonpartitioned indexes, once the index has been built, changes that were made to the table during index creation time are forward-fitted to the new index. Write access to the table is then briefly blocked while index creation completes, after which the new index becomes available.
 - For partitioned indexes, once the index partition has been built, changes that were made to the partition during creation time of that index partition are forward-fitted to the new index partition. Write access to the data partition is then blocked while index creation completes on the remaining data partitions. After the index partition for the last data partition is built and the transaction is committed, all data partitions are available for read and write.

To circumvent this default behavior, use the LOCK TABLE statement to explicitly lock the table before issuing a CREATE INDEX statement. (The table can be locked in either SHARE or EXCLUSIVE mode, depending on whether read access is to be allowed.)

- If the named table already contains data, CREATE INDEX creates the index entries for it. If the table does not yet contain data, CREATE INDEX creates a description of the index; the index entries are created when data is inserted into the table.
- Once the index is created and data is loaded into the table, it is advisable to issue the RUNSTATS command. The RUNSTATS command updates statistics collected on the database tables, columns, and indexes. These statistics are used to determine the optimal access path to the tables. By issuing the RUNSTATS command, the database manager can determine the characteristics of the new index. If data has been loaded before the CREATE INDEX statement is issued, it is recommended that the COLLECT STATISTICS option on the CREATE INDEX statement be used as an alternative to the RUNSTATS command.
- Creating an index with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- The optimizer can recommend indexes prior to creating the actual index.

- If an index specification is being defined for a data source table that has an index, the name of the index specification does not have to match the name of the index.
- The optimizer uses index specifications to improve access to the data source tables that the specifications apply to.
- **Compatibilities:** For compatibility with DB2 for z/OS:
 - The following syntax is tolerated and ignored:
 - CLOSE
 - DEFINE
 - FREEPAGE
 - GBPCACHE
 - PIECESIZE
 - TYPE 2
 - using-block
 - The following syntax is accepted as the default behavior:
 - COPY NO
 - DEFER NO

Examples

Example 1: Create an index named UNIQUE_NAM on the PROJECT table. The purpose of the index is to ensure that there are not two entries in the table with the same value for project name (PROJNAME). The index entries are to be in ascending order.

```
CREATE UNIQUE INDEX UNIQUE_NAM
ON PROJECT(PROJNAME)
```

Example 2: Create an index named JOB_BY_DPT on the EMPLOYEE table. Arrange the index entries in ascending order by job title (JOB) within each department (WORKDEPT).

```
CREATE INDEX JOB_BY_DPT
ON EMPLOYEE (WORKDEPT, JOB)
```

Example 3: The nickname EMPLOYEE references a data source table called CURRENT_EMP. After this nickname was created, an index was defined on CURRENT_EMP. The columns chosen for the index key were WORKDEBT and JOB. Create an index specification that describes this index. Through this specification, the optimizer will know that the index exists and what its key is. With this information, the optimizer can improve its strategy to access the table.

```
CREATE UNIQUE INDEX JOB_BY_DEPT
ON EMPLOYEE (WORKDEPT, JOB)
SPECIFICATION ONLY
```

Example 4: Create an extended index type named SPATIAL_INDEX on a structured type column location. The description in index extension GRID_EXTENSION is used to maintain SPATIAL_INDEX. The literal is given to GRID_EXTENSION to create the index grid size.

```
CREATE INDEX SPATIAL_INDEX ON CUSTOMER (LOCATION)
EXTEND USING (GRID_EXTENSION (x'000100100010001000400010'))
```

Example 5: Create an index named IDX1 on a table named TAB1, and collect basic index statistics on index IDX1.

```
CREATE INDEX IDX1 ON TAB1 (col1) COLLECT STATISTICS
```

Example 6: Create an index named IDX2 on a table named TAB1, and collect detailed index statistics on index IDX2.

```
CREATE INDEX IDX2 ON TAB1 (col2) COLLECT DETAILED STATISTICS
```

Example 7: Create an index named IDX3 on a table named TAB1, and collect detailed index statistics on index IDX3 using sampling.

```
CREATE INDEX IDX3 ON TAB1 (col3) COLLECT SAMPLED DETAILED STATISTICS
```

Example 8: Create a unique index named A_IDX on a partitioned table named MYNUMBERDATA in table space IDX_TBSP.

```
CREATE UNIQUE INDEX A_IDX ON MYNUMBERDATA (A) IN IDX_TBSP
```

Example 9: Create a non-unique index named B_IDX on a partitioned table named MYNUMBERDATA in table space IDX_TBSP.

```
CREATE INDEX B_IDX ON MYNUMBERDATA (B)  
NOT PARTITIONED IN IDX_TBSP
```

Example 10: Create an index over XML data on a table named COMPANYINFO, which contains an XML column named COMPANYDOCS. The XML column COMPANYDOCS contains a large number of XML documents similar to the one below:

```
<company name="Company1">  
  <emp id="31201" salary="60000" gender="Female">  
    <name>  
      <first>Laura</first>  
      <last>Brown</last>  
    </name>  
    <dept id="M25">  
      Finance  
    </dept>  
  </emp>  
</company>
```

Users of the COMPANYINFO table often need to retrieve employee information using the employee ID. An index like the following one can make that retrieval more efficient.

```
CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)  
GENERATE KEY USING XMLPATTERN '/company/emp/@id'  
AS SQL DOUBLE
```

Example 11: The following index is logically equivalent to the index created in the previous example, except that it uses unabbreviated syntax.

```
CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)  
GENERATE KEY USING XMLPATTERN '/child::company/child::emp/attribute::id'  
AS SQL DOUBLE
```

Example 12: Create an index on a column named DOC, indexing only the book title as a VARCHAR(100). Because the book title should be unique across all books, the index must be unique.

```
CREATE UNIQUE INDEX MYDOCSIDX ON MYDOCS(DOC)  
GENERATE KEY USING XMLPATTERN '/book/title'  
AS SQL VARCHAR(100)
```

Example 13: Create an index on a column named DOC, indexing the chapter number as a DOUBLE. This example includes namespace declarations.

```

CREATE INDEX MYDOCSIDX ON MYDOCS(DOC)
GENERATE KEY USING XMLPATTERN
'declare namespace b="http://www.foobar.com/book/";
declare namespace c="http://acme.org/chapters";
/b:book/c:chapter/@number'
AS SQL DOUBLE

```

CREATE METHOD

The CREATE METHOD statement is used to associate a method body with a method specification that is already part of the definition of a user-defined structured type.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CREATEIN privilege on the schema of the structured type referred to in the CREATE METHOD statement
- The owner of the structured type referred to in the CREATE METHOD statement
- DBADM authority

To associate an external method body with its method specification, the privileges held by the authorization ID of the statement must also include at least one of the following:

- CREATE_EXTERNAL_ROUTINE authority on the database
- DBADM authority

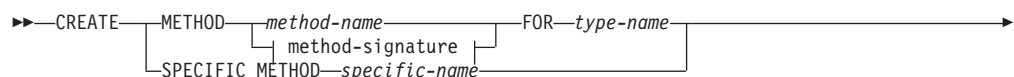
When creating an SQL method, the privileges held by the authorization ID of the statement must also include at least one of the following for each table, view, or nickname identified in any fullselect:

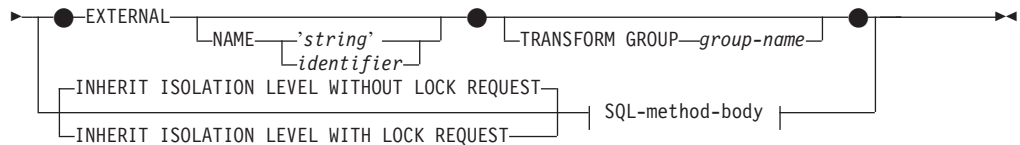
- CONTROL privilege on that table, view, or nickname
- SELECT privilege on that table, view, or nickname
- DATAACCESS authority

Group privileges other than PUBLIC are not considered for any table or view specified in the CREATE METHOD statement.

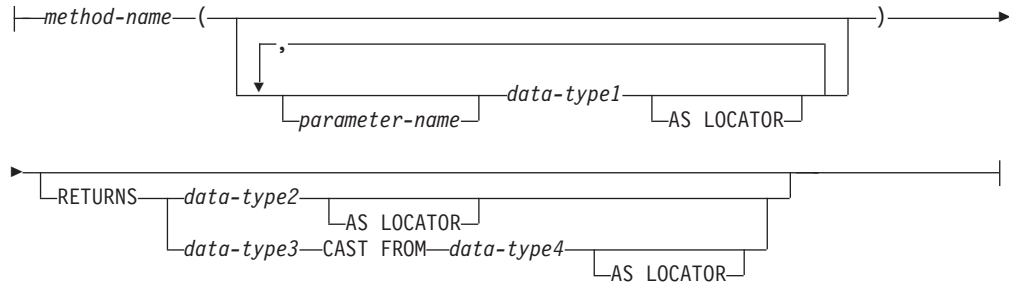
Authorization requirements of the data source for the table or view referenced by the nickname are applied when the method is invoked. The authorization ID of the connection can be mapped to a different remote authorization ID.

Syntax





method-signature:



SQL-method-body:



Notes:

- | 1 The compound SQL (inlined) statement is only supported for an
- | SQL-method-body in an SQL method definition in a non-partitioned database.

Description

METHOD

Identifies an existing method specification that is associated with a user-defined structured type. The method-specification can be identified through one of the following means:

method-name

Names the method specification for which a method body is being defined. The implicit schema is the schema of the subject type (*type-name*). There must be only one method specification for *type-name* that has this *method-name* (SQLSTATE 42725).

method-signature

Provides the method signature which uniquely identifies the method to be defined. The method signature must match the method specification that was provided on the CREATE TYPE or ALTER TYPE statement (SQLSTATE 42883).

method-name

Names the method specification for which a method body is being defined. The implicit schema is the schema of the subject type (*type-name*).

parameter-name

Identifies the parameter name. If parameter names are provided in the method signature, they must be exactly the same as the

corresponding parts of the matching method specification. Parameter names are supported in this statement solely for documentation purposes.

data-type1

Specifies the data type of each parameter. Array types are not supported (SQLSTATE 42815).

AS LOCATOR

For the LOB types or distinct types which are based on a LOB type, the AS LOCATOR clause can be added.

RETURNS

This clause identifies the output of the method. If a RETURNS clause is provided in the method signature, it must be exactly the same as the corresponding part of the matching method specification on CREATE TYPE. The RETURNS clause is supported in this statement solely for documentation purposes.

data-type2

Specifies the data type of the output. Array types are not supported (SQLSTATE 42815).

AS LOCATOR

For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be added. This indicates that a LOB locator is to be returned by the method instead of the actual value.

data-type3 **CAST FROM** *data-type4*

This form of the RETURNS clause is used to return a different data type to the invoking statement from the data type that was returned by the function code.

AS LOCATOR

For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be used to indicate that a LOB locator is to be returned from the method instead of the actual value.

FOR *type-name*

Names the type for which the specified method is to be associated. The name must identify a type already described in the catalog. (SQLSTATE 42704) In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

SPECIFIC METHOD *specific-name*

Identifies the particular method, using the specific name either specified or defaulted to at CREATE TYPE time. The specific-name must identify a method specification in the named or implicit schema; otherwise, an error is raised (SQLSTATE 42704).

EXTERNAL

This clause indicates that the CREATE METHOD statement is being used to register a method, based on code written in an external programming language, and adhering to the documented linkage conventions and interface. The matching method-specification in CREATE TYPE must specify a LANGUAGE other than SQL. When the method is invoked, the subject of the method is passed to the implementation as an implicit first parameter.

If the NAME clause is not specified, "NAME *method-name*" is assumed.

NAME

This clause identifies the name of the user-written code which implements the method being defined.

'string'

The 'string' option is a string constant with a maximum of 254 bytes. The format used for the string is dependent on the LANGUAGE specified. For more information on the specific language conventions, see "CREATE FUNCTION (External Scalar) statement".

identifier

This identifier specified is an SQL identifier. The SQL identifier is used as the library-id in the string. Unless it is a delimited identifier, the identifier is folded to upper case. If the identifier is qualified with a schema name, the schema name portion is ignored. This form of NAME can only be used with LANGUAGE C (as defined in the method-specification on CREATE TYPE).

TRANSFORM GROUP *group-name*

Indicates the transform group that is used for user-defined structured type transformations when invoking the method. A transform is required since the method definition includes a user-defined structured type.

It is strongly recommended that a transform group name be specified; if this clause is not specified, the default group-name used is DB2_FUNCTION. If the specified (or default) group-name is not defined for a referenced structured type, an error results (SQLSTATE 42741). Likewise, if a required FROM SQL or TO SQL transform function is not defined for the given group-name and structured type, an error results (SQLSTATE 42744).

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST or INHERIT ISOLATION LEVEL WITH LOCK REQUEST

Specifies whether or not a lock request can be associated with the isolation-clause of the statement when the method inherits the isolation level of the statement that invokes the method. The default is INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST.

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST

Specifies that, as the method inherits the isolation level of the invoking statement, it cannot be invoked in the context of an SQL statement which includes a lock-request-clause as part of a specified isolation-clause (SQLSTATE 42601).

INHERIT ISOLATION LEVEL WITH LOCK REQUEST

Specifies that, as the method inherits the isolation level of the invoking statement, it also inherits the specified lock-request-clause.

SQL-method-body

The SQL-method-body defines how the method is implemented if the method specification in CREATE TYPE is LANGUAGE SQL.

The SQL-method-body must comply with the following parts of method specification:

- DETERMINISTIC or NOT DETERMINISTIC (SQLSTATE 428C2)
- EXTERNAL ACTION or NO EXTERNAL ACTION (SQLSTATE 428C2)
- CONTAINS SQL or READS SQL DATA (SQLSTATE 42985)

Parameter names can be referenced in the SQL-method-body. The subject of the method is passed to the method implementation as an implicit first parameter named SELF.

For additional details, see “Compound SQL (inlined) statement” and “RETURN statement”.

Rules

- The method specification must be previously defined using the CREATE TYPE or ALTER TYPE statement before CREATE METHOD can be used (SQLSTATE 42723).
- If the method being created is an overriding method, those packages that are dependent on the following methods are invalidated:
 - The original method
 - Other overriding methods that have as their subject a supertype of the method being created
- The XML data type cannot be used in a method.

Notes

- If the method allows SQL, the external program must not attempt to access any federated objects (SQLSTATE 55047).
- *Privileges*: The definer of a method always receives the EXECUTE privilege on the method, as well as the right to drop the method.

If an EXTERNAL method is created, the definer of the method always receives the EXECUTE privilege WITH GRANT OPTION.

If an SQL method is created, the definer of the method will only be given the EXECUTE privilege WITH GRANT OPTION on the method when the definer has WITH GRANT OPTION on all privileges required to define the method, or if the definer has SYSADM or DBADM authority. The definer of an SQL method only acquires privileges if the privileges from which they are derived exist at the time the method is created. The definer must have these privileges either directly, or because PUBLIC has the privileges. Privileges held by groups of which the method definer is a member are not considered. When using the method, the connected user’s authorization ID must have the valid privileges on the table or view that the nickname references at the data source.

- *Table access restrictions*: If a method is defined as READS SQL DATA, no statement in the method can access a table that is being modified by the statement which invoked the method (SQLSTATE 57053).

Examples

Example 1:

```
CREATE METHOD BONUS (RATE DOUBLE)
FOR EMP
RETURN SELF..SALARY * RATE
```

Example 2:

```
CREATE METHOD SAMEZIP (addr address_t)
RETURNS INTEGER
FOR address_t
RETURN
(CASE
```



```

        WHEN (self..zip = addr..zip)
        THEN 1
        ELSE 0
    END)

```

Example 3:

```

CREATE METHOD DISTANCE (address_t)
FOR address_t
EXTERNAL NAME 'addresslib!distance'
TRANSFORM GROUP func_group

```

CREATE PROCEDURE (SQL)

The CREATE PROCEDURE (SQL) statement defines an SQL procedure at the current server.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

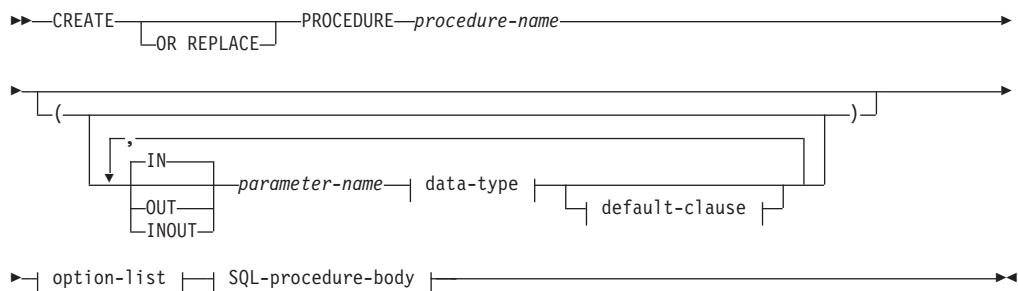
The privileges held by the authorization ID of the statement must include at least one of the following:

- If the implicit or explicit schema name of the procedure does not exist, IMPLICIT_SCHEMA authority on the database.
- If the schema name of the procedure refers to an existing schema, CREATEIN privilege on the schema.
- DBADM authority

The privileges held by the authorization ID of the statement must also include all of the privileges necessary to invoke the SQL statements that are specified in the procedure body.

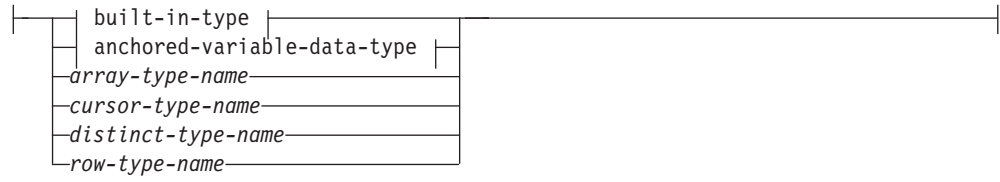
To replace an existing procedure, the authorization ID of the statement must be the owner of the existing procedure (SQLSTATE 42501).

Syntax

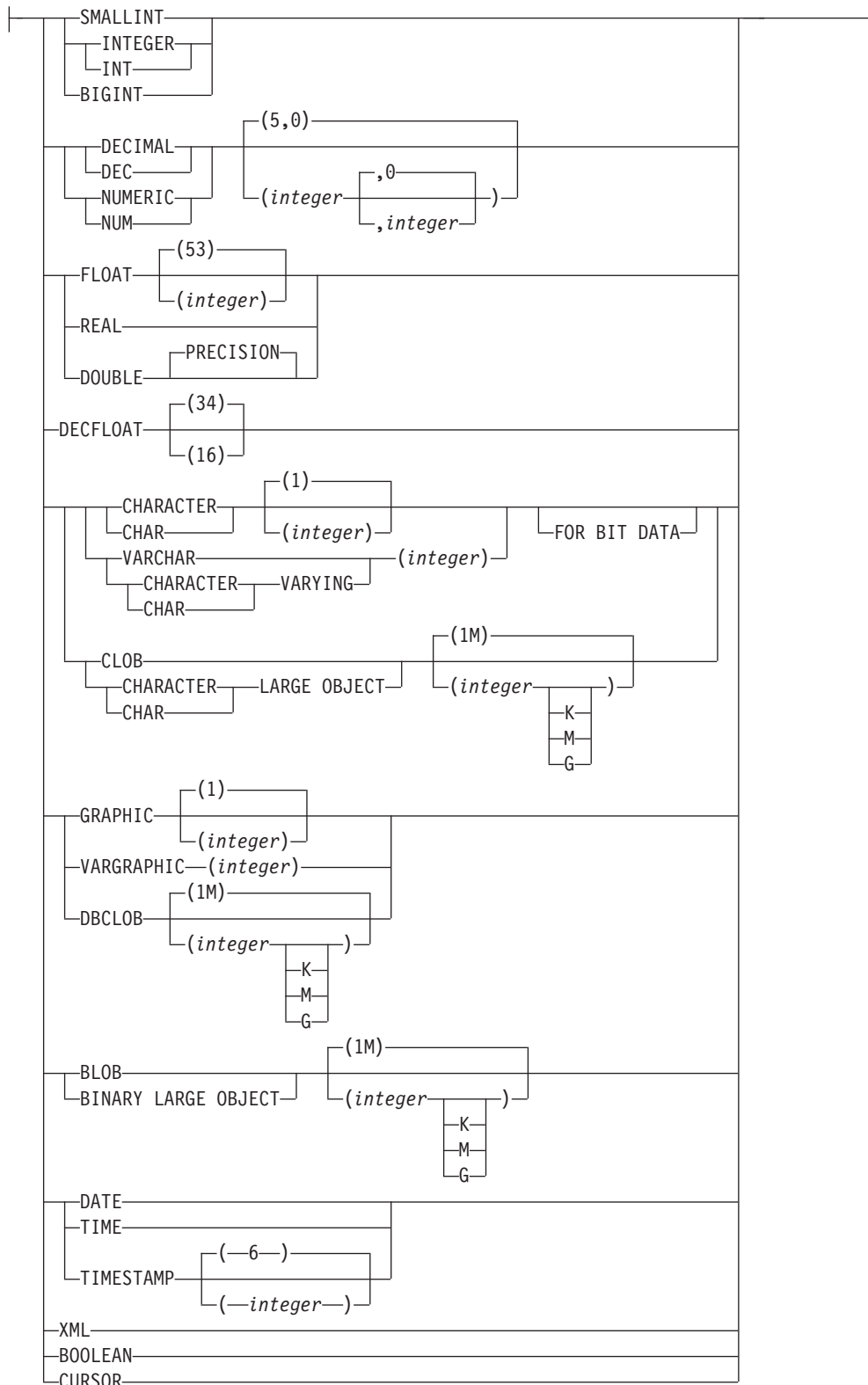


data-type:

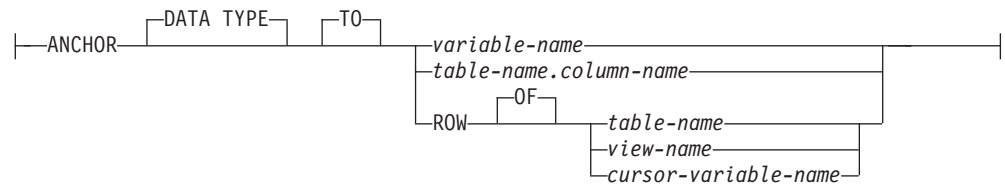
|



built-in-type:



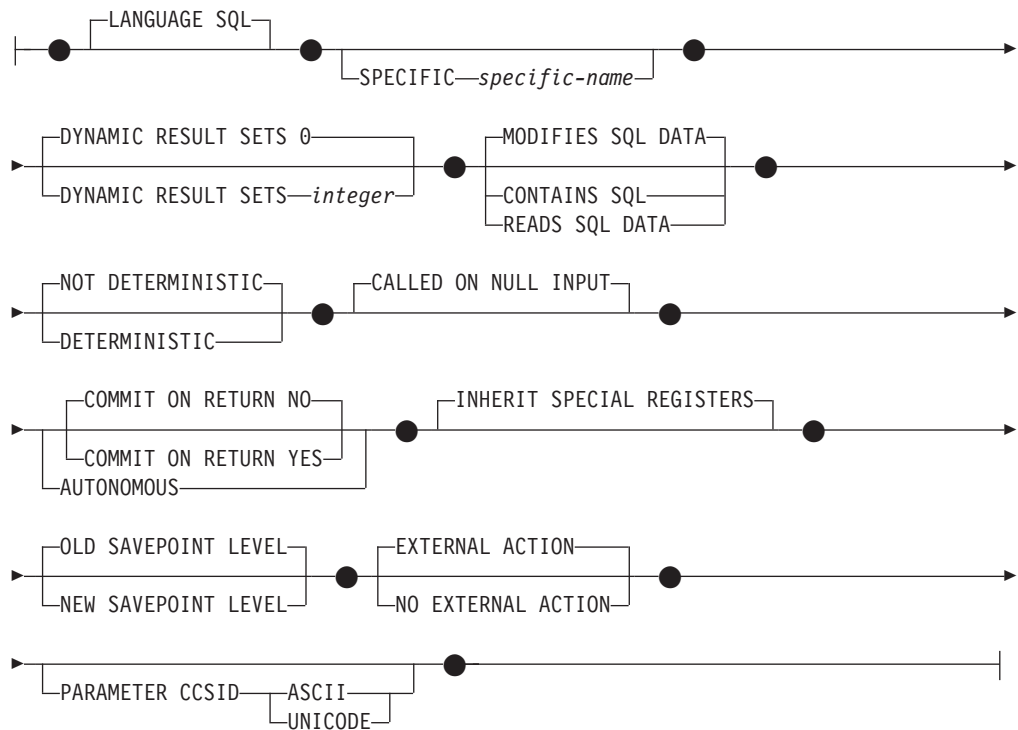
anchored-data-type:



default-clause:



option-list:



SQL-procedure-body:



Description

OR REPLACE

Specifies to replace the definition for the procedure if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the procedure are not affected. This option is ignored if a definition for the procedure does not exist at the current server. To replace an existing procedure,

the specific name and procedure name of the new definition must be the same as the specific name and procedure name of the old definition, or the signature of the new definition must match the signature of the old definition. Otherwise, a new procedure is created.

procedure-name

Names the procedure being defined. It is a qualified or unqualified name that designates a procedure. The unqualified form of *procedure-name* is an SQL identifier (with a maximum length of 128). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters, must not identify a procedure described in the catalog (SQLSTATE 42723). The unqualified name, together with the number of parameters, is unique within its schema, but does not need to be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS'; otherwise, an error is returned (SQLSTATE 42939).

(IN | OUT | INOUT *parameter-name data-type default-clause*,...)

Identifies the parameters of the procedure, and specifies the mode, name, data type, and optional default value of each parameter. One entry in the list must be specified for each parameter that the procedure will expect.

It is possible to register a procedure that has no parameters. In this case, the parentheses must still be coded, with no intervening data types. For example:

```
CREATE PROCEDURE SUBWOOFER() ...
```

No two identically-named procedures within a schema are permitted to have exactly the same number of parameters. A duplicate signature raises an SQL error (SQLSTATE 42723).

For example, given the statements:

```
CREATE PROCEDURE PART (IN NUMBER INT, OUT PART NAME CHAR(35)) ...  
CREATE PROCEDURE PART (IN COST DECIMAL(5,3), OUT COUNT INT) ...
```

the second statement will fail because the number of parameters in the procedure is the same, even if the data types are not.

IN | OUT | INOUT

Specifies the mode of the parameter.

If an error is returned by the procedure, OUT parameters are undefined and INOUT parameters are unchanged.

IN Identifies the parameter as an input parameter to the procedure. Any changes made to the parameter within the procedure are not available to the calling SQL application when control is returned. The default is IN.

OUT Identifies the parameter as an output parameter for the procedure.

INOUT

Identifies the parameter as both an input and output parameter for the procedure.

parameter-name

Specifies the name of the parameter. The parameter name must be unique for the procedure (SQLSTATE 42734).

data-type

Specifies the data type of the parameter. A structured type or reference type cannot be specified (SQLSTATE 429BB).

built-in-type

Specifies a built-in data type. For a more complete description of each built-in data type except BOOLEAN and CURSOR, which cannot be specified for a table, see "CREATE TABLE".

BOOLEAN

For a Boolean.

CURSOR

For a reference to an underlying cursor.

anchored-data-type

Identifies another object used to define the data type. The data type of the anchor object has the same limitations that apply to specifying the data type directly, or in the case of a row, to creating a row type.

ANCHOR DATA TYPE TO

Indicates an anchored data type is used to specify the data type.

variable-name

Identifies a global variable. The data type of the global variable is used as the data type for *parameter-name*.

table-name.column-name

Identifies a column name of an existing table or view. The data type of the column is used as the data type for *parameter-name*.

ROW OF *table-name* **or** *view-name*

Specifies a row of fields with names and data types that are based on the column names and column data types of the table identified by *table-name* or the view identified by *view-name*. The data type of *parameter-name* is an unnamed row type.

ROW OF *cursor-variable-name*

Specifies a row of fields with names and data types that are based on the field names and field data types of the cursor variable identified by *cursor-variable-name*. The specified cursor variable must be one of the following (SQLSTATE 428HS):

- A global variable with a strongly typed cursor data type
- A global variable with a weakly typed cursor data type that was created or declared with a CONSTANT clause specifying a *select-statement* where all the result columns are named.

If the cursor type of the cursor variable is not strongly-typed using a named row type, the data type of *parameter-name* is an unnamed row type.

array-type-name

Specifies the name of a user-defined array type. If *array-type-name* is specified without a schema name, the array type is resolved by searching the schemas in the SQL path.

cursor-type-name

Specifies the name of a cursor type. If *cursor-type-name* is specified without a schema name, the cursor type is resolved by searching the schemas in the SQL path.

distinct-type-name

Specifies the name of a distinct type. The length, precision, and scale of the parameter are, respectively, the length, precision, and scale of the source type of the distinct type. A distinct type parameter is passed as the source type of the distinct type. If *distinct-type-name* is specified without a schema name, the distinct type is resolved by searching the schemas in the SQL path.

row-type-name

Specifies the name of a user-defined row type. The fields of the parameter are the fields of the row type. If *row-type-name* is specified without a schema name, the row type is resolved by searching the schemas in the SQL path.

DEFAULT

Specifies a default value for the parameter. The default can be a constant, a special register, a global variable, an expression or the keyword NULL. The special registers that can be specified as the default are that same as those that can be specified for a column default (see *default-clause* in the CREATE TABLE statement). Other special registers can be specified as the default by using an expression.

The *expression* can be any expression of the type described in “Expressions”. If a default value is not specified, the parameter has no default and the corresponding argument cannot be omitted on invocation of the procedure. The maximum size of the *expression* is 64K bytes.

The default expression must not modify SQL data (SQLSTATE 428FL or SQLSTATE 429BL) or perform external action (SQLSTATE 42845). The expression must be assignment compatible to the parameter data type (SQLSTATE 42821).

A default cannot be specified in the following situations:

- For INOUT or OUT parameters (SQLSTATE 42601)
- For a parameter of type ARRAY, ROW, or CURSOR (SQLSTATE 429BB)

Parameters without defaults cannot be defined after a parameter with a default (SQLSTATE 428HG).

SPECIFIC *specific-name*

Provides a unique name for the instance of the procedure that is being defined. This specific name can be used when dropping the procedure or commenting on the procedure. It can never be used to invoke the procedure. The unqualified form of *specific-name* is an SQL identifier (with a maximum length of 18). The qualified form is a *schema-name* followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another procedure instance that exists at the application server; otherwise an error (SQLSTATE 42710) is raised.

The *specific-name* can be the same as an existing *procedure-name*.

If no qualifier is specified, the qualifier that was used for *procedure-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier for *procedure-name*, or an error (SQLSTATE 42882) is raised.

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is 'SQL' followed by a character timestamp: 'SQLyymmddhhmmssxx'.

DYNAMIC RESULT SETS *integer*

Indicates the estimated upper bound of returned result sets for the procedure.

CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA

Indicates the level of data access for SQL statements included in the procedure.

CONTAINS SQL

Indicates that SQL statements that neither read nor modify SQL data can be executed by the procedure (SQLSTATE 38004 or 42985). Statements that are not supported in procedures might return a different error (SQLSTATE 38003 or 42985).

READS SQL DATA

Indicates that some SQL statements that do not modify SQL data can be included in the procedure (SQLSTATE 38002 or 42985). Statements that are not supported in procedures might return a different error (SQLSTATE 38003 or 42985).

MODIFIES SQL DATA

Indicates that the procedure can execute any SQL statement except statements that are not supported in procedures (SQLSTATE 38003 or 42985).

If the BEGIN ATOMIC clause is used in a compound SQL procedure, the procedure can only be created if it is defined as MODIFIES SQL DATA.

DETERMINISTIC or NOT DETERMINISTIC

This clause specifies whether the procedure always returns the same results for given argument values (DETERMINISTIC) or whether the procedure depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC procedure must always return the same result from successive invocations with identical inputs.

This clause currently does not impact processing of the procedure.

CALLED ON NULL INPUT

CALLED ON NULL INPUT always applies to procedures. This means that the procedure is called regardless of whether any arguments are null. Any OUT or INOUT parameter can return a null value or a normal (non-null) value. Responsibility for testing for null argument values lies with the procedure.

COMMIT ON RETURN

Indicates whether a commit is to be issued on return from the procedure. The default is NO.

NO

A commit is not issued when the procedure returns.

YES

A commit is issued when the procedure returns if a positive SQLCODE is returned by the CALL statement

The commit operation includes the work that is performed by the calling application process and the procedure.

If the procedure returns result sets, the cursors that are associated with the result sets must have been defined as WITH HOLD to be usable after the commit.

AUTONOMOUS

Indicates the procedure should execute in its own autonomous transaction scope.

INHERIT SPECIAL REGISTERS

This optional clause specifies that updatable special registers in the procedure will inherit their initial values from the environment of the invoking statement. For a routine invoked in a nested object (for example a trigger or view), the initial values are inherited from the runtime environment (not inherited from the object definition).

No changes to the special registers are passed back to the caller of the procedure.

Non-updatable special registers, such as the datetime special registers, reflect a property of the statement currently executing, and are therefore set to their default values.

OLD SAVEPOINT LEVEL or NEW SAVEPOINT LEVEL

Specifies whether or not this procedure establishes a new savepoint level for savepoint names and effects. **OLD SAVEPOINT LEVEL** is the default behavior. For more information about savepoint levels, see “Rules” in “SAVEPOINT”.

LANGUAGE SQL

This clause is used to specify that the procedure body is written in the SQL language.

EXTERNAL ACTION or NO EXTERNAL ACTION

Specifies whether the procedure takes some action that changes the state of an object not managed by the database manager (**EXTERNAL ACTION**), or not (**NO EXTERNAL ACTION**). The default is **EXTERNAL ACTION**. If **NO EXTERNAL ACTION** is specified, the system can use certain optimizations that assume the procedure has no external impact.

PARAMETER CCSID

Specifies the encoding scheme to use for all string data passed into and out of the procedure. If the **PARAMETER CCSID** clause is not specified, the default is **PARAMETER CCSID UNICODE** for Unicode databases, and **PARAMETER CCSID ASCII** for all other databases.

ASCII

Specifies that string data is encoded in the database code page. If the database is a Unicode database, **PARAMETER CCSID ASCII** cannot be specified (SQLSTATE 56031).

UNICODE

Specifies that character data is in UTF-8, and that graphic data is in UCS-2. If the database is not a Unicode database, **PARAMETER CCSID UNICODE** cannot be specified (SQLSTATE 56031).

SQL-procedure-body

Specifies the SQL statement that is the body of the SQL procedure.

See *SQL-procedure-statement* in “Compound SQL (Compiled)” statement.

Rules

- **Autonomous routine restrictions:** Autonomous routines cannot return result sets and do not support the following (SQLSTATE 428H2):
 - User-defined cursor types
 - User-defined structured types

- XML as IN, OUT, and INOUT parameters

Session variables cannot be referenced within the autonomous scope.

- *Use of cursor and row types:* A procedure that uses a cursor type or row type for a parameter can only be invoked from within a compound SQL (compiled) statement (SQLSTATE 428H2), except for JDBC which can invoke a procedure with OUT parameters that have a cursor type.

Notes

- Creating a procedure with a schema name that does not already exist will result in the implicit creation of that schema, provided that the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- A procedure that is called from within a compound SQL (inlined) statement will execute as if it were created specifying NEW SAVEPOINT LEVEL, even if OLD SAVEPOINT LEVEL was specified or defaulted to when the procedure was created.
- *Creating procedures that are initially invalid:* If an object referenced in the procedure body does not exist or is marked invalid, or the definer temporarily doesn't have privileges to access the object, and if the database configuration parameter **auto_reval** is not set to DISABLED, then the procedure will still be created successfully. The procedure will be marked invalid and will be revalidated the next time it is invoked.
- *Setting of the default value:* Parameters of a procedure that are defined with a default value are set to their default value when the procedure is invoked, but only if a value is not supplied for the corresponding argument, or is specified as DEFAULT, when the procedure is invoked.
- *Privileges:* The definer of a procedure always receives the EXECUTE privilege WITH GRANT OPTION on the procedure, as well as the right to drop the procedure.
- *Compatibilities:* For compatibility with DB2 for z/OS:
 - The following syntax is accepted as the default behavior:
 - ASUTIME NO LIMIT
 - NO COLLID
 - STAY RESIDENT NO
 - For compatibility with previous versions of DB2:
 - RESULT SETS can be specified in place of DYNAMIC RESULT SETS.
 - NULL CALL can be specified in place of CALLED ON NULL INPUT.

Examples

Example 1: Create an SQL procedure that returns the median staff salary. Return a result set containing the name, position, and salary of all employees who earn more than the median salary.

```
CREATE PROCEDURE MEDIAN_RESULT_SET (OUT medianSalary DOUBLE)
  RESULT SETS 1
  LANGUAGE SQL
  BEGIN
    DECLARE v_numRecords INT DEFAULT 1;
    DECLARE v_counter INT DEFAULT 0;

    DECLARE c1 CURSOR FOR
      SELECT CAST(salary AS DOUBLE)
```

```

        FROM staff
        ORDER BY salary;
DECLARE c2 CURSOR WITH RETURN FOR
SELECT name, job, CAST(salary AS INTEGER)
FROM staff
WHERE salary > medianSalary
ORDER BY salary;

DECLARE EXIT HANDLER FOR NOT FOUND
SET medianSalary = 6666;

SET medianSalary = 0;
SELECT COUNT(*) INTO v_numRecords
FROM STAFF;
OPEN c1;
WHILE v_counter < (v_numRecords / 2 + 1)
DO
    FETCH c1 INTO medianSalary;
    SET v_counter = v_counter + 1;
END WHILE;
CLOSE c1;
OPEN c2;
END

```

CREATE ROLE

The CREATE ROLE statement defines a role at the current server.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax

►►—CREATE ROLE—*role-name*—►►

Description

role-name

Names the role. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The name must not identify an existing role at the current server (SQLSTATE 42710). The name must not begin with the characters 'SYS' and must not be 'ACCESSCTRL', 'DATAACCESS', 'DBADM', 'NONE', 'NULL', 'PUBLIC', 'SECADM', 'SQLADM', or 'WLMADM' (SQLSTATE 42939).

Example

Create a role named DOCTOR.

```
CREATE ROLE DOCTOR
```

CREATE SCHEMA

The CREATE SCHEMA statement defines a schema. It is also possible to create some objects and grant privileges on objects within the statement.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

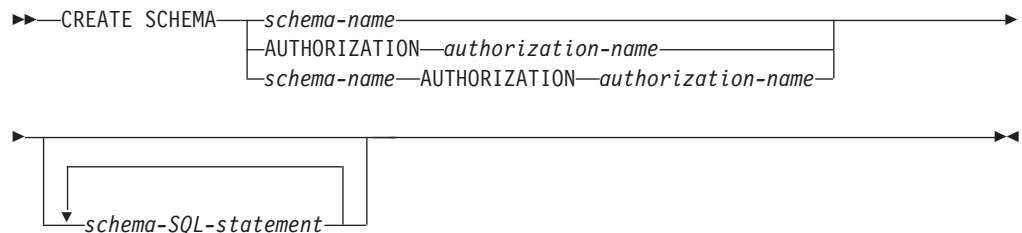
An authorization ID that holds DBADM authority can create a schema with any valid *schema-name* or *authorization-name*.

An authorization ID that does not hold DBADM authority can only create a schema with a *schema-name* or *authorization-name* that matches the authorization ID of the statement.

If the statement includes a *schema-SQL-statement*, the privileges held by the *authorization-name* (which, if not specified, defaults to the authorization ID of the statement) must include at least one of the following:

- The privileges required to perform each *schema-SQL-statement*
- DBADM authority

Syntax



Description

schema-name

Names the schema. The name must not identify a schema already described in the catalog (SQLSTATE 42710). The name cannot begin with 'SYS' (SQLSTATE 42939). The owner of the schema is the authorization ID that issued the statement.

AUTHORIZATION *authorization-name*

Identifies the user who is the owner of the schema. The value of *authorization-name* is also used to name the schema. The *authorization-name* must not identify a schema already described in the catalog (SQLSTATE 42710).

schema-name **AUTHORIZATION** *authorization-name*

Identifies a schema called *schema-name*, whose owner is *authorization-name*. The *schema-name* must not identify a schema already described in the catalog (SQLSTATE 42710). The *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

schema-SQL-statement

SQL statements that can be included as part of the CREATE SCHEMA statement are:

- CREATE TABLE statement, excluding typed tables and materialized query tables
- CREATE VIEW statement, excluding typed views
- CREATE INDEX statement
- COMMENT statement
- GRANT statement

Notes

- The owner of the schema is determined as follows:
 - If an AUTHORIZATION clause is specified, the specified *authorization-name* is the schema owner
 - If an AUTHORIZATION clause is not specified, the authorization ID that issued the CREATE SCHEMA statement is the schema owner.
- The schema owner is assumed to be a user (not a group).
- When the schema is explicitly created with the CREATE SCHEMA statement, the schema owner is granted CREATEIN, DROPIN, and ALTERIN privileges on the schema with the ability to grant these privileges to other users.
- The definer of any object created as part of the CREATE SCHEMA statement is the schema owner. The schema owner is also the grantor for any privileges granted as part of the CREATE SCHEMA statement.
- Unqualified object names in any SQL statement within the CREATE SCHEMA statement are implicitly qualified by the name of the created schema.
- If the CREATE statement contains a qualified name for the object being created, the schema name specified in the qualified name must be the same as the name of the schema being created (SQLSTATE 42875). Any other objects referenced within the statements may be qualified with any valid schema name.
- It is recommended not to use "SESSION" as a schema name. Since declared temporary tables must be qualified by "SESSION", it is possible to have an application declare a temporary table with a name identical to that of a persistent table. An SQL statement that references a table with the schema name "SESSION" will resolve (at statement compile time) to the declared temporary table rather than a persistent table with the same name. Since an SQL statement is compiled at different times for static embedded and dynamic embedded SQL statements, the results depend on when the declared temporary table is defined. If persistent tables, views or aliases are not defined with a schema name of "SESSION", these issues do not require consideration.

Examples

Example 1: As a user with DBADM authority, create a schema called RICK with the user RICK as the owner.

```
CREATE SCHEMA RICK AUTHORIZATION RICK
```

Example 2: Create a schema that has an inventory part table and an index over the part number. Give authority on the table to user JONES.

```
CREATE SCHEMA INVENTORY
```

```
CREATE TABLE PART (PARTNO SMALLINT NOT NULL,  
DESCR VARCHAR(24),  
QUANTITY INTEGER)
```

```
CREATE INDEX PARTIND ON PART (PARTNO)
```

```
GRANT ALL ON PART TO JONES
```

Example 3: Create a schema called PERS with two tables that each have a foreign key that references the other table. This is an example of a feature of the CREATE SCHEMA statement that allows such a pair of tables to be created without the use of the ALTER TABLE statement.

```
CREATE SCHEMA PERS
```

```
CREATE TABLE ORG (DEPTNUMB SMALLINT NOT NULL,  
DEPTNAME VARCHAR(14),  
MANAGER SMALLINT,  
DIVISION VARCHAR(10),  
LOCATION VARCHAR(13),  
CONSTRAINT PKEYDNO  
PRIMARY KEY (DEPTNUMB),  
CONSTRAINT FKEYMGR  
FOREIGN KEY (MANAGER)  
REFERENCES STAFF (ID) )
```

```
CREATE TABLE STAFF (ID SMALLINT NOT NULL,  
NAME VARCHAR(9),  
DEPT SMALLINT,  
JOB VARCHAR(5),  
YEARS SMALLINT,  
SALARY DECIMAL(7,2),  
COMM DECIMAL(7,2),  
CONSTRAINT PKEYID  
PRIMARY KEY (ID),  
CONSTRAINT FKEYDNO  
FOREIGN KEY (DEPT)  
REFERENCES ORG (DEPTNUMB) )
```

CREATE SECURITY LABEL

The CREATE SECURITY LABEL statement defines a security label.

Invocation

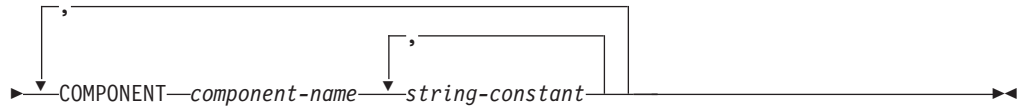
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax

```
→→ CREATE SECURITY LABEL security-label-name →→
```



Description

security-label-name

Names the security label. The name must be qualified with a security policy (SQLSTATE 42704), and must not identify an existing security label for this security policy (SQLSTATE 42710).

COMPONENT *component-name*

Specifies the name of a security label component. If the component is not part of the security policy *security-policy-name*, an error is returned (SQLSTATE 4274G). If a component is specified twice in the same statement, an error is returned (SQLSTATE 42713).

string-constant,...

Specifies a valid element for the security component. A valid element is one that was specified when the security component was created. If the element is invalid, an error is returned (SQLSTATE 4274F).

Examples

Example 1: Create a security label named EMPLOYEESECLABEL that is part of the DATA_ACCESS security policy, and that has the element Top Secret for the LEVEL component and the elements Research and Analysis for the COMPARTMENTS component.

```
CREATE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABEL
  COMPONENT LEVEL 'Top Secret',
  COMPONENT COMPARTMENTS 'Research', 'Analysis'
```

Example 2: Create a security label named EMPLOYEESECLABELREAD that has the element Top Secret for the LEVEL component and the element Research for the COMPARTMENTS component.

```
CREATE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELREAD
  COMPONENT LEVEL 'Top Secret',
  COMPONENT COMPARTMENTS 'Research'
```

Example 3: Create a security label named EMPLOYEESECLABELWRITE that has the element Analysis for the COMPARTMENTS component and a null value for the LEVEL component. Assume that the security policy named DATA_ACCESS is the same security policy that is used in examples 1 and 2.

```
CREATE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELWRITE
  COMPONENT COMPARTMENTS 'Analysis'
```

Example 4: Create a security label named BEGINNER that is part of an existing CLASSPOLICY security policy, and that has the element Trainee for the TRUST component and the element Morning for the SECTIONS component.

```
CREATE SECURITY LABEL CLASSPOLICY.BEGINNER
  COMPONENT TRUST 'Trainee',
  COMPONENT SECTIONS 'Morning'
```

CREATE SECURITY LABEL COMPONENT

The CREATE SECURITY LABEL COMPONENT statement defines a component that is to be used as part of a security policy.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax

►► CREATE SECURITY LABEL COMPONENT *component-name* array-clause
set-clause
tree-clause ►►

array-clause:

| ARRAY [,
string-constant] |

set-clause:

| SET { ,
string-constant } |

tree-clause:

| TREE ((string-constant ROOT ,
string-constant UNDER string-constant)) |

Description

component-name

Names the security label component. This is a one-part name. The name must not identify an existing security label component at the current server (SQLSTATE 42710).

ARRAY

Specifies an ordered set of elements.

string-constant,...

One or more string constant values that make up the set of valid values for this security label component. The order in which the array elements

appear is important. The first element ranks higher than the second element. The second element ranks higher than the third element and so on.

SET

Specifies an unordered set of elements.

string-constant,...

One or more string constant values that make up the set of valid values for this security label component. The order of the elements is not important.

TREE

Specifies a tree structure of node elements.

string-constant

One or more string constant values that make up the set of valid values for this security label component.

ROOT

Specifies that the *string-constant* that follows the keyword is the root node element of the tree.

UNDER

Specifies that the *string-constant* before the **UNDER** keyword is a child of the *string-constant* that follows the **UNDER** keyword. An element must be defined as either being the root element or as being the child of another element before it can be used as a parent, otherwise an error (SQLSTATE 42704) is returned.

Rules

These rules apply to all three types of component (ARRAY, SET, and TREE):

- Element names cannot contain any of these characters:
 - Opening parenthesis - (
 - Closing parenthesis -)
 - Comma - ,
 - Colon - :
- An element name can have no more than 32 bytes (SQLSTATE 42622).
- If a security label component is a set or a tree, no more than 64 elements can be part of that component.
- A CREATE SECURITY LABEL COMPONENT statement can specify at most 65 535 elements for a security label component of type array.
- No element name can be used more than once in the same component (SQLSTATE 42713).

Examples

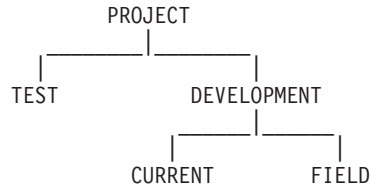
Example 1: Create an ARRAY type security label component named LEVEL. The component has the following four elements, listed in order of decreasing rank: Top Secret, Secret, Classified, and Unclassified.

```
CREATE SECURITY LABEL COMPONENT LEVEL
  ARRAY ['Top Secret', 'Secret', 'Classified', 'Unclassified']
```

Example 2: Create a SET type security label component named COMPARTMENTS. The component has the following three elements: Research, Analysis, and Collection.

```
CREATE SECURITY LABEL COMPONENT COMPARTMENTS
SET {'Collection', 'Research', 'Analysis'}
```

Example 3: Create a TREE type security label component named GROUPS. GROUPS has five elements: PROJECT, TEST, DEVELOPMENT, CURRENT, AND FIELD. The following diagram shows the relationship of these elements to one another:



```
CREATE SECURITY LABEL COMPONENT GROUPS
TREE (
  'PROJECT' ROOT,
  'TEST' UNDER 'PROJECT',
  'DEVELOPMENT' UNDER 'PROJECT',
  'CURRENT' UNDER 'DEVELOPMENT',
  'FIELD' UNDER 'DEVELOPMENT'
)
```

CREATE SECURITY POLICY

The CREATE SECURITY POLICY statement defines a security policy.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax

```

▶▶ CREATE SECURITY POLICY security-policy-name
▶ COMPONENTS component-name WITH DB2LBACRULES
▶ [ OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL
  | RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL ]

```

Description

security-policy-name

Names the security policy. This is a one-part name. The name must not identify an existing security policy at the current server (SQLSTATE 42710).

COMPONENTS *component-name,...*

Identifies a security label component. The name must identify a security label component that already exists at the current server (SQLSTATE 42704). The same security component must not be specified more than once for the security policy (SQLSTATE 42713). No more than 16 security label components can be specified for a security policy (SQLSTATE 54062).

WITH DB2LBACRULES

Indicates what rule set that will be used when comparing security labels that are part of this security policy. There is currently only one rule set: DB2LBACRULES.

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL or RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL

Specifies the action that is to be taken when a user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement issued against a table that is protected with this security policy. A user's security label and exemption credentials determine the user's authorization to write an explicitly provided security label. The default is OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL.

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL

Indicates that the value of the user's security label, rather than the explicitly specified security label, is to be used for write access during an insert or update operation.

RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL

Indicates that the insert or update operation will fail if the user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement (SQLSTATE 42519).

Notes

- **DB2LBACRULES rule set:** DB2LBACRULES is a predefined set of rules that includes the following rules: DB2LBACREADARRAY, DB2LBACREADSET, DB2LBACREADTREE, DB2LBACWRITEARRAY, DB2LBACWRITASET, DB2LBACWRITETREE.
- Group and role authorizations are not considered by default when a security policy is created. Use the ALTER SECURITY POLICY statement to change this behavior and have them considered.

Examples

Example 1: Create a security policy named DATA_ACCESS that uses the DB2LBACRULES rule set and has two components: LEVEL and COMPARTMENTS, in that order. Assume that both components already exist.

```
CREATE SECURITY POLICY DATA_ACCESS  
COMPONENTS LEVEL, COMPARTMENTS  
WITH DB2LBACRULES
```

Example 2: Create a security policy named CONTRIBUTIONS that has the components MEMBER and BADGE, which are assumed to already exist.

CREATE TABLE

The CREATE TABLE statement defines a table. The definition must include its name and the names and attributes of its columns. The definition can include other attributes of the table, such as its primary key or check constraints.

To create a created temporary table, use the CREATE GLOBAL TEMPORARY TABLE statement. To declare a declared temporary table, use the DECLARE GLOBAL TEMPORARY TABLE statement.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include either DBADM authority, or CREATETAB authority in combination with further authorization, as described here:

- One of the following privileges and authorities:
 - USE privilege on the table space
 - SYSADM
 - SYSCTRL
- Plus one of these privileges and authorities:
 - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
 - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema

If a subtable is being defined, the authorization ID must be the same as the owner of the root table of the table hierarchy.

To define a foreign key, the privileges held by the authorization ID of the statement must include one of the following on the parent table:

- REFERENCES privilege on the table
- REFERENCES privilege on each column of the specified parent key
- CONTROL privilege on the table
- DBADM authority

To define a materialized query table (using a fullselect), the privileges held by the authorization ID of the statement must include at least one of the following on each table or view identified in the fullselect (excluding group privileges):

- SELECT privilege on the table or view
- CONTROL privilege on the table or view
- DATAACCESS authority

When you are defining a materialized query table and you specify certain clauses of the CREATE TABLE statement, additional authorization might be required or can be used instead:

- If WITH NO DATA is specified, at least one of the following authorities is also sufficient:
 - DBADM
 - SQLADM
 - EXPLAIN
- If REFRESH DEFERRED or REFRESH IMMEDIATE is specified, at least one of the following privileges or authority is required on each table or view identified in the fullselect:
 - ALTER privilege on the table or view
 - CONTROL privilege on the table or view
 - DBADM authority

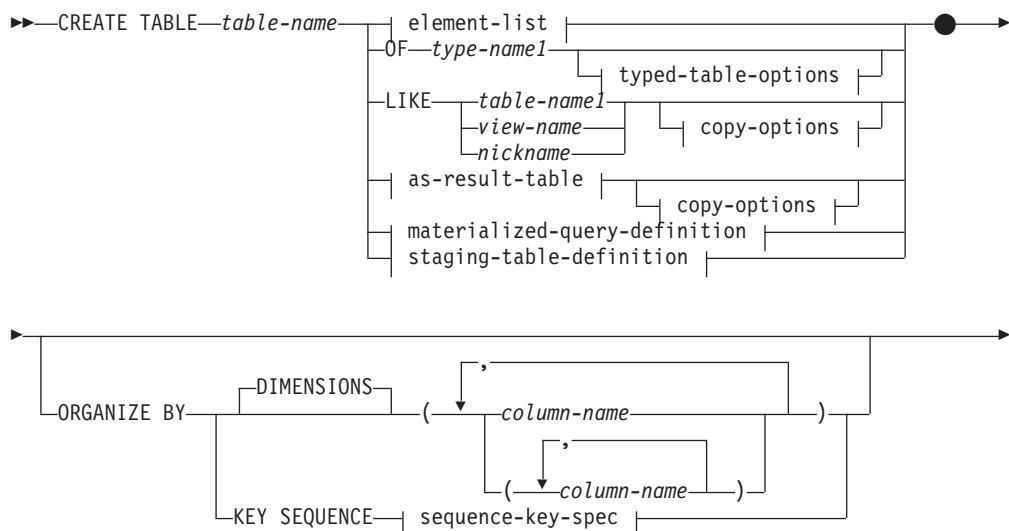
To define a staging table associated with a materialized query table, the privileges held by the authorization ID of the statement must include at least one of the following on the materialized query table:

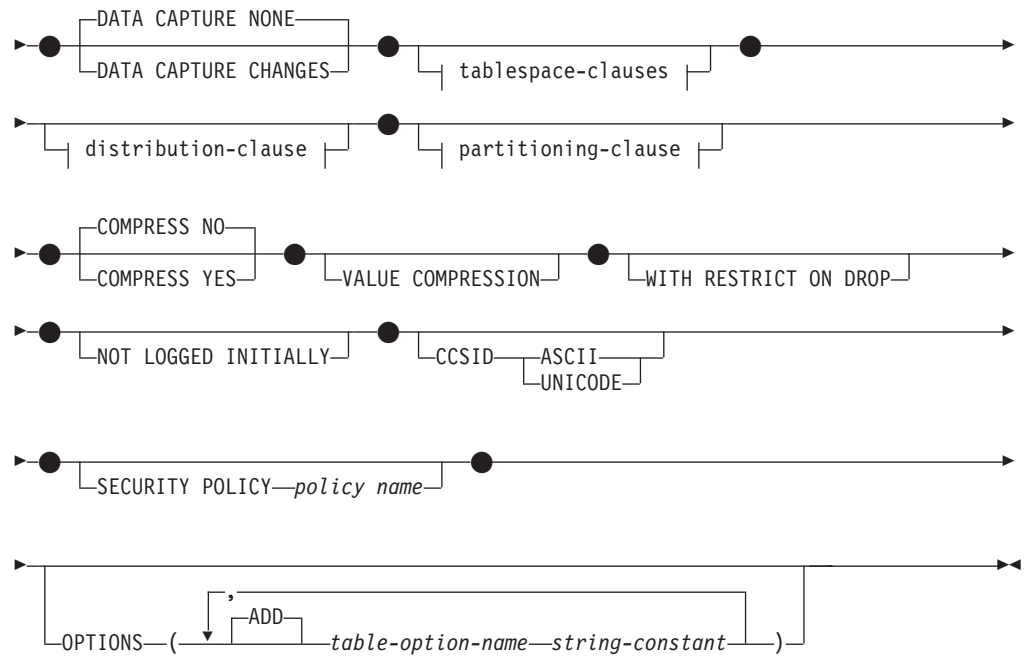
- ALTER privilege on the materialized query table
- CONTROL privilege on the materialized query table
- DBADM authority

and at least one of the following on each table or view identified in the fullselect of the materialized query table:

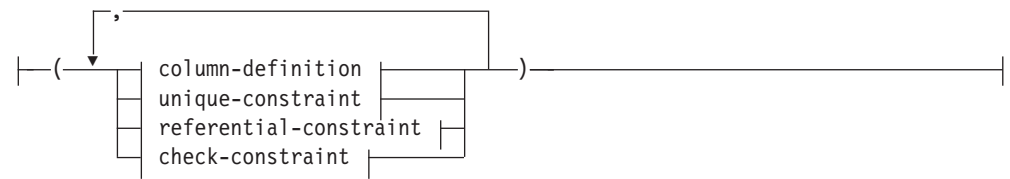
- SELECT privilege or DATAACCESS authority on the table or view, and at least one of the following:
 - ALTER privilege on the table or view
 - DBADM authority
- CONTROL privilege on the table or view

Syntax

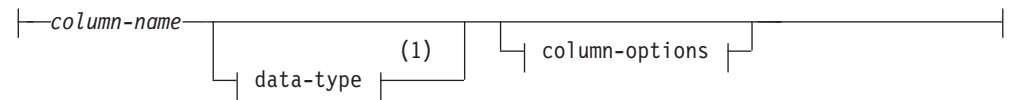




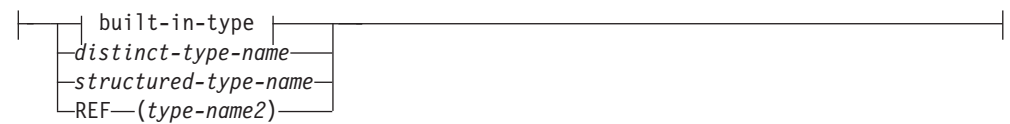
element-list:



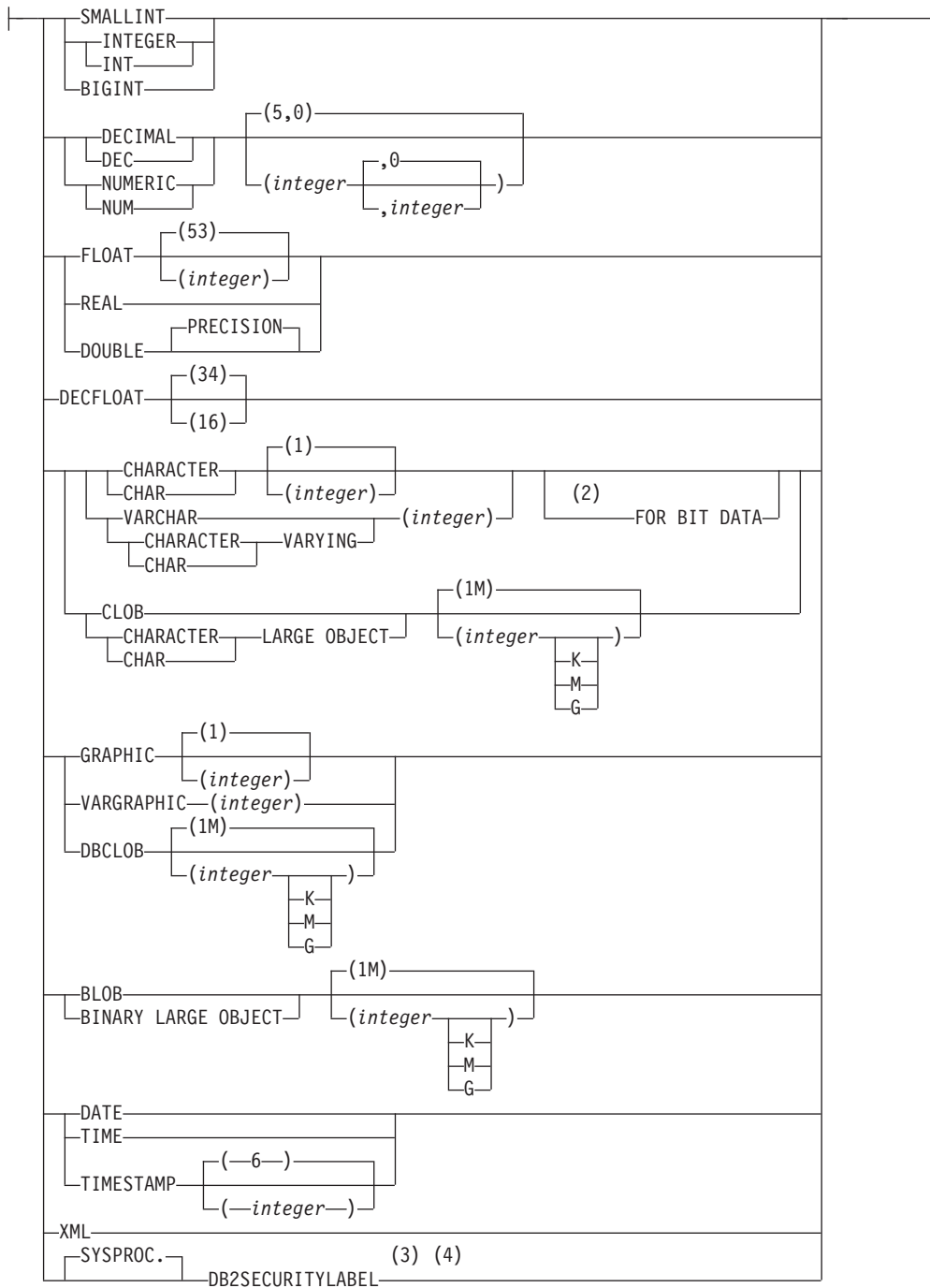
column-definition:



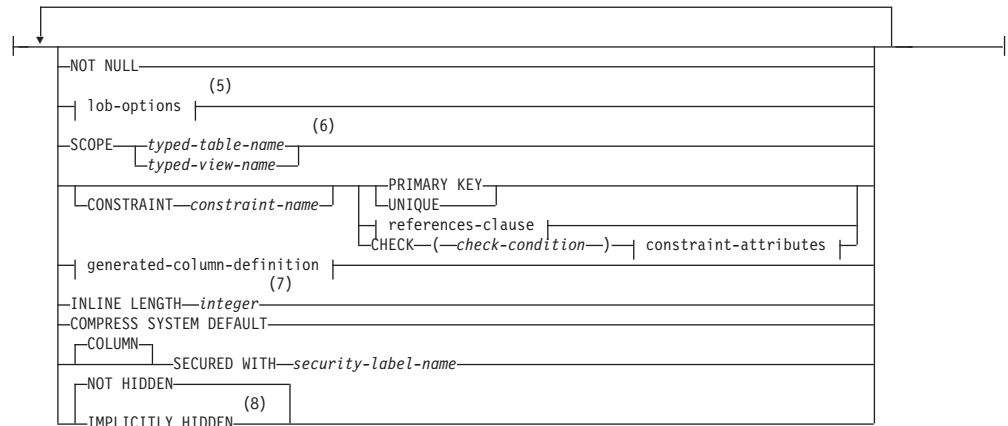
data-type:



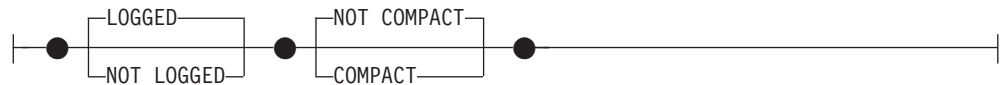
built-in-type:



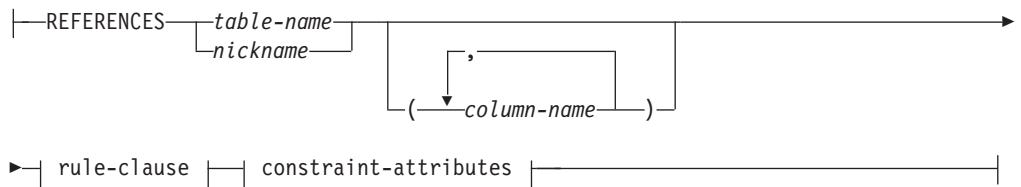
column-options:



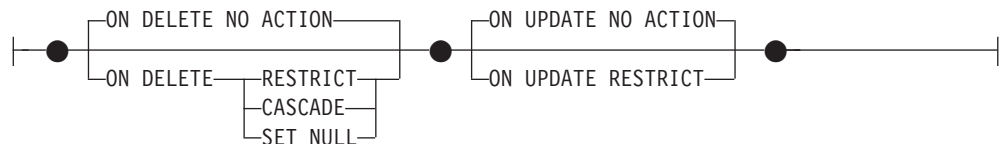
lob-options:



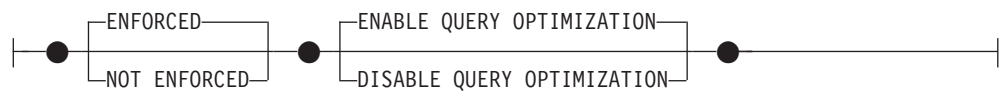
references-clause:



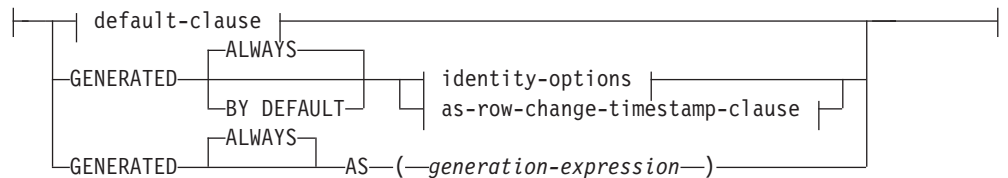
rule-clause:



constraint-attributes:



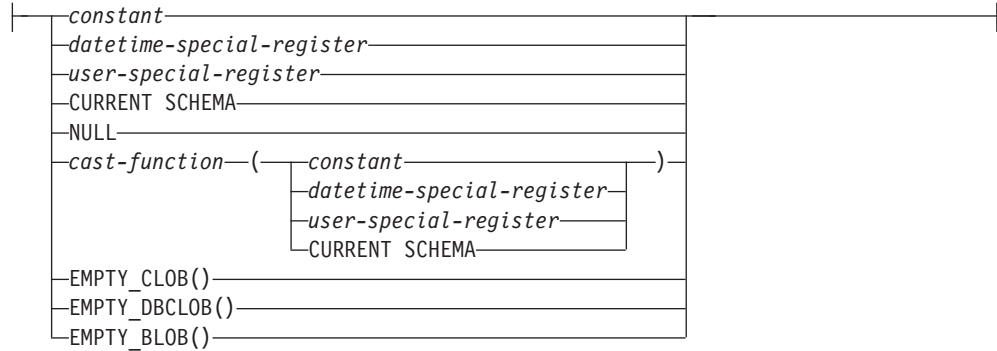
generated-column-definition:



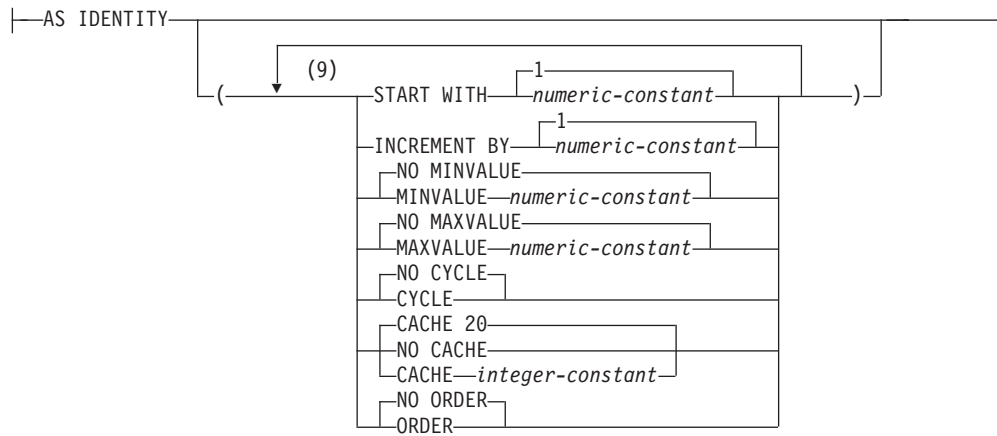
default-clause:



default-values:



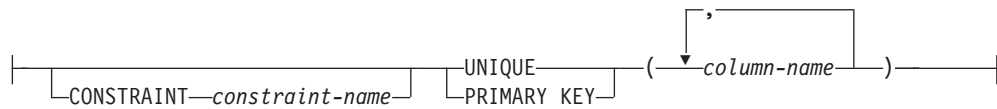
identity-options:



as-row-change-timestamp-clause:



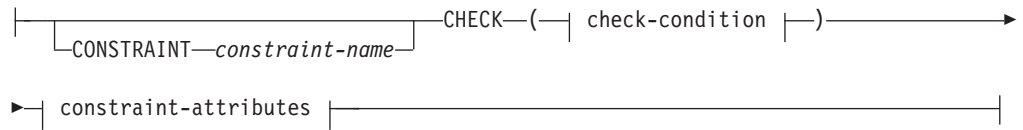
unique-constraint:



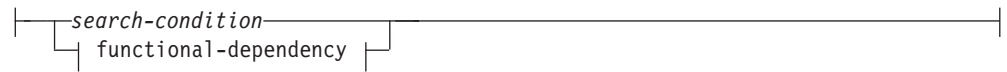
referential-constraint:



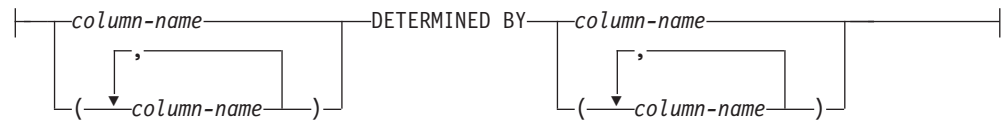
check-constraint:



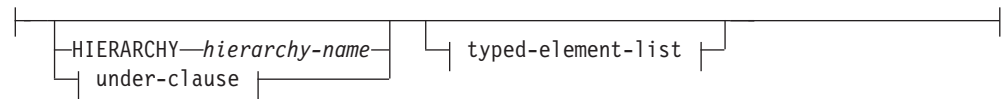
check-condition:



functional-dependency:



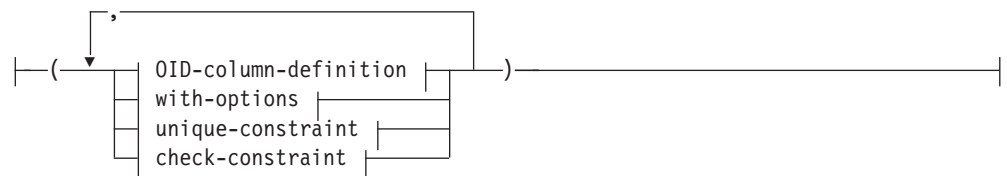
typed-table-options:



under-clause:



typed-element-list:



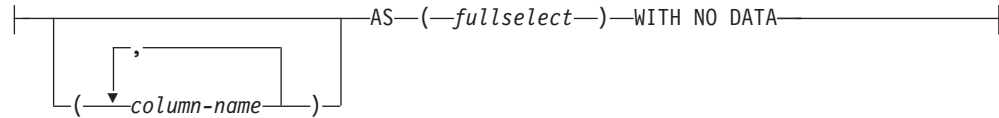
OID-column-definition:



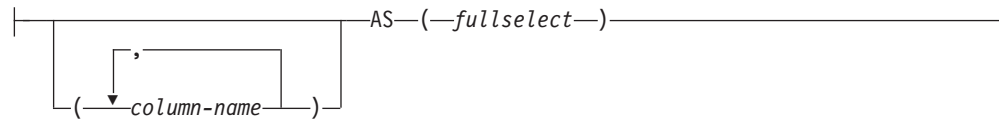
with-options:



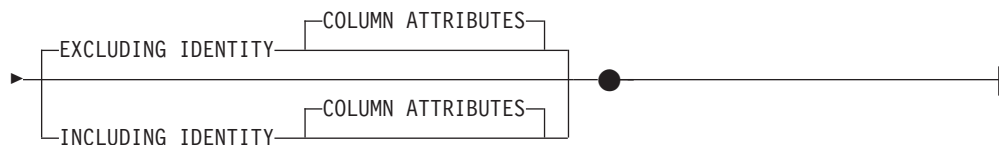
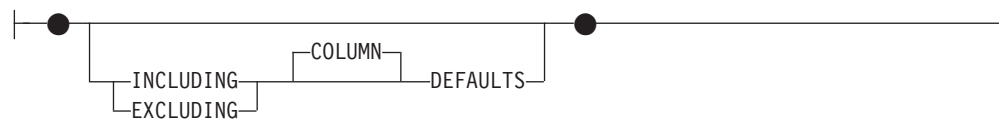
as-result-table:



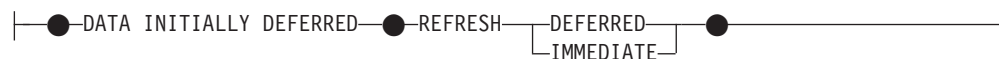
materialized-query-definition:



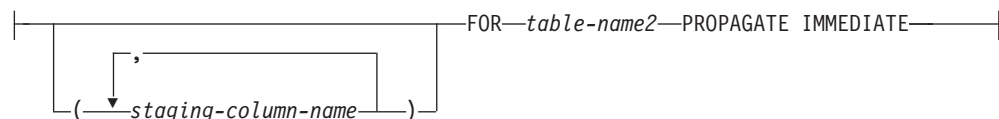
copy-options:



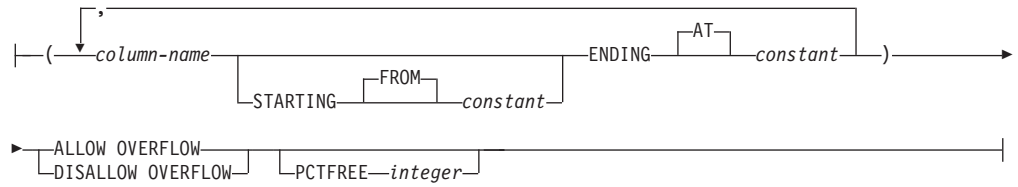
refreshable-table-options:



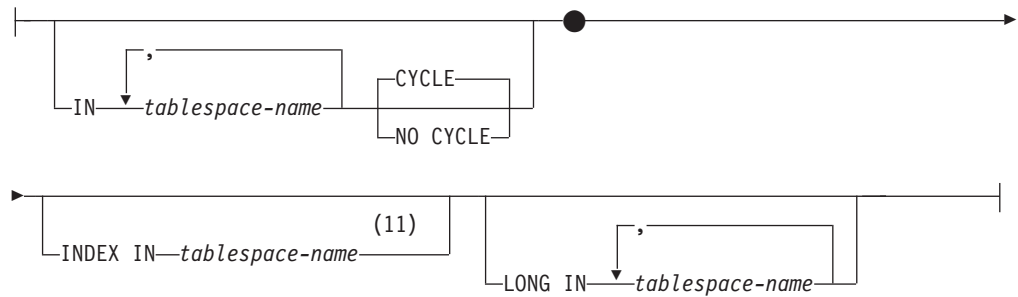
staging-table-definition:



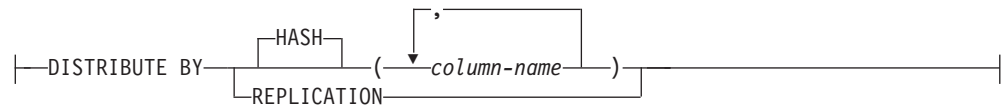
sequence-key-spec:



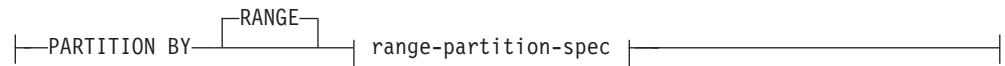
tablespace-clauses:



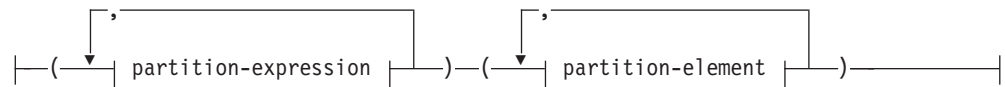
distribution-clause:



partitioning-clause:



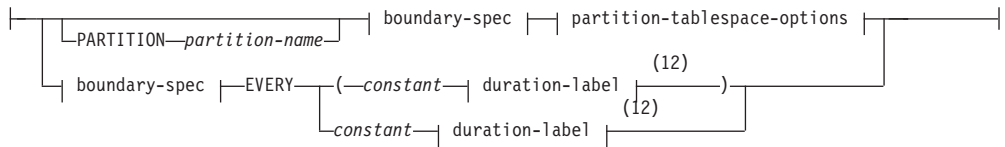
range-partition-spec:



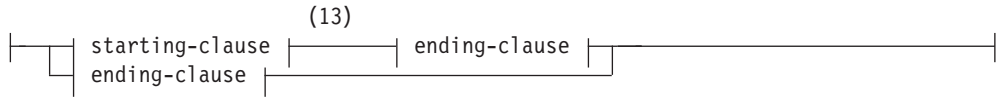
partition-expression:



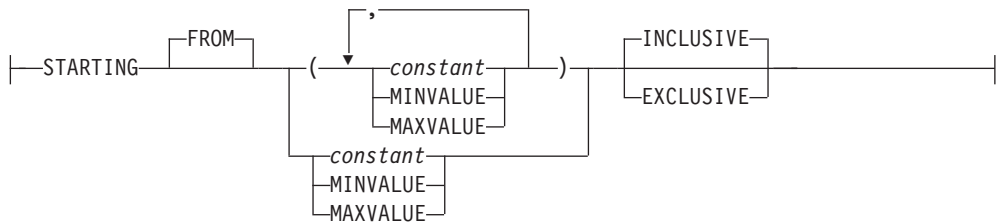
partition-element:



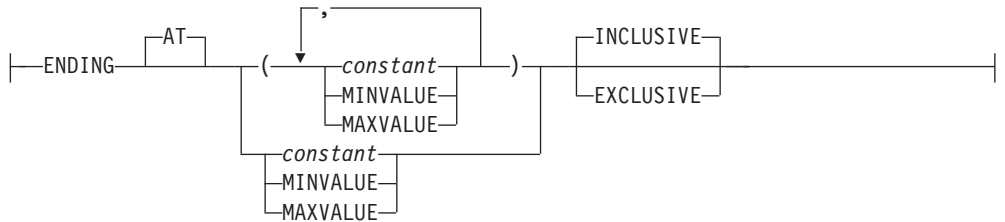
boundary-spec:



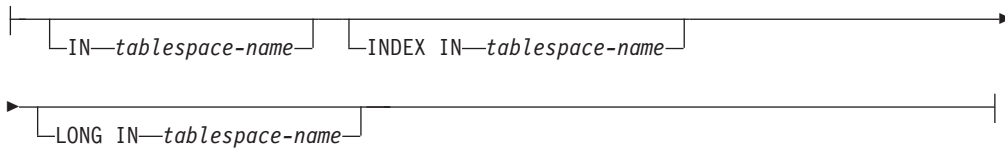
starting-clause:



ending-clause:



partition-tablespace-options:



duration-label:

YEAR
YEARS
MONTH
MONTHS
DAY
DAYS
HOUR
HOURS
MINUTE
MINUTES
SECOND
SECONDS
MICROSECOND
MICROSECONDS

Notes:

- 1 If the first column-option chosen is a generated-column-definition with a generation-expression, then the data-type can be omitted. It will be determined from the resulting data type of the generation-expression.
- 2 The FOR BIT DATA clause can be specified in any order with the other column constraints that follow.
- 3 DB2SECURITYLABEL is the built-in distinct type that must be used to define the row security label column of a protected table.
- 4 For a column of type DB2SECURITYLABEL, NOT NULL WITH DEFAULT is implicit and cannot be explicitly specified (SQLSTATE 42842). The default value for a column of type DB2SECURITYLABEL is the session authorization ID's security label for write access.
- 5 The lob-options clause only applies to large object types (BLOB, CLOB and DBCLOB) and distinct types based on large object types.
- 6 The SCOPE clause only applies to the REF type.
- 7 INLINE LENGTH applies only to columns defined as structured, XML, or LOB types.
- 8 IMPLICITLY HIDDEN can only be specified if ROW CHANGE TIMESTAMP is also specified.
- 9 The same clause must not be specified more than once.
- 10 Data type is optional for a row change timestamp column if the first column-option specified is a generated-column-definition. The data type default is TIMESTAMP(6).
- 11 Specifying which table space will contain a table's indexes can be done when the table is created. If the table is a partitioned table, the index table space for a nonpartitioned index can be specified with the IN clause of the CREATE INDEX statement.
- 12 This syntax for a partition-element is valid if there is only one partition-expression with a numeric or datetime data type.
- 13 The first partition-element must include a starting-clause and the last partition-element must include an ending-clause.

Description

System-maintained materialized query tables and user-maintained materialized query tables are referred to by the common term *materialized query table*, unless there is a need to identify each one separately.

table-name

Names the table. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname, or alias described in the catalog. The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

element-list

Defines the elements of a table. This includes the definition of columns and constraints on the table.

column-definition

Defines the attributes of a column.

column-name

Names a column of the table. The name cannot be qualified, and the same name cannot be used for more than one column of the table (SQLSTATE 42711).

A table may have the following:

- A 4K page size with a maximum of 500 columns, where the byte counts of the columns must not be greater than 4 005.
- An 8K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 8 101.
- A 16K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 16 293.
- A 32K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 32 677.

For more details, see Row Size Limit.

data-type

Specifies the data type of the column.

built-in-type

For built-in types, use one of the following types.

SMALLINT

For a small integer.

INTEGER or INT

For a large integer.

BIGINT

For a big integer.

DECIMAL(*precision-integer, scale-integer*) **or** **DEC**(*precision-integer, scale-integer*)

For a decimal number. The first integer is the precision of the number; that is, the total number of digits; it may range from 1 to 31. The second integer is the scale of the number; that is, the number of digits to the right of the decimal point; it may range from 0 to the precision of the number.

If precision and scale are not specified, the default values of 5,0 are used. The words **NUMERIC** and **NUM** can be used as synonyms for **DECIMAL** and **DEC**.

FLOAT(*integer*)

For a single or double-precision floating-point number, depending on the value of the *integer*. The value of the integer must be in the range 1 through 53. The values 1 through 24 indicate single precision and the values 25 through 53 indicate double-precision.

You can also specify:

REAL For single precision floating-point.

DOUBLE

For double-precision floating-point.

DOUBLE PRECISION

For double-precision floating-point.

FLOAT

For double-precision floating-point.

DECFLOAT(*precision-integer*)

For a decimal floating-point number. The value of *precision-integer* is the precision of the number; that is, the total number of digits, which can be 16 or 34.

If the precision is not specified, a default value of 34 is used.

CHARACTER(*integer*) or **CHAR**(*integer*) or **CHARACTER** or **CHAR**

For a fixed-length character string of length *integer* bytes, which may range from 1 to 254. If the length specification is omitted, a length of 1 is assumed.

VARCHAR(*integer*), or **CHARACTER VARYING**(*integer*), or **CHAR VARYING**(*integer*)

For a varying-length character string of maximum length *integer* bytes, which may range from 1 to 32 672.

FOR BIT DATA

Specifies that the contents of the column are to be treated as bit (binary) data. During data exchange with other systems, code page conversions are not performed. Comparisons are done in binary, irrespective of the database collating sequence.

CLOB or **CHARACTER (CHAR) LARGE OBJECT**(*integer* [*K* | *M* | *G*])

For a character large object string of the specified maximum length in bytes.

The meaning of the *integer* *K* | *M* | *G* is the same as for BLOB.

If the length specification is omitted, a length of 1 048 576 (1 megabyte) is assumed.

To create CLOB strings greater than 1 gigabyte, you must specify the **NOT LOGGED** option.

It is not possible to specify the **FOR BIT DATA** clause for CLOB columns. However, a **CHAR FOR BIT DATA** string can be assigned to a CLOB column, and a **CHAR FOR BIT DATA** string can be concatenated with a CLOB string.

GRAPHIC(*integer*)

For a fixed-length graphic string of length *integer* which may range from 1 to 127. If the length specification is omitted, a length of 1 is assumed.

VARGRAPHIC(*integer*)

For a varying-length graphic string of maximum length *integer*, which may range from 1 to 16 336.

DBCLOB(*integer* [K | M | G])

For a double-byte character large object string of the specified maximum length in double-byte characters.

The meaning of the *integer* K | M | G is similar to that for BLOB. The differences are that the number specified is the number of double-byte characters, and that the maximum size is 1 073 741 823 double-byte characters.

If the length specification is omitted, a length of 1 048 576 double-byte characters is assumed.

To create DBCLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

BLOB or BINARY LARGE OBJECT(*integer* [K | M | G])

For a binary large object string of the specified maximum length in bytes.

The length may be in the range of 1 byte to 2 147 483 647 bytes.

If *integer* by itself is specified, that is the maximum length.

If *integer* K (in either upper- or lowercase) is specified, the maximum length is 1 024 times *integer*. The maximum value for *integer* is 2 097 152.

If *integer* M is specified, the maximum length is 1 048 576 times *integer*. The maximum value for *integer* is 2 048.

If *integer* G is specified, the maximum length is 1 073 741 824 times *integer*. The maximum value for *integer* is 2.

If a multiple of K, M or G that calculates out to 2 147 483 648 is specified, the actual value used is 2 147 483 647 (or 2 gigabytes minus 1 byte), which is the maximum length for a LOB column.

If the length specification is omitted, a length of 1 048 576 (1 megabyte) is assumed.

To create BLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

Any number of spaces is allowed between the integer and K, M, or G, and a space is not required. For example, all of the following are valid:

BLOB(50K) BLOB(50 K) BLOB (50 K)

DATE

For a date.

TIME

For a time.

TIMESTAMP(*integer*) or **TIMESTAMP**

For a timestamp. The *integer* must be between 0 and 12 and

specifies the precision of fractional seconds from 0 (seconds) to 12 (picoseconds). The default is 6 (microseconds).

XML

For an XML document. Only well-formed XML documents can be inserted into an XML column.

An XML column has the following restrictions:

- The column cannot be part of any index except an index over XML data. Therefore, it cannot be included as a column of a primary key or unique constraint (SQLSTATE 42962).
- The column cannot be a foreign key of a referential constraint (SQLSTATE 42962).
- A default value (WITH DEFAULT) cannot be specified for the column (SQLSTATE 42613). If the column is nullable, the default for the column is the null value.
- The column cannot be used as the distribution key (SQLSTATE 42997).
- The column cannot be used as a data partitioning key (SQLSTATE 42962).
- The column cannot be used to organize a multidimensional clustering (MDC) table (SQLSTATE 42962).
- The column cannot be used in a range-clustered table (SQLSTATE 429BG).
- The column cannot be referenced in a check constraint except in a VALIDATED predicate (SQLSTATE 42621).

When a column of type XML is created, an XML path index is created on that column. A table-level XML region index is also created when the first column of type XML is created. The name of these indexes is 'SQL' followed by a character timestamp (*yyymmddhhmmssxxx*). The schema name is SYSIBM.

SYSPROC.DB2SECURITYLABEL

This is a built-in distinct type that must be used to define the row security label column of a protected table. The underlying data type of a column of the built-in distinct type DB2SECURITYLABEL is VARCHAR(128) FOR BIT DATA. A table can have at most one column of type DB2SECURITYLABEL (SQLSTATE 428C1).

distinct-type-name

For a user-defined type that is a distinct type. If a distinct type name is specified without a schema name, the distinct type name is resolved by searching the schemas on the SQL path (defined by the FUNCSPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a distinct type, then the data type of the column is the distinct type. The length and the scale of the column are respectively the length and the scale of the source type of the distinct type.

If a column defined using a distinct type is a foreign key of a referential constraint, then the data type of the corresponding column of the primary key must have the same distinct type.

structured-type-name

For a user-defined type that is a structured type. If a structured type

name is specified without a schema name, the structured type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL, and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a structured type, then the static data type of the column is the structured type. The column may include values with a dynamic type that is a subtype of *structured-type-name*.

A column defined using a structured type cannot be used in a primary key, unique constraint, foreign key, index key or distribution key (SQLSTATE 42962).

If a column is defined using a structured type, and contains a reference-type attribute at any level of nesting, that reference-type attribute is unscoped. To use such an attribute in a dereference operation, it is necessary to specify a SCOPE explicitly, using a CAST specification.

REF (*type-name2*)

For a reference to a typed table. If *type-name2* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The underlying data type of the column is based on the representation data type specified in the REF USING clause of the CREATE TYPE statement for *type-name2* or the root type of the data type hierarchy that includes *type-name2*.

column-options

Defines additional options related to columns of the table.

NOT NULL

Prevents the column from containing null values.

If NOT NULL is not specified, the column can contain null values, and its default value is either the null value or the value provided by the WITH DEFAULT clause.

NOT HIDDEN or IMPLICITLY HIDDEN

Specifies whether or not the column is to be defined as hidden. The hidden attribute determines whether the column is included in an implicit reference to the table, or whether it can be explicitly referenced in SQL statements. The default is NOT HIDDEN.

NOT HIDDEN

Specifies that the column is included in implicit references to the table, and that the column can be explicitly referenced.

IMPLICITLY HIDDEN

Specifies that the column is not visible in SQL statements unless the column is explicitly referenced by name. For example, assuming that a table includes a column defined with the IMPLICITLY HIDDEN clause, the result of a SELECT * does not include the implicitly hidden column. However, the result of a SELECT that explicitly refers to the name of an implicitly hidden column will include that column in the result table.

IMPLICITLY HIDDEN must only be specified for a ROW CHANGE TIMESTAMP column (SQLSTATE 42867). The ROW

CHANGE TIMESTAMP FOR *table-designator* expression will resolve to an IMPLICITLY HIDDEN ROW CHANGE TIMESTAMP column. IMPLICITLY HIDDEN must not be specified for all columns of the table (SQLSTATE 428GU).

lob-options

Specifies options for LOB data types.

LOGGED

Specifies that changes made to the column are to be written to the log. The data in such columns is then recoverable with database utilities (such as RESTORE DATABASE). LOGGED is the default.

LOBs greater than 1 gigabyte cannot be logged (SQLSTATE 42993).

NOT LOGGED

Specifies that changes made to the column are not to be logged. This only applies to LOB data that is not inlined.

NOT LOGGED has no effect on a commit or rollback operation; that is, the database's consistency is maintained even if a transaction is rolled back, regardless of whether or not the LOB value is logged. The implication of not logging is that during a roll forward operation, after a backup or load operation, the LOB data will be replaced by zeros for those LOB values that would have had log records replayed during the roll forward. During crash recovery, all committed changes and changes rolled back will reflect the expected results.

COMPACT

Specifies that the values in the LOB column should take up minimal disk space (free any extra disk pages in the last group used by the LOB value), rather than leave any leftover space at the end of the LOB storage area that might facilitate subsequent append operations. Note that storing data in this way may cause a performance penalty in any append (length-increasing) operations on the column.

NOT COMPACT

Specifies some space for insertions to assist in future changes to the LOB values in the column. This is the default.

SCOPE

Identifies the scope of the reference type column.

A scope must be specified for any column that is intended to be used as the left operand of a dereference operator or as the argument of the Deref function. Specifying the scope for a reference type column may be deferred to a subsequent ALTER TABLE statement to allow the target table to be defined, usually in the case of mutually referencing tables.

typed-table-name

The name of a typed table. The table must already exist or be the same as the name of the table being created (SQLSTATE 42704). The data type of *column-name* must be REF(S), where S is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name

The name of a typed view. The view must already exist or be the same as the name of the view being created (SQLSTATE 42704). The data type of *column-name* must be REF(S), where S is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

CONSTRAINT *constraint-name*

Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same CREATE TABLE statement. (SQLSTATE 42710).

If this clause is omitted, an 18 byte long identifier that is unique among the identifiers of existing constraints defined on the table is generated by the system. (The identifier consists of "SQL" followed by a sequence of 15 numeric characters generated by a timestamp-based function.)

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* may be used as the name of an index that is created to support the constraint.

PRIMARY KEY

This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause is specified as a separate clause.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) because the primary key is inherited from the supertable.

A ROW CHANGE TIMESTAMP column cannot be used as part of a primary key (SQLSTATE 429BV).

See PRIMARY KEY within the description of the *unique-constraint* below.

UNIQUE

This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause is specified as a separate clause.

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3) since unique constraints are inherited from the supertable.

See UNIQUE within the description of the *unique-constraint* below.

references-clause

This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* under *referential-constraint* below.

CHECK (*check-condition*)

This provides a shorthand method of defining a check constraint that applies to a single column. See CHECK (*check-condition*) below.

generated-column-definition

Specifies a generated value for the column.

default-clause

Specifies a default value for the column.

WITH

An optional keyword.

DEFAULT

Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in "ALTER TABLE".

If a column is defined as XML, a default value cannot be specified (SQLSTATE 42613). The only possible default is NULL.

If the column is based on a column of a typed table, a specific default value must be specified when defining a default. A default value cannot be specified for the object identifier column of a typed table (SQLSTATE 42997).

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

If a column is defined using a structured type, the *default-clause* cannot be specified (SQLSTATE 42842).

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column. If such a column is defined NOT NULL, then the column does not have a valid default.

default-values

Specific types of default values that can be specified are as follows.

constant

Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment
- not be a floating-point constant unless the column is defined with a floating-point data type
- be a numeric constant or a decimal floating-point special value if the data type of the column is a decimal floating-point. Floating-point constants are first interpreted as DOUBLE and then converted to decimal floating-point if the target column is DECFLOAT. For DECFLOAT(16) columns, decimal constants having precision greater than 16 digits will be rounded using the rounding modes specified by the CURRENT DECFLOAT ROUNDING MODE special register.

- not have nonzero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 bytes including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*

datetime-special-register

Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified).

user-special-register

Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and CURRENT_USER can be specified in place of CURRENT USER.

CURRENT SCHEMA

Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register.

NULL

Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL may be specified within the same column definition but will result in an error on any attempt to set the column to the default value.

cast-function

This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other

datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant

Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the *cast-function* is BLOB, the constant must be a string constant.

datetime-special-register

Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register

Specifies CURRENT USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA

Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. If the cast-function is BLOB, the length attribute must be at least 8 bytes.

EMPTY_CLOB(), EMPTY_DBCLOB(), or EMPTY_BLOB()

Specifies a zero-length string as the default for the column. The column must have the data type that corresponds to the result data type of the function.

If the value specified is not valid, an error is returned (SQLSTATE 42894).

GENERATED

Indicates that DB2 generates values for the column. GENERATED must be specified if the column is to be considered an IDENTITY column or a ROW CHANGE TIMESTAMP column.

ALWAYS

Specifies that DB2 will always generate a value for the column when a row is inserted into the table, or whenever the result value of the *generation-expression* changes. The result of the expression is stored in the table. GENERATED ALWAYS is the recommended value unless data propagation or unload and reload operations are being done. GENERATED ALWAYS is the required value for generated columns.

BY DEFAULT

Specifies that DB2 will generate a value for the column when a row is inserted, or updated specifying the DEFAULT clause,

unless an explicit value is specified. BY DEFAULT is the recommended value when using data propagation or performing an unload and reload operation.

Although not explicitly required, to ensure uniqueness of the values, define a unique single-column index on generated IDENTITY columns.

AS IDENTITY

Specifies that the column is to be the identity column for this table. A table can only have a single IDENTITY column (SQLSTATE 428C1). The IDENTITY keyword can only be specified if the data type associated with the column is an exact numeric type with a scale of zero, or a user-defined distinct type for which the source type is an exact numeric type with a scale of zero (SQLSTATE 42815). SMALLINT, INTEGER, BIGINT, or DECIMAL with a scale of zero, or a distinct type based on one of these types, are considered exact numeric types. By contrast, single- and double-precision floating points are considered approximate numeric data types. Reference types, even if represented by an exact numeric type, cannot be defined as identity columns.

An identity column is implicitly NOT NULL. An identity column cannot have a DEFAULT clause (SQLSTATE 42623).

START WITH *numeric-constant*

Specifies the first value for the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA). The default is MINVALUE for ascending sequences, and MAXVALUE for descending sequences.

INCREMENT BY *numeric-constant*

Specifies the interval between consecutive values of the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), and does not exceed the value of a large integer constant (SQLSTATE 42820), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, this is a descending sequence. If this value is 0, or positive, this is an ascending sequence. The default is 1.

NO MINVALUE or MINVALUE

Specifies the minimum value at which a descending identity column either cycles or stops generating values, or an ascending identity column cycles to after reaching the maximum value.

NO MINVALUE

For an ascending sequence, the value is the START WITH value, or 1 if START WITH was not specified. For a descending sequence, the value is the minimum value of the data type of the column. This is the default.

MINVALUE *numeric-constant*

Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could

be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

NO MAXVALUE or MAXVALUE

Specifies the maximum value at which an ascending identity column either cycles or stops generating values, or a descending identity column cycles to after reaching the minimum value.

NO MAXVALUE

For an ascending sequence, the value is the maximum value of the data type of the column. For a descending sequence, the value is the START WITH value, or -1 if START WITH was not specified. This is the default.

MAXVALUE *numeric-constant*

Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

NO CYCLE or CYCLE

Specifies whether this identity column should continue to generate values after generating either its maximum or minimum value.

NO CYCLE

Specifies that values will not be generated for the identity column once the maximum or minimum value has been reached. This is the default.

CYCLE

Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, after an ascending identity column reaches the maximum value, it generates its minimum value; or after a descending sequence reaches the minimum value, it generates its maximum value. The maximum and minimum values for the identity column determine the range that is used for cycling.

When CYCLE is in effect, DB2 may generate duplicate values for an identity column. Although not explicitly required, a unique, single-column index should be defined on the generated column to ensure uniqueness of the values, if unique values are desired. If a unique index exists on such an identity column and a non-unique value is generated, an error occurs (SQLSTATE 23505).

NO CACHE or CACHE

Specifies whether to keep some pre-allocated values in memory for faster access. If a new value is needed for the identity column, and there are none available in the cache, then the end of the new cache block must be logged. However, when a new value is needed for the identity column, and there is an unused value in the cache, then the allocation of that identity

value is faster, because no logging is necessary. This is a performance and tuning option.

NO CACHE

Specifies that values for the identity column are not to be pre-allocated.

When this option is specified, the values of the identity column are not stored in the cache. In this case, every request for a new identity value results in synchronous I/O to the log.

CACHE *integer-constant*

Specifies how many values of the identity sequence are to be pre-allocated and kept in memory. When values are generated for the identity column, pre-allocating and storing values in the cache reduces synchronous I/O to the log.

If a new value is needed for the identity column and there are no unused values available in the cache, the allocation of the value involves waiting for I/O to the log. However, when a new value is needed for the identity column and there is an unused value in the cache, the allocation of that identity value can happen more quickly by avoiding the I/O to the log.

In the event of a database deactivation, either normally or due to a system failure, all cached sequence values that have not been used in committed statements are *lost*; that is, they will never be used. The value specified for the CACHE option is the maximum number of values for the identity column that could be lost in case of database deactivation. (If a database is not explicitly activated, using the ACTIVATE command or API, when the last application is disconnected from the database, an implicit deactivation occurs.)

The minimum value is 2 (SQLSTATE 42815). The default value is CACHE 20.

NO ORDER or ORDER

Specifies whether the identity values must be generated in order of request.

NO ORDER

Specifies that the values do not need to be generated in order of request. This is the default.

ORDER

Specifies that the values must be generated in order of request.

FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

Specifies that the column is a timestamp column for the table. A value is generated for the column in each row that is inserted, and for any row in which any column is updated. The value that is generated for a ROW CHANGE TIMESTAMP column is a timestamp that corresponds to the insert or update time for that

row. If multiple rows are inserted or updated with a single statement, the value of the ROW CHANGE TIMESTAMP column might be different for each row.

A table can only have one ROW CHANGE TIMESTAMP column (SQLSTATE 428C1). If *data-type* is specified, it must be TIMESTAMP or TIMESTAMP(6) (SQLSTATE 42842). A ROW CHANGE TIMESTAMP column cannot have a DEFAULT clause (SQLSTATE 42623). NOT NULL must be specified for a ROW CHANGE TIMESTAMP column (SQLSTATE 42831).

GENERATED ALWAYS AS (*generation-expression*)

Specifies that the definition of the column is based on an expression. (If the expression for a GENERATED ALWAYS column includes a user-defined external function, changing the executable for the function (such that the results change for given arguments) can result in inconsistent data. This can be avoided by using the SET INTEGRITY statement to force the generation of new values.) The *generation-expression* cannot contain any of the following (SQLSTATE 42621):

- Subqueries
- XMLQUERY or XMLEXISTS expressions
- Column functions
- Dereference operations or Deref functions
- User-defined or built-in functions that are non-deterministic
- User-defined functions using the EXTERNAL ACTION option
- User-defined functions that are not defined with NO SQL
- Host variables or parameter markers
- Special registers and built-in functions that depend on the value of a special register
- Global variables
- References to columns defined later in the column list
- References to other generated columns
- References to columns of type XML

The data type for the column is based on the result data type of the *generation-expression*. A CAST specification can be used to force a particular data type and to provide a scope (for a reference type only). If *data-type* is specified, values are assigned to the column according to the appropriate assignment rules. A generated column is implicitly considered nullable, unless the NOT NULL column option is used. The data type of a generated column and the result data type of the *generation-expression* must have equality defined (see “Assignments and comparisons”). This excludes columns and generation expressions of type LOB data types, XML, structured types, and distinct types based on any of these types (SQLSTATE 42962).

INLINE LENGTH *integer*

This option is valid only for a column defined using a structured type, XML or LOB data type (SQLSTATE 42842).

For a column of data type XML or LOB, *integer* indicates the maximum byte size of the internal representation of an XML document or LOB data to store in the base table row. XML documents that have a larger

internal representation are stored separately from the base table row in an auxiliary storage object. This takes place automatically. There is no default inline length for XML type columns. If the XML document or LOB data is stored inlined in the base table row, there is an additional overhead. For LOB data, the overhead is 4 bytes.

For a column of data type LOB, the default inline length is set to be the maximum size of the LOB descriptor if the clause is not specified. Any explicit `INLINE LENGTH` must be at least the maximum LOB descriptor size. The following table summarizes the LOB descriptor sizes.

Table 9. Sizes of the LOB descriptor for various LOB lengths

Maximum LOB length in bytes	Minimum explicit <code>INLINE LENGTH</code>
1,024	68
8,192	92
65,536	116
524,000	140
4,190,000	164
134,000,000	196
536,000,000	220
1,070,000,000	252
1,470,000,000	276
2,147,483,647	312

For a structured type column, *integer* indicates the maximum size in bytes of an instance of a structured type to store inline with the rest of the values in the row. Instances of structured types that cannot be stored inline are stored separately from the base table row, similar to the way that LOB values are stored. This takes place automatically. The default `INLINE LENGTH` for a structured-type column is the inline length of its type (specified explicitly or by default in the `CREATE TYPE` statement). If `INLINE LENGTH` of the structured type is less than 292, the value 292 is used for the `INLINE LENGTH` of the column.

Note: The inline lengths of subtypes are not counted in the default inline length, meaning that instances of subtypes may not fit inline unless an explicit `INLINE LENGTH` is specified at `CREATE TABLE` time to account for existing and future subtypes.

The explicit `INLINE LENGTH` value cannot exceed 32 673. For a structured type or XML data type, it must be at least 292 (SQLSTATE 54010).

COMPRESS SYSTEM DEFAULT

Specifies that system default values are to be stored using minimal space. If the `VALUE COMPRESSION` clause is not specified, a warning is returned (SQLSTATE 01648), and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of extra checking that is done.

The base data type must not be a DATE, TIME, TIMESTAMP, XML, or structured data type (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

COLUMN SECURED WITH *security-label-name*

Identifies a security label that exists for the security policy that is associated with the table. The name must not be qualified (SQLSTATE 42601). The table must have a security policy associated with it (SQLSTATE 55064).

unique-constraint

Defines a unique or primary key constraint. If the table has a distribution key, any unique or primary key must be a superset of the distribution key. A unique or primary key constraint cannot be specified for a table that is a subtable (SQLSTATE 429B3). Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE). If the table is a root table, the constraint applies to the table and all its subtables.

CONSTRAINT *constraint-name*

Names the primary key or unique constraint.

UNIQUE (*column-name,...*)

Defines a unique key composed of the identified columns. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see Byte Counts. For key length limits, see "SQL limits". No LOB, XML, distinct type based on one of these types, or structured type can be used as part of a unique key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008).

The set of columns in the unique key cannot be the same as the set of columns in the primary key or another unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3), because unique constraints are inherited from the supertable.

The description of the table as recorded in the catalog includes the unique key and its unique index. A unique bidirectional index, which allows forward and reverse scans, will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

PRIMARY KEY (*column-name,...*)

Defines a primary key composed of the identified columns. The clause must not be specified more than once, and the identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table, and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see Byte Counts. For key length limits, see "SQL limits". No LOB, XML, distinct type based on one of these types, or structured type can be used as part of a primary key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008).

The set of columns in the primary key cannot be the same as the set of columns in a unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

Only one primary key can be defined on a table.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) because the primary key is inherited from the supertable.

The description of the table as recorded in the catalog includes the primary key and its primary index. A unique bidirectional index, which allows forward and reverse scans, will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

If the table has a distribution key, the columns of a *unique-constraint* must be a superset of the distribution key columns; column order is unimportant.

referential-constraint

Defines a referential constraint.

CONSTRAINT *constraint-name*

Names the referential constraint.

FOREIGN KEY (*column-name,...*)

Defines a referential constraint with the specified *constraint-name*.

Let T1 denote the object table of the statement. The foreign key of the referential constraint is composed of the identified columns. Each name in the list of column names must identify a column of T1 and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see Byte Counts. For key length limits, see "SQL limits". NoLOB, XML, distinct type based on one of these types, or structured type column can be used as part of a foreign key (SQLSTATE 42962). There must be the same number of foreign key columns as there are in the parent key and the data types of the corresponding columns must be compatible (SQLSTATE 42830). Two column descriptions are compatible if they have compatible data types (both columns are numeric, character strings, graphic, date/time, or have the same distinct type).

references-clause

Specifies the parent table or the parent nickname, and the parent key for the referential constraint.

REFERENCES *table-name* or *nickname*

The table or nickname specified in a REFERENCES clause must identify a base table or a nickname that is described in the catalog, but must not identify a catalog table.

A referential constraint is a duplicate if its foreign key, parent key, and parent table or parent nickname are the same as the foreign key, parent key, and parent table or parent nickname of a previously specified referential constraint. Duplicate referential constraints are ignored, and a warning is returned (SQLSTATE 01543).

In the following discussion, let T2 denote the identified parent table, and let T1 denote the table being created (or altered). (T1 and T2 may be the same table).

The specified foreign key must have the same number of columns as the parent key of T2 and the description of the *n*th column of the foreign key must be comparable to the description of the *n*th column of that parent key. Datetime columns are not considered to be comparable to string columns for the purposes of this rule.

(*column-name*,...)

The parent key of a referential constraint is composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of T2. The same column must not be identified more than once.

The list of column names must match the set of columns (in any order) of the primary key or a unique constraint that exists on T2 (SQLSTATE 42890). If a column name list is not specified, then T2 must have a primary key (SQLSTATE 42888). Omission of the column name list is an implicit specification of the columns of that primary key in the sequence originally specified.

The referential constraint specified by a FOREIGN KEY clause defines a relationship in which T2 is the parent and T1 is the dependent.

rule-clause

Specifies what action to take on dependent tables.

ON DELETE

Specifies what action is to take place on the dependent tables when a row of the parent table is deleted. There are four possible actions:

- NO ACTION (default)
- RESTRICT
- CASCADE
- SET NULL

The delete rule applies when a row of T2 is the object of a DELETE or propagated delete operation and that row has dependents in T1. Let *p* denote such a row of T2.

- If RESTRICT or NO ACTION is specified, an error occurs and no rows are deleted.
- If CASCADE is specified, the delete operation is propagated to the dependents of *p* in T1.
- If SET NULL is specified, each nullable column of the foreign key of each dependent of *p* in T1 is set to null.

SET NULL must not be specified unless some column of the foreign key allows null values. Omission of the clause is an implicit specification of ON DELETE NO ACTION.

If T1 is delete-connected to T2 through multiple paths, defining two SET NULL rules with overlapping foreign key definitions is not allowed. For example: T1 (i1, i2, i3). Rule1 with foreign key (i1, i2) and Rule2 with foreign key (i2, i3) is not allowed.

The firing order of the rules is:

1. RESTRICT
2. SET NULL OR CASCADE
3. NO ACTION

If any row in T1 is affected by two different rules, an error occurs and no rows are deleted.

A referential constraint cannot be defined if it would cause a table to be delete-connected to itself by a cycle involving two or more tables, and where one of the delete rules is RESTRICT or SET NULL (SQLSTATE 42915).

A referential constraint that would cause a table to be delete-connected to either itself or another table by multiple paths can be defined, except in the following cases (SQLSTATE 42915):

- A table must not be both a dependent table in a CASCADE relationship (self-referencing, or referencing another table), and have a self-referencing relationship in which the delete rule is RESTRICT or SET NULL.
- A key overlaps another key when at least one column in one key is the same as a column in the other key. When a table is delete-connected to another table through multiple relationships with overlapping foreign keys, those relationships must have the same delete rule, and none of the delete rules can be SET NULL.
- When a table is delete-connected to another table through multiple relationships, and at least one of those relationships is specified with a delete rule of SET NULL, the foreign key definitions of these relationships must not contain any distribution key or multidimensional clustering (MDC) key column.
- When two tables are delete-connected to the same table through CASCADE relationships, the two tables must not be delete-connected to each other if the delete rule of the last relationship in each delete-connected path is RESTRICT or SET NULL.

If any row in T1 is affected by different delete rules, the result would be the effect of all the actions specified by these rules. AFTER triggers and CHECK constraints on T1 will also see the effect of all the actions. An example of this is a row that is targeted to be set null through one delete-connected path to an ancestor table, and targeted to be deleted by a second delete-connected path to the same ancestor table. The result would be the deletion of the row. AFTER DELETE triggers on this descendant table would be activated, but AFTER UPDATE triggers would not.

In applying the above rules to referential constraints, in which either the parent table or the dependent table is a member of a

typed table hierarchy, all the referential constraints that apply to any table in the respective hierarchies are taken into consideration.

ON UPDATE

Specifies what action is to take place on the dependent tables when a row of the parent table is updated. The clause is optional. ON UPDATE NO ACTION is the default and ON UPDATE RESTRICT is the only alternative.

The difference between NO ACTION and RESTRICT is described in the "Notes" section.

check-constraint

Defines a check constraint. A *check-constraint* is a *search-condition* that must evaluate to not false or a functional dependency that is defined between columns.

CONSTRAINT *constraint-name*

Names the check constraint.

CHECK (*check-condition*)

Defines a check constraint. The *search-condition* must be true or unknown for every row of the table.

search-condition

The *search-condition* has the following restrictions:

- A column reference must be to a column of the table being created.
- The *search-condition* cannot contain a TYPE predicate.
- The *search-condition* cannot contain any of the following (SQLSTATE 42621):
 - Subqueries
 - XMLQUERY or XMLEXISTS expressions
 - Dereference operations or Deref functions where the scoped reference argument is other than the object identifier (OID) column
 - CAST specifications with a SCOPE clause
 - Column functions
 - Functions that are not deterministic
 - Functions defined to have an external action
 - User-defined functions defined with either CONTAINS SQL or READS SQL DATA
 - Host variables
 - Parameter markers
 - *sequence-references*
 - OLAP specifications
 - Special registers and built-in functions that depend on the value of a special register
 - Global variables
 - References to generated columns other than the identity column
 - References to columns of type XML (except in a VALIDATED predicate)
 - An error tolerant *nested-table-expression*

functional-dependency

Defines a functional dependency between columns.

*column-name DETERMINED BY column-name or (column-name,...)
DETERMINED BY (column-name,...)*

The parent set of columns contains the identified columns that immediately precede the DETERMINED BY clause. The child set of columns contains the identified columns that immediately follow the DETERMINED BY clause. All of the restrictions on the *search-condition* apply to parent set and child set columns, and only simple column references are allowed in the set of columns (SQLSTATE 42621). The same column must not be identified more than once in the functional dependency (SQLSTATE 42709). The data type of the column must not be a LOB data type, a distinct type based on a LOB data type, an XML data type, or a structured type (SQLSTATE 42962). A ROW CHANGE TIMESTAMP column cannot be used as part of a primary key (SQLSTATE 429BV). No column in the child set of columns can be a nullable column (SQLSTATE 42621).

If a check constraint is specified as part of a *column-definition*, a column reference can only be made to the same column. Check constraints specified as part of a table definition can have column references identifying columns previously defined in the CREATE TABLE statement. Check constraints are not checked for inconsistencies, duplicate conditions, or equivalent conditions. Therefore, contradictory or redundant check constraints can be defined, resulting in possible errors at execution time.

The *search-condition* "IS NOT NULL" can be specified; however, it is recommended that nullability be enforced directly, using the NOT NULL attribute of a column. For example, CHECK (salary + bonus > 30000) is accepted if salary is set to NULL, because CHECK constraints must be either satisfied or unknown, and in this case, salary is unknown. However, CHECK (salary IS NOT NULL) would be considered false and a violation of the constraint if salary is set to NULL.

Check constraints with *search-condition* are enforced when rows in the table are inserted or updated. A check constraint defined on a table automatically applies to all subtables of that table.

A functional dependency is not enforced by the database manager during normal operations such as insert, update, delete, or set integrity. The functional dependency might be used during query rewrite to optimize queries. Incorrect results might be returned if the integrity of a functional dependency is not maintained.

constraint-attributes

Defines attributes associated with referential integrity or check constraints.

ENFORCED or NOT ENFORCED

Specifies whether the constraint is enforced by the database manager during normal operations such as insert, update, or delete. The default is ENFORCED.

ENFORCED

The constraint is enforced by the database manager. ENFORCED cannot be specified for a functional dependency (SQLSTATE 42621). ENFORCED cannot be specified when a referential constraint refers to a nickname (SQLSTATE 428G7).

NOT ENFORCED

The constraint is not enforced by the database manager. This should only be specified if the table data is independently known to conform to the constraint.

ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION

Specifies whether the constraint or functional dependency can be used for query optimization under appropriate circumstances. The default is ENABLE QUERY OPTIMIZATION.

ENABLE QUERY OPTIMIZATION

The constraint is assumed to be true and can be used for query optimization.

DISABLE QUERY OPTIMIZATION

The constraint cannot be used for query optimization.

OF *type-name1*

Specifies that the columns of the table are based on the attributes of the structured type identified by *type-name1*. If *type-name1* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPTH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The type name must be the name of an existing user-defined type (SQLSTATE 42704) and it must be an instantiable structured type (SQLSTATE 428DP) with at least one attribute (SQLSTATE 42997).

If UNDER is not specified, an object identifier column must be specified (refer to the *OID-column-definition*). This object identifier column is the first column of the table. The object ID column is followed by columns based on the attributes of *type-name1*.

HIERARCHY *hierarchy-name*

Names the hierarchy table associated with the table hierarchy. It is created at the same time as the root table of the hierarchy. The data for all subtables in the typed table hierarchy is stored in the hierarchy table. A hierarchy table cannot be directly referenced in SQL statements. A *hierarchy-name* is a *table-name*. The *hierarchy-name*, including the implicit or explicit schema name, must not identify a table, nickname, view, or alias described in the catalog. If the schema name is specified, it must be the same as the schema name of the table being created (SQLSTATE 428DQ). If this clause is omitted when defining the root table, a name is generated by the system. This name consists of the name of the table being created, followed by a unique suffix, such that the identifier is unique among the identifiers of existing tables, views, and nicknames.

UNDER *supertable-name*

Indicates that the table is a subtable of *supertable-name*. The supertable must be an existing table (SQLSTATE 42704) and the table must be defined using a structured type that is the immediate supertype of *type-name1* (SQLSTATE 428DB). The schema name of *table-name* and *supertable-name* must be the same (SQLSTATE 428DQ). The table identified by *supertable-name* must not have any existing subtable already defined using *type-name1* (SQLSTATE 42742).

The columns of the table include the object identifier column of the supertable with its type modified to be REF(*type-name1*), followed by columns based on the attributes of *type-name1* (remember that the type includes the attributes of its supertype). The attribute names cannot be the same as the OID column name (SQLSTATE 42711).

Other table options, including table space, data capture, not logged initially, and distribution key options cannot be specified. These options are inherited from the supertable (SQLSTATE 42613).

INHERIT SELECT PRIVILEGES

Any user or group holding a SELECT privilege on the supertable will be granted an equivalent privilege on the newly created subtable. The subtable definer is considered to be the grantor of this privilege.

typed-element-list

Defines the additional elements of a typed table. This includes the additional options for the columns, the addition of an object identifier column (root table only), and constraints on the table.

OID-column-definition

Defines the object identifier column for the typed table.

REF IS *OID-column-name* USER GENERATED

Specifies that an object identifier (OID) column is defined in the table as the first column. An OID is required for the root table of a table hierarchy (SQLSTATE 428DX). The table must be a typed table (the OF clause must be present) that is not a subtable (SQLSTATE 42613). The name for the column is defined as *OID-column-name* and cannot be the same as the name of any attribute of the structured type *type-name1* (SQLSTATE 42711). The column is defined with type REF(*type-name1*), NOT NULL and a system required unique index (with a default index name) is generated. This column is referred to as the *object identifier column* or *OID column*. The keywords USER GENERATED indicate that the initial value for the OID column must be provided by the user when inserting a row. Once a row is inserted, the OID column cannot be updated (SQLSTATE 42808).

with-options

Defines additional options that apply to columns of a typed table.

column-name

Specifies the name of the column for which additional options are specified. The *column-name* must correspond to the name of a column of the table that is not also a column of a supertable (SQLSTATE 428DJ). A column name can only appear in one WITH OPTIONS clause in the statement (SQLSTATE 42613).

If an option is already specified as part of the type definition (in CREATE TYPE), the options specified here override the options in CREATE TYPE.

WITH OPTIONS *column-options*

Defines options for the specified column. See *column-options* described earlier. If the table is a subtable, primary key or unique constraints cannot be specified (SQLSTATE 429B3).

LIKE *table-name1* or *view-name* or *nickname*

Specifies that the columns of the table have exactly the same name and description as the columns of the identified table (*table-name1*), view (*view-name*) or nickname (*nickname*). The name specified after LIKE must identify a table, view or nickname that exists in the catalog, or a declared temporary table. A typed table or typed view cannot be specified (SQLSTATE 428EC).

The use of LIKE is an implicit definition of *n* columns, where *n* is the number of columns in the identified table (including implicitly hidden columns), view,

or nickname. A column of the new table that corresponds to an implicitly hidden column in the existing table will also be defined as implicitly hidden. The implicit definition depends on what is identified after LIKE:

- If a table is identified, then the implicit definition includes the column name, data type, hidden attribute, and nullability characteristic of each of the columns of *table-name1*. If EXCLUDING COLUMN DEFAULTS is not specified, then the column default is also included.
- If a view is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each of the result columns of the fullselect defined in *view-name*.
- If a nickname is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each column of *nickname*.
- If a protected table is identified in the LIKE clause, the new table inherits the same security policy and protected columns as the identified table.

Column default and identity column attributes may be included or excluded, based on the copy-attributes clauses. The implicit definition does not include any other attributes of the identified table, view or nickname. Thus the new table does not have any unique constraints, foreign key constraints, triggers, or indexes. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

When a table is identified in the LIKE clause and that table contains a ROW CHANGE TIMESTAMP column, the corresponding column of the new table inherits only the data type of the ROW CHANGE TIMESTAMP column. The new column is not considered to be a generated column.

copy-options

These options specify whether or not to copy additional attributes of the source result table definition (table, view or fullselect).

INCLUDING COLUMN DEFAULTS

Column defaults for each updatable column of the source result table definition are copied. Columns that are not updatable will not have a default defined in the corresponding column of the created table.

If LIKE *table-name* is specified and *table-name* identifies a base table, created temporary table, or declared temporary table, then INCLUDING COLUMN DEFAULTS is the default.

EXCLUDING COLUMN DEFAULTS

Columns defaults are not copied from the source result table definition.

This clause is the default, except when LIKE *table-name* is specified and *table-name* identifies a base table, created temporary table, or declared temporary table.

INCLUDING IDENTITY COLUMN ATTRIBUTES

Identity column attributes are copied from the source result table definition, if possible. It is possible to copy the identity column attributes, if the element of the corresponding column in the table, view, or fullselect is the name of a table column, or the name of a view column which directly or indirectly maps to the name of a base table column with the identity property. In all other cases, the columns of the new table will not get the identity property. For example:

- the select-list of the fullselect includes multiple instances of an identity column name (that is, selecting the same column more than once)

- the select list of the fullselect includes multiple identity columns (that is, it involves a join)
- the identity column is included in an expression in the select list
- the fullselect includes a set operation (union, except, or intersect).

EXCLUDING IDENTITY COLUMN ATTRIBUTES

Identity column attributes are not copied from the source result table definition.

as-result-table

column-name

Names the columns in the table. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the *fullselect*. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the result table of the *fullselect*.

A list of column names must be specified if the result table of the fullselect has duplicate column names of an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

AS

Introduces the query that is used for the definition of the table.

fullselect

Defines the query on which the table is based. The resulting column definitions are the same as those for a view defined with the same query. A column of the new table that corresponds to an implicitly hidden column of a base table referenced in the *fullselect* is not considered hidden in the new table.

Every select list element must have a name (use the AS clause for expressions). The *as-result-table* defines attributes of the table.

The *fullselect* cannot include a *data-change-table-reference* clause (SQLSTATE 428FL).

Any valid *fullselect* that does not reference a typed table or a typed view can be specified.

WITH NO DATA

The query is used only to define the table. The table is not populated using the results of the query.

The columns of the table are defined based on the definitions of the columns that result from the *fullselect*. If the *fullselect* references a single table in the FROM clause, select list items that are columns of that table are defined using the column name, data type, and nullability characteristic of the referenced table.

materialized-query-definition

column-name

Names the columns in the table. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the result table of the fullselect.

A list of column names must be specified if the result table of the *fullselect* has duplicate column names of an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

AS

Introduces the query that is used for the definition of the table and that determines the data to be included in the table.

fullselect

Defines the query on which the table is based. The resulting column definitions are the same as those for a view defined with the same query. A column of the new table that corresponds to an implicitly hidden column of a base table referenced in the *fullselect* is not considered hidden in the new table.

Every select list element must have a name (use the AS clause for expressions). The *materialized-query-definition* defines attributes of the materialized query table. The option chosen also defines the contents of the *fullselect* as follows.

The *fullselect* cannot include a *data-change-table-reference* clause (SQLSTATE 428FL).

When REFRESH DEFERRED or REFRESH IMMEDIATE is specified, the *fullselect* cannot include (SQLSTATE 428EC):

- References to a materialized query table, created temporary table, declared temporary table, or typed table in any FROM clause
- References to a view where the *fullselect* of the view violates any of the listed restrictions on the *fullselect* of the materialized query table
- Expressions that are a reference type (or distinct type based on this type)
- Functions that have any of the following attributes:
 - EXTERNAL ACTION
 - LANGUAGE SQL
 - CONTAINS SQL
 - READS SQL DATA
 - MODIFIES SQL DATA
- Functions that depend on physical characteristics (for example, DBPARTITIONNUM, HASHEDVALUE, RID_BIT, RID)
- A ROW CHANGE expression or reference to a ROW CHANGE TIMESTAMP column of the row
- Table or view references to system objects (Explain tables also should not be specified)
- Expressions that are a structured type, LOB type (or a distinct type based on a LOB type), or XML type
- References to a protected table or protected nickname

When DISTRIBUTE BY REPLICATION is specified, the following restrictions apply:

- The GROUP BY clause is not allowed.
- The materialized query table must only reference a single table; that is, it cannot include a join.

When REFRESH IMMEDIATE is specified:

- The query must be a subselect, with the exception that UNION ALL is supported in the input table expression of a GROUP BY.
- The query cannot be recursive.
- The query cannot include:
 - References to a nickname
 - Functions that are not deterministic
 - Scalar fullselects
 - Predicates with fullselects
 - Special registers and built-in functions that depend on the value of a special register
 - Global variables
 - SELECT DISTINCT
 - An error tolerant *nested-table-expression*
- If the FROM clause references more than one table or view, it can only define an inner join without using the explicit INNER JOIN syntax.
- When a GROUP BY clause is specified, the following considerations apply:
 - The supported column functions are SUM, COUNT, COUNT_BIG and GROUPING (without DISTINCT). The select list must contain a COUNT(*) or COUNT_BIG(*) column. If the materialized query table select list contains SUM(X), where X is a nullable argument, the materialized query table must also have COUNT(X) in its select list. These column functions cannot be part of any expressions.
 - A HAVING clause is not allowed.
 - If in a multiple partition database partition group, the distribution key must be a subset of the GROUP BY items.
- The materialized query table must not contain duplicate rows, and the following restrictions specific to this uniqueness requirement apply, depending upon whether or not a GROUP BY clause is specified.
 - When a GROUP BY clause is specified, the following uniqueness-related restrictions apply:
 - All GROUP BY items must be included in the select list.
 - When the GROUP BY contains GROUPING SETS, CUBE, or ROLLUP, the GROUP BY items and associated GROUPING column functions in the select list must form a unique key of the result set. Thus, the following restrictions must be satisfied:
 - No grouping sets can be repeated. For example, ROLLUP(X,Y),X is not allowed, because it is equivalent to GROUPING SETS((X,Y), (X), (X)).
 - If X is a nullable GROUP BY item that appears within GROUPING SETS, CUBE, or ROLLUP, then GROUPING(X) must appear in the select list.
 - When a GROUP BY clause is not specified, the following uniqueness-related restrictions apply:
 - The materialized query table's uniqueness requirement is achieved by deriving a unique key for the materialized view from one of the unique key constraints defined in each of the underlying tables. Therefore, the underlying tables must have at least one unique key constraint defined on them, and the columns of these keys must appear in the select list of the materialized query table definition.

- When MAINTAINED BY FEDERATED_TOOL is specified, only references to nicknames are allowed in a FROM clause.

When REFRESH DEFERRED is specified:

- If the materialized query table is created with the intention of providing it with an associated staging table in a later statement, the fullselect of the materialized query table must follow the same restrictions and rules as a fullselect used to create a materialized query table with the REFRESH IMMEDIATE option.
- If the query is recursive, the materialized query table is not used to optimize the processing of queries.

A materialized query table whose fullselect contains a GROUP BY clause is summarizing data from the tables referenced in the fullselect. Such a materialized query table is also known as a *summary table*. A summary table is a specialized type of materialized query table.

refreshable-table-options

Define the refreshable options of the materialized query table attributes.

DATA INITIALLY DEFERRED

Data is not inserted into the table as part of the CREATE TABLE statement. A REFRESH TABLE statement specifying the *table-name* is used to insert data into the table.

REFRESH

Indicates how the data in the table is maintained.

DEFERRED

The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH TABLE statement is processed. System-maintained materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807). User-maintained materialized query tables defined with this attribute do allow INSERT, UPDATE, or DELETE statements.

IMMEDIATE

The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the materialized query table. In this case, the content of the table, at any point-in-time, is the same as if the specified *subselect* is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

ENABLE QUERY OPTIMIZATION

The materialized query table can be used for query optimization under appropriate circumstances.

DISABLE QUERY OPTIMIZATION

The materialized query table will not be used for query optimization. The table can still be queried directly.

MAINTAINED BY

Specifies whether the data in the materialized query table is maintained by the system, user, or replication tool. The default is SYSTEM.

SYSTEM

Specifies that the data in the materialized query table is maintained by the system.

USER

Specifies that the data in the materialized query table is maintained by the user. The user is allowed to perform update, delete, or insert operations against user-maintained materialized query tables. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against user-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY USER.

FEDERATED_TOOL

Specifies that the data in the materialized query table is maintained by the replication tool. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against federated_tool-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY FEDERATED_TOOL.

staging-table-definition

Defines the query supported by the staging table indirectly through an associated materialized query table. The underlying tables of the materialized query table are also the underlying tables for its associated staging table. The staging table collects changes that need to be applied to the materialized query table to synchronize it with the contents of the underlying tables.

staging-column-name

Names the columns in the staging table. If a list of column names is specified, it must consist of *two* more names than there are columns in the materialized query table for which the staging table is defined. If the materialized query table is a replicated materialized query table, or the query defining the materialized query table does not contain a GROUP BY clause, the list of column names must consist of *three* more names than there are columns in the materialized query table for which the staging table is defined. Each column name must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the associated materialized query table. The additional columns are named GLOBALTRANSID and GLOBALTRANSTIME, and if a third column is necessary, it is named OPERATIONTYPE.

Table 10. Extra Columns Appended in Staging Tables

Column Name	Data Type	Column Description
GLOBALTRANSID	CHAR(8) FOR BIT DATA	The global transaction ID for each propagated row
GLOBALTRANSTIME	CHAR(13) FOR BIT DATA	The timestamp of the transaction
OPERATIONTYPE	INTEGER	Operation for the propagated row, either insert, update, or delete.

A list of column names must be specified if any of the columns of the associated materialized query table duplicates any of the generated column names (SQLSTATE 42711).

FOR *table-name2*

Specifies the materialized query table that is used for the definition of the staging table. The name, including the implicit or explicit schema, must identify a materialized query table that exists at the current server defined with REFRESH DEFERRED. The fullselect of the associated materialized query table must follow the same restrictions and rules as a fullselect used to create a materialized query table with the REFRESH IMMEDIATE option.

The contents of the staging table can be used to refresh the materialized query table, by invoking the REFRESH TABLE statement, if the contents of the staging table are consistent with the associated materialized query table and the underlying source tables.

PROPAGATE IMMEDIATE

The changes made to the underlying tables as part of a delete, insert, or update operation are cascaded to the staging table in the same delete, insert, or update operation. If the staging table is not marked inconsistent, its content, at any point-in-time, is the delta changes to the underlying table since the last refresh materialized query table.

ORGANIZE BY DIMENSIONS (*column-name,...*)

Specifies a dimension for each column or group of columns used to cluster the table data. The use of parentheses within the dimension list specifies that a group of columns is to be treated as one dimension. The DIMENSIONS keyword is optional. A table whose definition specifies this clause is known as a multidimensional clustering (MDC) table.

A clustering block index is automatically maintained for each specified dimension, and a block index, consisting of all columns used in the clause, is maintained if none of the clustering block indexes includes them all. The set of columns used in the ORGANIZE BY clause must follow the rules for the CREATE INDEX statement that specifies CLUSTER.

Each column name specified in the ORGANIZE BY clause must be defined for the table (SQLSTATE 42703). A dimension cannot occur more than once in the dimension list (SQLSTATE 42709). The dimensions cannot contain a ROW CHANGE TIMESTAMP column (SQLSTATE 429BV) or an XML column (SQLSTATE 42962).

Pages of the table are arranged in blocks of equal size, which is the extent size of the table space, and all rows of each block contain the same combination of dimension values.

A table can be both a multidimensional clustering (MDC) table and a partitioned table. Columns in such a table can be used in both the *range-partition-spec* and in the MDC key. Note that table partitioning is multi-column, not multidimensional.

ORGANIZE BY KEY SEQUENCE *sequence-key-spec*

Specifies that the table is organized in ascending key sequence with a fixed size based on the specified range of key sequence values. A table organized in this way is referred to as a *range-clustered table*. Each possible key value in the defined range has a predetermined location in the physical table. The storage required for a range-clustered table must be available when the table is created, and must be sufficient to contain the number of rows in the specified range multiplied by the row size (for details on determining the space requirement, see Row Size Limit and Byte Counts).

column-name

Specifies a column of the table that is included in the unique key that determines the sequence of the range-clustered table. The data type of the column must be SMALLINT, INTEGER, or BIGINT (SQLSTATE 42611), and the columns must be defined as NOT NULL (SQLSTATE 42831). The same column must not be identified more than once in the sequence key. The number of identified columns must not exceed 64 (SQLSTATE 54008).

A unique index entry will automatically be created in the catalog for the columns in the key sequence specified with ascending order for each column. The name of the index will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name. An actual index object is not created in storage, because the table organization is ordered by this key. If a primary key or a unique constraint is defined on the same columns as the range-clustered table sequence key, this same index entry is used for the constraint.

For the key sequence specification, a check constraint exists to reflect the column constraints. If the DISALLOW OVERFLOW clause is specified, the name of the check constraint will be RCT, and the check constraint is enforced. If the ALLOW OVERFLOW clause is specified, the name of the check constraint will be RCT_OFLOW, and the check constraint is not enforced.

STARTING FROM *constant*

Specifies the constant value at the low end of the range for *column-name*. Values less than the specified constant are only allowed if the ALLOW OVERFLOW option is specified. If *column-name* is a SMALLINT or INTEGER column, the constant must be an INTEGER constant. If *column-name* is a BIGINT column, the constant must be an INTEGER or BIGINT constant (SQLSTATE 42821). If a starting constant is not specified, the default value is 1.

ENDING AT *constant*

Specifies the constant value at the high end of the range for *column-name*. Values greater than the specified constant are only allowed if the ALLOW OVERFLOW option is specified. The value of the ending constant must be greater than the starting constant. If *column-name* is a SMALLINT or INTEGER column, the constant must be an INTEGER constant. If *column-name* is a BIGINT column, the constant must be an INTEGER or BIGINT constant (SQLSTATE 42821).

ALLOW OVERFLOW

Specifies that the range-clustered table allows rows with key values that are outside of the defined range of values. When a range-clustered table is created to allow overflows, the rows with key values outside of the range are placed at the end of the defined range without any predetermined order. Operations involving these overflow rows are less efficient than operations on rows having key values within the defined range.

DISALLOW OVERFLOW

Specifies that the range-clustered table does not allow rows with key values that are not within the defined range of values (SQLSTATE 23513). Range-clustered tables that disallow overflows will always maintain all rows in ascending key sequence.

PCTFREE *integer*

Specifies the percentage of each page that is to be left as free space. The first row on each page is added without restriction. When additional rows

are added to a page, at least *integer* percent of the page is left as free space. The value of *integer* can range from 0 to 99. A PCTFREE value of -1 in the system catalog (SYSCAT.TABLES) is interpreted as the default value. The default PCTFREE value for a table page is 0.

DATA CAPTURE

Indicates whether extra information for inter-database data replication is to be written to the log. This clause cannot be specified when creating a subtable (SQLSTATE 42613).

If the table is a typed table, then this option is not supported (SQLSTATE 428DH or 42HDR).

NONE

Indicates that no extra information will be logged.

CHANGES

Indicates that extra information regarding SQL changes to this table will be written to the log. This option is required if this table will be replicated and the Capture program is used to capture changes for this table from the log.

If the schema name (implicit or explicit) of the table is longer than 18 bytes, this option is not supported (SQLSTATE 42997).

IN *tablespace-name*,...

Identifies the table spaces in which the table will be created. The table spaces must exist, they must be in the same database partition group, and they must be all regular DMS or all large DMS or all SMS table spaces (SQLSTATE 42838) on which the authorization ID of the statement holds the USE privilege.

A maximum of one IN clause is allowed at the table level. All data table spaces used by a table must have the same page size and extent size. If they do not all have the same prefetch size, a warning is returned. If all table spaces have AUTOMATIC prefetch size, no warning is returned.

If only one table space is specified, all table parts are stored in this table space. This clause cannot be specified when creating a subtable (SQLSTATE 42613), because the table space is inherited from the root table of the table hierarchy. If this clause is not specified, a table space for the table is determined as follows:

```
IF table space IBMDEFAULTGROUP (over which the user
  has USE privilege) exists with sufficient page size
  THEN choose it
ELSE IF a table space (over which the user has USE privilege)
  exists with sufficient page size (see below when
  multiple table spaces qualify)
  THEN choose it
ELSE return an error (SQLSTATE 42727)
```

If more than one table space is identified by the ELSE IF condition, choose the table space with the smallest sufficient page size. If more than one table space qualifies, choose the table space in the following order of preference, depending on to whom the USE privilege was granted:

1. The authorization ID
2. A group to which the authorization ID belongs
3. PUBLIC

If more than one table space still qualifies, the final choice is made by the database manager.

Table space determination can change if:

- Table spaces are dropped or created
- USE privileges are granted or revoked

Partitioned tables can have their data partitions spread across multiple table spaces. When multiple table spaces are specified, all of the table spaces must exist, and they must all be either SMS or regular DMS or large DMS table spaces (SQLSTATE 42838). The authorization ID of the statement must hold the USE privilege on all of the specified table spaces.

The sufficient page size of a table is determined by either the byte count of the row or the number of columns. For more information, see Row Size Limits.

When a table is placed in a large table space:

- The table can be larger than a table in a regular table space. For details on table and table space limits, see “SQL limits”.
- The table can support more than 255 rows per data page, which can improve space utilization on data pages.
- Indexes that are defined on the table will require an additional 2 bytes per row entry, compared to indexes defined on a table that resides in a regular table space.

CYCLE or NO CYCLE

Specifies whether or not the number of data partitions with no explicit table space can exceed the number of specified table spaces.

CYCLE

Specifies that if the number of data partitions with no explicit table space exceeds the number of specified table spaces, the table spaces are assigned to data partitions in a round-robin fashion.

NO CYCLE

Specifies that the number of data partitions with no explicit table space must not exceed the number of specified tables spaces (SQLSTATE 428G1). This option prevents the round-robin assignment of table spaces to data partitions.

tablespace-options

Specifies the table space in which indexes or long column values are to be stored. For details on types of table spaces, see “CREATE TABLESPACE”.

INDEX IN *tablespace-name*

Identifies the table space in which any indexes on a nonpartitioned table or nonpartitioned indexes on a partitioned table are to be created. The specified table space must exist; it must be a DMS table space if the table has data in DMS table spaces, or an SMS table space if the partitioned table has data in SMS table spaces; it must be a table space on which the authorization ID of the statement holds the USE privilege; and it must be in the same database partition group as *tablespace-name* (SQLSTATE 42838).

Specifying which table space will contain indexes can be done when a table is created or, in the case of partitioned tables, it can be done by specifying the IN clause of the CREATE INDEX statement for a nonpartitioned index. Checking for the USE privilege on the table space is done at table creation time, not when an index is created later.

For a nonpartitioned index on a partitioned table, storage of the index is as follows:

- The table space by the IN clause of the CREATE INDEX statement

- The table-level table space specified for the INDEX IN clause of the CREATE TABLE statement
- If neither of the preceding are specified, the index is stored in the table space of the first attached or visible data partition

For information about partitioned indexes on partitioned tables, see the description of the partition-element INDEX IN clause.

LONG IN *tablespace-name*

Identifies the table spaces in which the values of any long columns are to be stored. Long columns include those with LOB data types, XML type, distinct types with any of these as source types, or any columns defined with user-defined structured types whose values cannot be stored inline. This option is allowed only if the IN clause identifies a DMS table space.

The specified table space must exist. It can be a regular table space if it is the same table space in which the data is stored; otherwise, it must be a large DMS table space on which the authorization ID of the statement holds the USE privilege. It must also be in the same database partition group as *tablespace-name* (SQLSTATE 42838).

Specifying which table space will contain long, LOB, or XML columns can only be done when a table is created. Checking for the USE privilege is done at table creation time, not when a long or LOB column is added later.

For rules governing the use of the LONG IN clause with partitioned tables, see “Large object behavior in partitioned tables”.

distribution-clause

Specifies the database partitioning or the way the data is distributed across multiple database partitions.

DISTRIBUTE BY HASH (*column-name,...*)

Specifies the use of the default hashing function on the specified columns, called a *distribution key*, as the distribution method across database partitions. The *column-name* must be an unqualified name that identifies a column of the table (SQLSTATE 42703). The same column must not be identified more than once (SQLSTATE 42709). No column whose data type is BLOB, CLOB, DBCLOB, XML, distinct type based on any of these types, or structured type can be used as part of a distribution key (SQLSTATE 42962). The distribution key cannot contain a ROW CHANGE TIMESTAMP column (SQLSTATE 429BV). A distribution key cannot be specified for a table that is a subtable (SQLSTATE 42613), because the distribution key is inherited from the root table in the table hierarchy or a table with a column of data type XML (SQLSTATE 42997). If this clause is not specified, and the table resides in a multiple partition database partition group with multiple database partitions, the distribution key is defined as follows:

- If the table is a typed table, the object identifier column is the distribution key.
- If a primary key is defined, the first column of the primary key is the distribution key.
- Otherwise, the first column whose data type is valid for a distribution key becomes the distribution key.

The columns of the distribution key must be a subset of the columns that make up any unique constraints.

If none of the columns satisfies the requirements for a default distribution key, the table is created without one. Such tables are allowed only in table spaces that are defined on single-partition database partition groups.

For tables in table spaces that are defined on single-partition database partition groups, any collection of columns with data types that are valid for a distribution key can be used to define the distribution key. If you do not specify this clause, no distribution key is created.

For restrictions related to the distribution key, see Rules.

DISTRIBUTE BY REPLICATION

Specifies that the data stored in the table is physically replicated on each database partition of the database partition group for the table spaces in which the table is defined. This means that a copy of all of the data in the table exists on each database partition. This option can only be specified for a materialized query table (SQLSTATE 42997).

partitioning-clause

Specifies how the data is partitioned within a database partition.

PARTITION BY RANGE *range-partition-spec*

Specifies the table partitioning scheme for the table.

partition-expression

Specifies the key data over which the range is defined to determine the target data partition of the data.

column-name

Identifies a column of the table-partitioning key. The *column-name* must be an unqualified name that identifies a column of the table (SQLSTATE 42703). The same column must not be identified more than once (SQLSTATE 42709). No column with a data type that is a BLOB, CLOB, DBCLOB, XML, distinct type based on any of these types, or structured type can be used as part of a table-partitioning key (SQLSTATE 42962).

The numeric literals used in the range specification are governed by the rules for numeric literals. All of the numeric literals (except the decimal floating-point special values) used in ranges corresponding to numeric columns are interpreted as integer, floating-point or decimal constants, in accordance with the rules specified for numeric constants. As a result, for decimal floating-point columns, the minimum and maximum numeric constant value that can be used in the range specification of a data partition is the smallest DOUBLE value and the largest DOUBLE value, respectively. Decimal floating-point special values can be used in the range specification. All decimal floating-point special values are interpreted as greater than MINVALUE and less than MAXVALUE.

The table partitioning columns cannot contain a ROW CHANGE TIMESTAMP column (SQLSTATE 429BV). The number of identified columns must not exceed 16 (SQLSTATE 54008).

NULLS LAST

Indicates that null values compare high.

NULLS FIRST

Indicates that null values compare low.

partition-element

Specifies ranges for a data partitioning key and the table space where rows of the table in the range will be stored.

PARTITION *partition-name*

Names the data partition. The name must not be the same as any other data partition for the table (SQLSTATE 42710). If this clause is not specified, the name will be 'PART' followed by the character form of an integer value to make the name unique for the table.

boundary-spec

Specifies the boundaries of a range partition. The lowest range partition must include a starting-clause, and the highest range partition must include an ending-clause (SQLSTATE 56016). Range partitions between the lowest and the highest can include either a starting-clause, ending-clause, or both clauses. If only the ending-clause is specified, the previous range partition must also have included an ending-clause (SQLSTATE 56016).

starting-clause

Specifies the low end of the range for a data partition. There must be at least one starting value specified and no more values than the number of columns in the data partitioning key (SQLSTATE 53038). If there are fewer values specified than the number of columns, the remaining values are implicitly MINVALUE.

STARTING FROM

Introduces the *starting-clause*.

constant

Specifies a constant value with a data type that is assignable to the data type of the *column-name* to which it corresponds (SQLSTATE 53045). The value must not be in the range of any other boundary-spec for the table (SQLSTATE 56016).

MINVALUE

Specifies a value that is lower than the lowest possible value for the data type of the *column-name* to which it corresponds.

MAXVALUE

Specifies a value that is greater than the greatest possible value for the data type of the *column-name* to which it corresponds.

INCLUSIVE

Indicates that the specified range values are to be included in the data partition.

EXCLUSIVE

Indicates that the specified *constant* values are to be excluded from the data partition. This specification is ignored when MINVALUE or MAXVALUE is specified.

ending-clause

Specifies the high end of the range for a data partition. There must be at least one starting value specified and no more values than the number of columns in the data partitioning key

(SQLSTATE 53038). If there are fewer values specified than the number of columns, the remaining values are implicitly MAXVALUE.

ENDING AT

Introduces the *ending-clause*.

constant

Specifies a constant value with a data type that is assignable to the data type of the *column-name* to which it corresponds (SQLSTATE 53045). The value must not be in the range of any other boundary-spec for the table (SQLSTATE 56016).

MINVALUE

Specifies a value that is lower than the lowest possible value for the data type of the *column-name* to which it corresponds.

MAXVALUE

Specifies a value that is greater than the greatest possible value for the data type of the *column-name* to which it corresponds.

INCLUSIVE

Indicates that the specified range values are to be included in the data partition.

EXCLUSIVE

Indicates that the specified *constant* values are to be excluded from the data partition. This specification is ignored when MINVALUE or MAXVALUE is specified.

IN *tablespace-name*

Specifies the table space where the data partition is to be stored. The named table space must have the same page size, be in the same database partition group, and manage space in the same way as the other table spaces of the partitioned table (SQLSTATE 42838); it must be a table space on which the authorization ID of the statement holds the USE privilege. If this clause is not specified, a table space is assigned by default in a round-robin fashion from the list of table spaces specified for the table. If a table space was not specified for large objects using the LONG IN clause, large objects are placed in the same table space as are the rest of the rows for the data partition. For partitioned tables, the LONG IN clause can be used to provide a list of table spaces. This list is used in round robin-fashion to place large objects for each data partition. For rules governing the use of the LONG IN clause with partitioned tables, see "Large object behavior in partitioned tables".

If the INDEX IN clause is not specified on the CREATE TABLE or the CREATE INDEX statement, the index is placed in the same table space as the first visible or attached partition of the table.

INDEX IN *tablespace-name*

Specifies the table space where the partitioned index on the partitioned table is to be stored.

The partition-element level INDEX IN clause only affects the storage of partitioned indexes. Storage of the index is as follows:

- If the INDEX IN clause is specified at the partition level when the table is created, the partitioned index is stored in the specified table space.
- If the INDEX IN clause is not specified at the partition level when the table is created, the partitioned index is stored in the table space of the corresponding data partition.

The INDEX IN clause can only be specified if the data table spaces are DMS table spaces and the table space specified by the INDEX IN clause is a DMS table space. If the data table space is an SMS table space, an error is returned (SQLSTATE 42839).

LONG IN *tablespace-name*

Identifies the table spaces in which the values of any long columns are to be stored. Long columns include those with LOB data types, XML type, distinct types with any of these as source types, or any columns defined with user-defined structured types whose values cannot be stored inline. This option is allowed only if the IN clause identifies a DMS table space.

The specified table space must exist. It can be a regular table space if it is the same table space in which the data is stored; otherwise, it must be a large DMS table space on which the authorization ID of the statement holds the USE privilege. It must also be in the same database partition group as *tablespace-name* (SQLSTATE 42838).

Specifying which table space will contain long, LOB, or XML columns can only be done when a table is created. Checking for the USE privilege is done at table creation time, not when a long or LOB column is added later.

For rules governing the use of the LONG IN clause with partitioned tables, see “Large object behavior in partitioned tables”.

EVERY (*constant*)

Specifies the width of each data partition range when using the automatically generated form of the syntax. Data partitions will be created starting at the STARTING FROM value and containing this number of values in the range. This form of the syntax is only supported for tables that are partitioned by a single numeric or datetime column (SQLSTATE 53038).

If the partitioning key column is a numeric type, the starting value of the first partition is the value specified in the starting-clause. The ending value for the first and all other partitions is calculated by adding the starting value of the partition to the increment value specified as *constant* in the EVERY clause. The starting value for all other partitions is calculated by taking the starting value for the previous partition and adding the increment value specified as *constant* in the EVERY clause.

If the partitioning key column is a DATE or a TIMESTAMP, the starting value of the first partition is the value specified in the starting-clause. The ending value for the first and all other partitions is calculated by adding the starting value of the partition to the increment value specified as a labeled duration in the EVERY clause. The starting value for all other partitions is

calculated by taking the starting value for the previous partition and adding the increment value specified as a labeled duration in the EVERY clause.

For a numeric column, the EVERY value must be a positive numeric constant, and for a datetime column, the EVERY value must be a labeled duration (SQLSTATE 53045).

COMPRESS

Specifies whether data compression applies to the rows of the table

YES

Specifies that data row compression is enabled. Insert and update operations on the table will be subject to compression. After the table is sufficiently populated with data, a compression dictionary is automatically created and rows are subject to compression. This also applies to the data in the XML storage object. If there is sufficient data in the XML storage object, a compression dictionary is automatically created and XML documents are subject to compression.

NO

Specifies that data row compression is disabled.

VALUE COMPRESSION

This determines the row format that is to be used. Each data type has a different byte count depending on the row format that is used. For more information, see Byte Counts. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

The NULL value is stored using three bytes. This is the same or less space than when VALUE COMPRESSION is not active for columns of all data types, with the exception of CHAR(1). Whether or not a column is defined as nullable has no affect on the row size calculation. The zero-length data values for columns whose data type is VARCHAR, VARGRAPHIC, LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, BLOB, or XML are to be stored using two bytes only, which is less than the storage required when VALUE COMPRESSION is not active. When a column is defined using the COMPRESS SYSTEM DEFAULT option, this also allows the system default value for the column to be stored using three bytes of total storage. The row format that is used to support this determines the byte counts for each data type, and tends to cause data fragmentation when updating to or from NULL, a zero-length value, or the system default value.

WITH RESTRICT ON DROP

Indicates that the table cannot be dropped, and that the table space that contains the table cannot be dropped.

NOT LOGGED INITIALLY

Any changes made to the table by an Insert, Delete, Update, Create Index, Drop Index, or Alter Table operation in the same unit of work in which the table is created are not logged. For other considerations when using this option, see the "Notes" section of this statement.

All catalog changes and storage related information are logged, as are all operations that are done on the table in subsequent units of work.

Note: If non-logged activity occurs against a table that has the NOT LOGGED INITIALLY attribute activated, and if a statement fails (causing a rollback), or a ROLLBACK TO SAVEPOINT is executed, the entire unit of work is rolled back (SQL1476N). Furthermore, the table for which the NOT LOGGED INITIALLY

attribute was activated is marked inaccessible after the rollback has occurred, and can only be dropped. Therefore, the opportunity for errors within the unit of work in which the NOT LOGGED INITIALLY attribute is activated should be minimized.

CCSID

Specifies the encoding scheme for string data stored in the table. If the CCSID clause is not specified, the default is CCSID UNICODE for Unicode databases, and CCSID ASCII for all other databases.

ASCII

Specifies that string data is encoded in the database code page. If the database is a Unicode database, CCSID ASCII cannot be specified (SQLSTATE 56031).

UNICODE

Specifies that string data is encoded in Unicode. If the database is a Unicode database, character data is in UTF-8, and graphic data is in UCS-2. If the database is not a Unicode database, character data is in UTF-8.

If the database is not a Unicode database, tables can be created with CCSID UNICODE, but the following rules apply:

- The alternate collating sequence must be specified in the database configuration before creating the table (SQLSTATE 56031). CCSID UNICODE tables collate with the alternate collating sequence specified in the database configuration.
- Tables or table functions created with CCSID ASCII, and tables or table functions created with CCSID UNICODE, cannot both be used in a single SQL statement (SQLSTATE 53090). This applies to tables and table functions referenced directly in the statement, as well as to tables and table functions referenced indirectly (such as, for example, through referential integrity constraints, triggers, materialized query tables, and tables in the body of views).
- Tables created with CCSID UNICODE cannot be referenced in SQL functions or SQL methods (SQLSTATE 560C0).
- An SQL statement that references a table created with CCSID UNICODE cannot invoke an SQL function or SQL method (SQLSTATE 53090).
- Graphic types, the XML type, and user-defined types cannot be used in CCSID UNICODE tables (SQLSTATE 560C1).
- Anchored data types cannot anchor to columns of a table created with CCSID UNICODE (SQLSTATE 428HS).
- Tables cannot have both the CCSID UNICODE clause and the DATA CAPTURE CHANGES clause specified (SQLSTATE 42613).
- The Explain tables cannot be created with CCSID UNICODE (SQLSTATE 55002).
- Created temporary tables and declared temporary tables cannot be created with CCSID UNICODE (SQLSTATE 56031).
- CCSID UNICODE tables cannot be created in a CREATE SCHEMA statement (SQLSTATE 53090).
- The exception table for a load operation must have the same CCSID as the target table for the operation (SQLSTATE 428A5).
- The exception table for a SET INTEGRITY statement must have the same CCSID as the target table for the statement (SQLSTATE 53090).
- The target table for event monitor data must not be declared as CCSID UNICODE (SQLSTATE 55049).

- Statements that reference a CCSID UNICODE table can only be invoked from a DB2 Version 8.1 or later client (SQLSTATE 42997).
- SQL statements are always interpreted in the database code page. In particular, this means that every character in literals, hex literals, and delimited identifiers must have a representation in the database code page; otherwise, the character will be replaced with the substitution character.

Host variables in the application are always in the application code page, regardless of the CCSID of any tables in the SQL statements that are invoked. DB2 will perform code page conversions as necessary to convert data between the application code page and the section code page. The registry variable DB2CODEPAGE can be set at the client to change the application code page.

SECURITY POLICY

Names the security policy to be associated with the table.

policy-name

Identifies a security policy that already exists at the current server (SQLSTATE 42704).

OPTIONS (ADD *table-option-name string-constant, ...*)

Table options are used to identify the remote base table. The *table-option-name* is the name of the option. The *string-constant* specifies the setting for the table option. The *string-constant* must be enclosed in single quotation marks.

The remote server (the server name that was specified in the CREATE SERVER statement) must be specified in the OPTIONS clause. The OPTIONS clause can also be used to override the schema or the unqualified name of the remote base table that is being created.

It is recommended that a schema name be specified. If a remote schema name is not specified, the qualifier for the table name is used. If the table name has no qualifier, the authorization ID of the statement is used.

If an unqualified name for the remote base table is not specified, *table-name* is used.

Rules

- The sum of the byte counts of the columns, including the inline lengths of all structured or XML type columns, must not be greater than the row size limit that is based on the page size of the table space (SQLSTATE 54010). For more information, see Byte Counts. For typed tables, the byte count is applied to the columns of the root table of the table hierarchy, and every additional column introduced by every subtable in the table hierarchy (additional subtable columns must be considered nullable for byte count purposes, even if defined as not nullable). There is also an additional 4 bytes of overhead to identify the subtable to which each row belongs.
- The number of columns in a table cannot exceed 1 012 (SQLSTATE 54011). For typed tables, the total number of attributes of the types of all of the subtables in the table hierarchy cannot exceed 1010.
- An object identifier column of a typed table cannot be updated (SQLSTATE 42808).
- Any unique or primary key constraint defined on the table must be a superset of the distribution key (SQLSTATE 42997).
- The following rules only apply to multiple database partition databases.

- Tables composed only of columns with types LOB, XML, a distinct type based on one of these types, or a structured type can only be created in table spaces that are defined on single-partition database partition groups.
- The distribution key definition of a table in a table space that is defined on a multiple partition database partition group cannot be altered.
- The distribution key column of a typed table must be the OID column.
- Partitioned staging tables are not supported.
- The following restrictions apply to range-clustered tables:
 - A range-clustered table cannot be specified in a database with multiple database partitions (SQLSTATE 42997).
 - A clustering index cannot be created.
 - Altering the table to add a column is not supported.
 - Altering the table to change the data type of a column is not supported.
 - Altering the table to change PCTFREE is not supported.
 - Altering the table to set APPEND ON is not supported.
 - DETAILED statistics are not available.
 - The load utility cannot be used to populate the table.
 - Columns cannot be of type XML.
- A table is not protected unless it has a security policy associated with it and it includes either a column of type DB2SECURITYLABEL or a column defined with the SECURED WITH clause. The former indicates that the table is a protected table with **row level granularity** and the latter indicates that the table is a protected table with **column level granularity**.
- Declaring a column of type DB2SECURITYLABEL fails if the table does not have a security policy associated with it (SQLSTATE 55064).
- A security policy cannot be added to a typed table (SQLSTATE 428DH), materialized query table, or staging table (SQLSTATE 428FG).
- An error tolerant *nested-table-expression* cannot be specified in the fullselect of a *materialized-query-definition* (SQLSTATE 428GG).
- The *isolation-clause* cannot be specified in the *full-select* of the *materialized-query-table-definition* (SQLSTATE 42601).
- Subselect statements containing a *lock-request-clause* are not be eligible for MQT routing.

Notes

- Creating a table with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- If a foreign key is specified:
 - All packages with a delete usage on the parent table are invalidated.
 - All packages with an update usage on at least one column in the parent key are invalidated.
- Creating a subtable causes invalidation of all packages that depend on any table in table hierarchy.
- VARCHAR and VARGRAPHIC columns that are greater than 4 000 and 2 000 respectively should not be used as input parameters in functions in SYSFUN schema. Errors will occur when the function is invoked with an argument value that exceeds these lengths (SQLSTATE 22001).

- The use of NO ACTION or RESTRICT as delete or update rules for referential constraints determines when the constraint is enforced. A delete or update rule of RESTRICT is enforced *before* all other constraints, including those referential constraints with modifying rules such as CASCADE or SET NULL. A delete or update rule of NO ACTION is enforced *after* other referential constraints. One example where different behavior is evident involves the deletion of rows from a view that is defined as a UNION ALL of related tables.

Table T1 is a parent of table T3; delete rule as noted below.
 Table T2 is a parent of table T3; delete rule CASCADE.

```
CREATE VIEW V1 AS SELECT * FROM T1 UNION ALL SELECT * FROM T2

DELETE FROM V1
```

If table T1 is a parent of table T3 with a delete rule of RESTRICT, a restrict violation will be raised (SQLSTATE 23001) if there are any child rows for parent keys of T1 in T3.

If table T1 is a parent of table T3 with a delete rule of NO ACTION, the child rows may be deleted by the delete rule of CASCADE when deleting rows from T2 before the NO ACTION delete rule is enforced for the deletes from T1. If deletes from T2 did not result in deleting all child rows for parent keys of T1 in T3, then a constraint violation will be raised (SQLSTATE 23504).

Note that the SQLSTATE returned is different depending on whether the delete or update rule is RESTRICT or NO ACTION.

- For tables in table spaces defined on multiple partition database partition groups, table collocation should be considered when choosing the distribution keys. Following is a list of items to consider:
 - The tables must be in the same database partition group for collocation. The table spaces may be different, but must be defined in the same database partition group.
 - The distribution keys of the tables must have the same number of columns, and the corresponding key columns must be database partition-compatible for collocation.
 - The choice of distribution key also has an impact on performance of joins. If a table is frequently joined with another table, you should consider the joining column(s) as a distribution key for both tables.
- The NOT LOGGED INITIALLY option is useful for situations where a large result set needs to be created with data from an alternate source (another table or a file) and recovery of the table is not necessary. Using this option will save the overhead of logging the data. The following considerations apply when this option is specified:
 - When the unit of work is committed, all changes that were made to the table during the unit of work are flushed to disk.
 - When you run the rollforward utility and it encounters a log record that indicates that a table in the database was either populated by the Load utility or created with the NOT LOGGED INITIALLY option, the table will be marked as unavailable. The table will be dropped by the rollforward utility if it later encounters a DROP TABLE log. Otherwise, after the database is recovered, an error will be issued if any attempt is made to access the table (SQLSTATE 55019). The only operation permitted is to drop the table.
 - Once such a table is backed up as part of a database or table space back up, recovery of the table becomes possible.
- A REFRESH DEFERRED system-maintained materialized query table defined with ENABLE QUERY OPTIMIZATION can be used to optimize the processing of queries if CURRENT REFRESH AGE is set to ANY and CURRENT

MAINTAINED TABLE TYPES FOR OPTIMIZATION is set such that it includes system-maintained materialized query tables. A REFRESH DEFERRED user-maintained materialized query table defined with ENABLE QUERY OPTIMIZATION can be used to optimize the processing of queries if CURRENT REFRESH AGE is set to ANY and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION is set such that it includes user-maintained materialized query tables. A REFRESH IMMEDIATE materialized query table defined with ENABLE QUERY OPTIMIZATION is always considered for optimization. For this optimization to be able to use a REFRESH DEFERRED or a REFRESH IMMEDIATE materialized query table, the fullselect must conform to certain rules in addition to those already described. The fullselect must:

- be a subselect with a GROUP BY clause or a subselect with a single table reference
- not include DISTINCT anywhere in the select list
- not include any special registers and built-in functions that depend on the value of a special register
- not include any global variables
- not include functions that are not deterministic.

If the query specified when creating a materialized query table does not conform to these rules, a warning is returned (SQLSTATE 01633).

- If a materialized query table is defined with REFRESH IMMEDIATE, or a staging table is defined with PROPAGATE IMMEDIATE, it is possible for an error to occur when attempting to apply the change resulting from an insert, update, or delete operation on an underlying table. The error will cause the failure of the insert, update, or delete operation on the underlying table.
- Materialized query tables or staging tables cannot be used as exception tables when constraints are checked in bulk, such as during load operations or during execution of the SET INTEGRITY statement.
- Certain operations cannot be performed on a table that is referenced by a materialized query table defined with REFRESH IMMEDIATE, or defined with REFRESH DEFERRED with an associated staging table:
 - IMPORT REPLACE cannot be used.
 - ALTER TABLE NOT LOGGED INITIALLY WITH EMPTY TABLE cannot be done.
- In a federated system, nicknames for relational data sources or local tables can be used as the underlying tables to create a materialized query table. Nicknames for non-relational data sources are not supported. When a nickname is one of the underlying tables, the REFRESH DEFERRED option must be used. System-maintained materialized query tables that reference nicknames are not supported in a partitioned database environment.
- **Transparent DDL:** In a federated system, a remote base table can be created, altered, or dropped using DB2 SQL. This capability is known as *transparent DDL*. Before a remote base table can be created on a data source, the federated server must be configured to access that data source. This configuration includes creating the wrapper for the data source, supplying the server definition for the server where the remote base table will be located, and creating the user mappings between the federated server and the data source.

Transparent DDL does impose some limitations on what can be included in the CREATE TABLE statement:

- Only columns and a primary key can be created on the remote base table.
- Specific clauses supported by transparent DDL include:

- *column-definition* and *unique-constraint* in the *element-list* clause
- NOT NULL and PRIMARY KEY in the *column-options* clause
- OPTIONS
- The remote data source must support:
 - The remote column data types to which the DB2 column data types are mapped
 - The primary key option in the CREATE TABLE statement

Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.

When a remote base table is created using transparent DDL, a nickname is automatically created for that remote base table.

- A referential constraint may be defined in such a way that either the parent table or the dependent table is a part of a table hierarchy. In such a case, the effect of the referential constraint is as follows:
 1. Effects of INSERT, UPDATE, and DELETE statements:
 - If a referential constraint exists, in which PT is a parent table and DT is a dependent table, the constraint ensures that for each row of DT (or any of its subtables) that has a non-null foreign key, a row exists in PT (or one of its subtables) with a matching parent key. This rule is enforced against any action that affects a row of PT or DT, regardless of how that action is initiated.
 2. Effects of DROP TABLE statements:
 - for referential constraints in which the dropped table is the parent table or dependent table, the constraint is dropped
 - for referential constraints in which a supertable of the dropped table is the parent table the rows of the dropped table are considered to be deleted from the supertable. The referential constraint is checked and its delete rule is invoked for each of the deleted rows.
 - for referential constraints in which a supertable of the dropped table is the dependent table, the constraint is not checked. Deletion of a row from a dependent table cannot result in violation of a referential constraint.
- **Privileges:** When any table is created, the definer of the table is granted CONTROL privilege. When a subtable is created, the SELECT privilege that each user or group has on the immediate supertable is automatically granted on the subtable with the table definer as the grantor.
- **Row size limit:** The maximum number of bytes allowed in the row of a table is dependent on the page size of the table space in which the table is created (*tblspace-name1*). The following list shows the row size limit and number of columns limit associated with each table space page size.

Table 11. Limits for Number of Columns and Row Size in Each Table Space Page Size

Page Size	Row Size Limit	Column Count Limit
4K	4 005	500
8K	8 101	1 012
16K	16 293	1 012
32K	32 677	1 012

The actual number of columns for a table can be further limited by the following formula:

Total Columns * 8 + Number of LOB Columns * 12 <= Row Size Limit for Page Size

- **Byte counts:** The following table contains the byte counts of columns by data type. This is used to calculate the row size. The byte counts depend on whether or not VALUE COMPRESSION is active. When VALUE COMPRESSION is not active, the byte counts also depend on whether or not the column is nullable. If a table is based on a structured type, an additional 4 bytes of overhead is reserved to identify rows of subtables, regardless of whether or not subtables are defined. Additional subtable columns must be considered nullable for byte count purposes, even if defined as not nullable.

Table 12. Byte Counts of Columns by Data Type

Data type	VALUE COMPRESSION is		
	active ¹	Column is nullable	Column is not nullable
SMALLINT	4	3	2
INTEGER	6	5	4
BIGINT	10	9	8
REAL	6	5	4
DOUBLE	10	9	8
DECIMAL	The integral part of $(p/2)+3$, where p is the precision	The integral part of $(p/2)+2$, where p is the precision	The integral part of $(p/2)+1$, where p is the precision
DECFLOAT(16)	10	9	8
DECFLOAT(34)	18	17	16
CHAR(n)	$n+2$	$n+1$	n
VARCHAR(n)	$n+2$	$n+5$ (within a table)	$n+4$ (within a table)
LONG VARCHAR ²	22	25	24
GRAPHIC(n)	$n*2+2$	$n*2+1$	$n*2$
VARGRAPHIC(n)	$n*2+2$	$n*2+5$ (within a table)	$n*2+4$ (within a table)
LONG VARGRAPHIC ²	22	25	24
DATE	6	5	4
TIME	5	4	3
TIMESTAMP(p)	The integral part of $(p+1)/2+9$, where p is the precision of fractional seconds	The integral part of $(p+1)/2+8$, where p is the precision of fractional seconds	The integral part of $(p+1)/2+7$, where p is the precision of fractional seconds
XML (without INLINE LENGTH specified)	82	85	84
XML (with INLINE LENGTH specified)	INLINE LENGTH +2	INLINE LENGTH +4	INLINE LENGTH +3
Maximum LOB ³ length 1024 (without INLINE LENGTH specified)	70	73	72
Maximum LOB length 8192 (without INLINE LENGTH specified)	94	97	96
Maximum LOB length 65 536 (without INLINE LENGTH specified)	118	121	120

Table 12. Byte Counts of Columns by Data Type (continued)

Data type	VALUE COMPRESSION is	VALUE COMPRESSION is not active	
	active ¹	Column is nullable	Column is not nullable
Maximum LOB length 524 000 (without INLINE LENGTH specified)	142	145	144
Maximum LOB length 4 190 000 (without INLINE LENGTH specified)	166	169	168
Maximum LOB length 134 000 000 (without INLINE LENGTH specified)	198	201	200
Maximum LOB length 536 000 000 (without INLINE LENGTH specified)	222	225	224
Maximum LOB length 1 070 000 000 (without INLINE LENGTH specified)	254	257	256
Maximum LOB length 1 470 000 000 (without INLINE LENGTH specified)	278	281	280
Maximum LOB length 2 147 483 647 (without INLINE LENGTH specified)	314	317	316
LOB with INLINE LENGTH specified	INLINE LENGTH + 2	INLINE LENGTH + 5	INLINE LENGTH + 4

¹ There is an additional 2 bytes of storage used by each row when VALUE COMPRESSION is active for that row.

²The LONG VARCHAR and LONG VARGRAPHIC data types are supported but are deprecated and might be removed in a future release.

³ Each LOB value has a *LOB descriptor* in the base record that points to the location of the actual value. The size of the descriptor varies according to the maximum length defined for the column.

For a *distinct type*, the byte count is equivalent to the length of the source type of the distinct type. For a *reference type*, the byte count is equivalent to the length of the built-in data type on which the reference type is based. For a *structured type*, the byte count is equivalent to the `INLINE LENGTH + 4`. The `INLINE LENGTH` is the value specified (or implicitly calculated) for the column in the *column-options* clause.

The row sizes for the following sample tables assume that VALUE COMPRESSION is not specified:

```
DEPARTMENT 63 (0 + 3 + 33 + 7 + 3 + 17)
ORG         57 (0 + 3 + 19 + 2 + 15 + 18)
```

If VALUE COMPRESSION were to be specified, the row sizes would change to:

```
DEPARTMENT 69 (2 + 5 + 31 + 8 + 5 + 18)
ORG         53 (2 + 4 + 16 + 4 + 12 + 15)
```

- **Storage byte counts:** The following table contains the storage byte counts of columns by data type for data values. The byte counts depend on whether or not VALUE COMPRESSION is active. When VALUE COMPRESSION is not active, the byte counts also depend on whether or not the column is nullable. The values in the table represent the amount of storage (in bytes) that is used to store the value.

Table 13. Storage Byte Counts Based on Row Format, Data Type, and Data Value

Data value →	NULL	NULL	zero-length	system default ²	all other data values	all other data values	all other data values
VALUE COMPRESSION →	not active	active ¹	active ¹	active ¹	not active	not active	active ¹
Column nullability →	nullable	nullable	n/a	n/a	nullable	not nullable	n/a
Data type							
SMALLINT	3	3	-	3	3	2	4
INTEGER	5	3	-	3	5	4	6
BIGINT	9	3	-	3	9	8	10
REAL	5	3	-	3	5	4	6
DOUBLE	9	3	-	3	9	8	10
DECIMAL	The integral part of $(p/2)+2$, where p is the precision	3	-	3	The integral part of $(p/2)+2$, where p is the precision	The integral part of $(p/2)+1$, where p is the precision	The integral part of $(p/2)+3$, where p is the precision
DECFLOAT(16)	9	3	-	3	9	8	10
DECFLOAT(34)	17	3	-	3	17	16	18
CHAR(n)	$n+1$	3	-	3	$n+1$	n	$n+2$
VARCHAR(n)	5	3	2	2	$N+5$, where N is the number of bytes in the data	$N+4$, where N is the number of bytes in the data	$N+2$, where N is the number of bytes in the data
LONG VARCHAR ³	5	3	2	2	25	24	22
GRAPHIC(n)	$n*2+1$	3	-	3	$n*2+1$	$n*2$	$n*2+2$
VARGRAPHIC(n)	5	3	2	2	$N*2+5$, where N is the number of bytes in the data	$N*2+4$, where N is the number of bytes in the data	$N*2+2$, where N is the number of bytes in the data
LONG VARGRAPHIC ³	5	3	2	2	25	24	22
DATE	5	3	-	-	5	4	6
TIME	4	3	-	-	4	3	5
TIMESTAMP(p)	The integral part of $(p+1)/2+8$, where p is the precision of fractional seconds	3	-	-	The integral part of $(p+1)/2+8$, where p is the precision of fractional seconds	The integral part of $(p+1)/2+7$, where p is the precision of fractional seconds	The integral part of $(p+1)/2+9$, where p is the precision of fractional seconds
Maximum LOB ² length 1024	5	3	2	2	$(60 \text{ to } 68)+5$	$(60 \text{ to } 68)+4$	$(60 \text{ to } 68)+2$
Maximum LOB length 8192	5	3	2	2	$(60 \text{ to } 92)+5$	$(60 \text{ to } 92)+4$	$(60 \text{ to } 92)+2$
Maximum LOB length 65 536	5	3	2	2	$(60 \text{ to } 116)+5$	$(60 \text{ to } 116)+4$	$(60 \text{ to } 116)+2$
Maximum LOB length 524 000	5	3	2	2	$(60 \text{ to } 140)+5$	$(60 \text{ to } 140)+4$	$(60 \text{ to } 140)+2$

Table 13. Storage Byte Counts Based on Row Format, Data Type, and Data Value (continued)

Data value →	NULL	NULL	zero-length	system default ²	all other data values	all other data values	all other data values
VALUE COMPRESSION →	not active	active ¹	active ¹	active ¹	not active	not active	active ¹
Column nullability →	nullable	nullable	n/a	n/a	nullable	not nullable	n/a
Data type							
Maximum LOB length 4 190 000	5	3	2	2	(60 to 164)+5	(60 to 164)+4	(60 to 164)+2
Maximum LOB length 134 000 000	5	3	2	2	(60 to 196)+5	(60 to 196)+4	(60 to 196)+2
Maximum LOB length 536 000 000	5	3	2	2	(60 to 220)+5	(60 to 220)+4	(60 to 220)+2
Maximum LOB length 1 070 000 000	5	3	2	2	(60 to 252)+5	(60 to 252)+4	(60 to 252)+2
Maximum LOB length 1 470 000 000	5	3	2	2	(60 to 276)+5	(60 to 276)+4	(60 to 276)+2
Maximum LOB length 2 147 483 647	5	3	2	2	(60 to 312)+5	(60 to 312)+4	(60 to 312)+2
XML	5	3	-	-	85	84	82

¹ There is an additional 2 bytes of storage used by each row when VALUE COMPRESSION is active for that row.

² When COMPRESS SYSTEM DEFAULT is specified for the column.

³The LONG VARCHAR and LONG VARGRAPHIC data types are supported but are deprecated and might be removed in a future release.

- **Dimension columns:** Because each distinct value of a dimension column is assigned to a different block of the table, clustering on an expression may be desirable, such as "INTEGER(ORDER_DATE)/100". In this case, a generated column can be defined for the table, and this generated column may then be used in the ORGANIZE BY DIMENSIONS clause. If the expression is monotonic with respect to a column of the table, DB2 may use the dimension index to satisfy range predicates on that column. For example, if the expression is simply *column-name + some-positive-constant*, it is monotonic increasing. User-defined functions, certain built-in functions, and using more than one column in an expression, prevent monotonicity or its detection.

Dimensions involving generated columns whose expressions are non-monotonic, or whose monotonicity cannot be determined, can still be created, but range queries along slice or cell boundaries of these dimensions are not supported. Equality and IN predicates *can* be processed by slices or cells.

A generated column is monotonic if the following is true with respect to the generating function, fn:

- Monotonic increasing.

For every possible pair of values x1 and x2, if x2>x1, then fn(x2)>fn(x1). For example:

SALARY - 10000

- Monotonic decreasing.
For every possible pair of values x_1 and x_2 , if $x_2 > x_1$, then $fn(x_2) < fn(x_1)$. For example:
-SALARY
- Monotonic non-decreasing.
For every possible pair of values x_1 and x_2 , if $x_2 > x_1$, then $fn(x_2) \geq fn(x_1)$. For example:
SALARY/1000
- Monotonic non-increasing.
For every possible pair of values x_1 and x_2 , if $x_2 > x_1$, then $fn(x_2) \leq fn(x_1)$. For example:
-SALARY/1000

The expression "PRICE*DISCOUNT" is not monotonic, because it involves more than one column of the table.

- **Range-clustered tables:** Organizing a table by key sequence is effective for certain types of tables. The table should have an integer key that is tightly clustered (dense) over the range of possible values. The columns of this integer key must not be nullable, and the key should logically be the primary key of the table. The organization of a range-clustered table precludes the need for a separate unique index object, providing direct access to the row for a specified key value, or a range of rows for a specified range of key values. The allocation of all the space for the complete set of rows in the defined key sequence range is done during table creation, and must be considered when defining a range-clustered table. The storage space is not available for any other use, even though the rows are initially marked deleted. If the full key sequence range will be populated with data only over a long period of time, this table organization may not be an appropriate choice.
- A table can have at most one security policy.
- DB2 enforces referential integrity constraints that are defined on protected tables. Constraints violations in this case can be difficult to debug, because DB2 will not allow you to see what row has caused a violation if you do not have the appropriate security label or exemptions credentials.
- When defining the order of columns in a table, frequently updated columns should be placed at the end of the definition to minimize the amount of data logged for updates. This includes ROW CHANGE TIMESTAMP columns. ROW CHANGE TIMESTAMP columns are guaranteed to be updated on each row update.
- **Security and replication:** Replication can cause data rows from a protected table to be replicated outside of the database. Care must be taken when setting up replication for a protected table, because DB2 cannot protect data that is outside of the database.
- **Compatibilities:** For compatibility with DB2 for z/OS:
 - The following syntax is accepted as the default behavior:
 - IN database-name.tablespace-name
 - IN DATABASE database-name
 - FOR MIXED DATA
 - FOR SBCS DATA
 - PART can be specified in place of PARTITION
 - PARTITION *partition-number* can be specified instead of PARTITION *partition-name*. A *partition-number* must not identify a partition that was

previously specified in the CREATE TABLE statement. If a *partition-number* is not specified, a unique partition number is generated by the database manager.

- VALUES can be specified in place of ENDING AT

For compatibility with previous versions of DB2 databases:

- The CONSTRAINT keyword can be omitted from a *column-definition* defining a references-clause
- *constraint-name* can be specified following FOREIGN KEY (without the CONSTRAINT keyword)
- SUMMARY can optionally be specified after CREATE
- DEFINITION ONLY can be specified in place of WITH NO DATA
- The PARTITIONING KEY clause can be specified in place of the DISTRIBUTE BY clause
- REPLICATED can be specified in place of DISTRIBUTE BY REPLICATION

For compatibility with previous versions of DB2 databases, and for consistency:

- A comma can be used to separate multiple options in the *identity-options* clause

The following syntax is also supported:

- NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER

Examples

Example 1: Create table TDEPT in the DEPARTX table space. DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT are column names. CHAR means the column will contain character data. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. The primary key consists of the column DEPTNO.

```
CREATE TABLE TDEPT
  (DEPTNO  CHAR(3)      NOT NULL,
   DEPTNAME VARCHAR(36) NOT NULL,
   MGRNO   CHAR(6),
   ADMRDEPT CHAR(3)    NOT NULL,
   PRIMARY KEY(DEPTNO))
IN DEPARTX
```

Example 2: Create table PROJ in the SCHED table space. PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF, PRSTDATE, PRENDATE, and MAJPROJ are column names. CHAR means the column will contain character data. DECIMAL means the column will contain packed decimal data. 5,2 means the following: 5 indicates the number of decimal digits, and 2 indicates the number of digits to the right of the decimal point. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. DATE means the column will contain date information in a three-part format (year, month, and day).

```
CREATE TABLE PROJ
  (PROJNO  CHAR(6)      NOT NULL,
   PROJNAME VARCHAR(24) NOT NULL,
   DEPTNO  CHAR(3)      NOT NULL,
   RESPEMP CHAR(6)      NOT NULL,
   PRSTAFF DECIMAL(5,2) ,
   PRSTDATE DATE        ,
   PRENDATE DATE        ,
   MAJPROJ CHAR(6)      NOT NULL)
IN SCHED
```

Example 3: Create a table called EMPLOYEE_SALARY where any unknown salary is considered 0. No table space is specified, so that the table will be created in a table space selected by the system based on the rules described for the *IN tablespace-name* clause.

```
CREATE TABLE EMPLOYEE_SALARY
(DEPTNO CHAR(3) NOT NULL,
DEPTNAME VARCHAR(36) NOT NULL,
EMPNO CHAR(6) NOT NULL,
SALARY DECIMAL(9,2) NOT NULL WITH DEFAULT)
```

Example 4: Create distinct types for total salary and miles and use them for columns of a table created in the default table space. In a dynamic SQL statement assume the CURRENT SCHEMA special register is JOHNDOE and the CURRENT PATH is the default ("SYSIBM","SYSFUN","JOHNDOE").

If a value for SALARY is not specified it must be set to 0 and if a value for LIVING_DIST is not specified it must be set to 1 mile.

```
CREATE TYPE JOHNDOE.T_SALARY AS INTEGER WITH COMPARISONS

CREATE TYPE JOHNDOE.MILES AS FLOAT WITH COMPARISONS

CREATE TABLE EMPLOYEE
(ID INTEGER NOT NULL,
NAME CHAR(30),
SALARY T_SALARY NOT NULL WITH DEFAULT,
LIVING_DIST MILES DEFAULT MILES(1) )
```

Example 5: Create distinct types for image and audio and use them for columns of a table. No table space is specified, so that the table will be created in a table space selected by the system based on the rules described for the *IN tablespace-name* clause. Assume the CURRENT PATH is the default.

```
CREATE TYPE IMAGE AS BLOB (10M)

CREATE TYPE AUDIO AS BLOB (1G)

CREATE TABLE PERSON
(SSN INTEGER NOT NULL,
NAME CHAR(30),
VOICE AUDIO,
PHOTO IMAGE)
```

Example 6: Create table EMPLOYEE in the HUMRES table space. The constraints defined on the table are the following:

- The values of department number must lie in the range 10 to 100.
- The job of an employee can only be either 'Sales', 'Mgr' or 'Clerk'.
- Every employee that has been with the company since 1986 must make more than \$40,500.

Note: If the columns included in the check constraints are nullable they could also be NULL.

```
CREATE TABLE EMPLOYEE
(ID SMALLINT NOT NULL,
NAME VARCHAR(9),
DEPT SMALLINT CHECK (DEPT BETWEEN 10 AND 100),
JOB CHAR(5) CHECK (JOB IN ('Sales','Mgr','Clerk')),
HIREDATE DATE,
SALARY DECIMAL(7,2),
COMM DECIMAL(7,2),
PRIMARY KEY (ID),
```

```

        CONSTRAINT YEARSAL CHECK (YEAR(HIREDATE) > 1986
        OR SALARY > 40500)
    )
    IN HUMRES

```

Example 7: Create a table that is wholly contained in the PAYROLL table space.

```

CREATE TABLE EMPLOYEE .....
    IN PAYROLL

```

Example 8: Create a table with its data part in ACCOUNTING and its index part in ACCOUNT_IDX.

```

CREATE TABLE SALARY.....
    IN ACCOUNTING INDEX IN ACCOUNT_IDX

```

Example 9: Create a table and log SQL changes in the default format.

```

CREATE TABLE SALARY1 .....

```

or

```

CREATE TABLE SALARY1 .....
    DATA CAPTURE NONE

```

Example 10: Create a table and log SQL changes in an expanded format.

```

CREATE TABLE SALARY2 .....
    DATA CAPTURE CHANGES

```

Example 11: Create a table EMP_ACT in the SCHED table space. EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, and EMENDATE are column names. Constraints defined on the table are:

- The value for the set of columns, EMPNO, PROJNO, and ACTNO, in any row must be unique.
- The value of PROJNO must match an existing value for the PROJNO column in the PROJECT table and if the project is deleted all rows referring to the project in EMP_ACT should also be deleted.

```

CREATE TABLE EMP_ACT
(EMPNO      CHAR(6) NOT NULL,
 PROJNO     CHAR(6) NOT NULL,
 ACTNO      SMALLINT NOT NULL,
 EMPTIME    DECIMAL(5,2),
 EMSTDATE   DATE,
 EMENDATE   DATE,
 CONSTRAINT EMP_ACT_UNIQ UNIQUE (EMPNO,PROJNO,ACTNO),
 CONSTRAINT FK_ACT_PROJ FOREIGN KEY (PROJNO)
                        REFERENCES PROJECT (PROJNO) ON DELETE CASCADE
)
    IN SCHED

```

A unique index called EMP_ACT_UNIQ is automatically created in the same schema to enforce the unique constraint.

Example 12: Create a table that is to hold information about famous goals for the ice hockey hall of fame. The table will list information about the player who scored the goal, the goaltender against who it was scored, the date and place, and a description. The description column is nullable.

```

CREATE TABLE HOCKEY_GOALS
( BY_PLAYER  VARCHAR(30) NOT NULL,
  BY_TEAM    VARCHAR(30) NOT NULL,

```

```

AGAINST_PLAYER VARCHAR(30) NOT NULL,
AGAINST_TEAM   VARCHAR(30) NOT NULL,
DATE_OF_GOAL   DATE       NOT NULL,
DESCRIPTION    CLOB(5000) )

```

Example 13: Suppose an exception table is needed for the EMPLOYEE table. One can be created using the following statement.

```

CREATE TABLE EXCEPTION_EMPLOYEE AS
(SELECT EMPLOYEE.*,
CURRENT_TIMESTAMP AS TIMESTAMP,
CAST (' ' AS CLOB(32K)) AS MSG
FROM EMPLOYEE
) WITH NO DATA

```

Example 14: Given the following table spaces with the indicated attributes:

TBSPACE	PAGESIZE	USER	USERAUTH
DEPT4K	4096	BOBBY	Y
PUBLIC4K	4096	PUBLIC	Y
DEPT8K	8192	BOBBY	Y
DEPT8K	8192	RICK	Y
PUBLIC8K	8192	PUBLIC	Y

- If RICK creates the following table, it is placed in table space PUBLIC4K since the byte count is less than 4005; but if BOBBY creates the same table, it is placed in table space DEPT4K, since BOBBY has USE privilege because of an explicit grant:

```

CREATE TABLE DOCUMENTS
(SUMMARY VARCHAR(1000),
REPORT VARCHAR(2000))

```

- If BOBBY creates the following table, it is placed in table space DEPT8K since the byte count is greater than 4005, and BOBBY has USE privilege because of an explicit grant. However, if DUNCAN creates the same table, it is placed in table space PUBLIC8K, since DUNCAN has no specific privileges:

```

CREATE TABLE CURRICULUM
(SUMMARY VARCHAR(1000),
REPORT VARCHAR(2000),
EXERCISES VARCHAR(1500))

```

Example 15: Create a table with a LEAD column defined with the structured type EMP. Specify an INLINE LENGTH of 300 bytes for the LEAD column, indicating that any instances of LEAD that cannot fit within the 300 bytes are stored outside the table (separately from the base table row, similar to the way LOB values are handled).

```

CREATE TABLE PROJECTS (PID INTEGER,
LEAD EMP INLINE LENGTH 300,
STARTDATE DATE,
... )

```

Example 16: Create a table DEPT with five columns named DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, and LOCATION. Column DEPT is to be defined as an IDENTITY column such that DB2 will always generate a value for it. The values for the DEPT column should begin with 500 and increment by 1.

```

CREATE TABLE DEPT
(DEPTNO SMALLINT NOT NULL
GENERATED ALWAYS AS IDENTITY
(START WITH 500, INCREMENT BY 1),

```

```

DEPTNAME  VARCHAR(36)  NOT NULL,
MGRNO     CHAR(6),
ADMRDEPT  SMALLINT    NOT NULL,
LOCATION    CHAR(30)

```

Example 17: Create a SALES table that is distributed on the YEAR column, and that has dimensions on the REGION and YEAR columns. Data will be distributed across database partitions according to hashed values of the YEAR column. On each database partition, data will be organized into extents based on unique combinations of values of the REGION and YEAR columns on those database partitions.

```

CREATE TABLE SALES
(CUSTOMER  VARCHAR(80),
 REGION    CHAR(5),
 YEAR      INTEGER)
DISTRIBUTE BY HASH (YEAR)
ORGANIZE BY DIMENSIONS (REGION, YEAR)

```

Example 18: Create a SALES table with a PURCHASEYEARMONTH column that is generated from the PURCHASEDATE column. Use an expression to create a column that is monotonic with respect to the original PURCHASEDATE column, and is therefore suitable for use as a dimension. The table is distributed on the REGION column, and organized within each database partition into extents according to the PURCHASEYEARMONTH column; that is, different regions will be on different database partitions, and different purchase months will belong to different cells (or sets of extents) within those database partitions.

```

CREATE TABLE SALES
(CUSTOMER          VARCHAR(80),
 REGION            CHAR(5),
 PURCHASEDATE      DATE,
 PURCHASEYEARMONTH INTEGER
GENERATED ALWAYS AS (INTEGER(PURCHASEDATE)/100))
DISTRIBUTE BY HASH (REGION)
ORGANIZE BY DIMENSIONS (PURCHASEYEARMONTH)

```

Example 19: Create a CUSTOMER table with a CUSTOMERNUMDIM column that is generated from the CUSTOMERNUM column. Use an expression to create a column that is monotonic with respect to the original CUSTOMERNUM column, and is therefore suitable for use as a dimension. The table is organized into cells according to the CUSTOMERNUMDIM column, so that there is a different cell in the table for every 50 customers. If a unique index were created on CUSTOMERNUM, customer numbers would be clustered in such a way that each set of 50 values would be found in a particular set of extents in the table.

```

CREATE TABLE CUSTOMER
(CUSTOMERNUM      INTEGER,
 CUSTOMERNAME     VARCHAR(80),
 ADDRESS          VARCHAR(200),
 CITY             VARCHAR(50),
 COUNTRY          VARCHAR(50),
 CODE             VARCHAR(15),
 CUSTOMERNUMDIM   INTEGER
GENERATED ALWAYS AS (CUSTOMERNUM/50))
ORGANIZE BY DIMENSIONS (CUSTOMERNUMDIM)

```

Example 20: Create a remote base table called EMPLOYEE on the Oracle server, ORASERVER. A nickname, named EMPLOYEE, which refers to this newly created remote base table, will also automatically be created.

```

CREATE TABLE EMPLOYEE
(EMP_NO  CHAR(6)  NOT NULL,
 FIRST_NAME VARCHAR(12) NOT NULL,

```

```

MID_INT      CHAR(1)      NOT NULL,
LAST_NAME    VARCHAR(15)  NOT NULL,
HIRE_DATE    DATE,
JOB          CHAR(8),
SALARY       DECIMAL(9,2),
PRIMARY KEY (EMP_NO))
OPTIONS
(REMOTE_SERVER 'ORASERVER',
REMOTE_SCHEMA 'J15USER1',
REMOTE_TABNAME 'EMPLOYEE')

```

The following CREATE TABLE statements show how to specify the table name, or the table name and the explicit remote base table name, to get the desired case. The lowercase identifier, employee, is used to illustrate the implicit folding of identifiers.

Create a remote base table called EMPLOYEE (uppercase characters) on an Informix server, and create a nickname named EMPLOYEE (uppercase characters) on that table:

```

CREATE TABLE employee
(EMP_NO CHAR(6) NOT NULL,
...)
OPTIONS
(REMOTE_SERVER 'INFX_SERVER')

```

If the REMOTE_TABNAME option is not specified, and *table-name* is not delimited, the remote base table name will be in uppercase characters, even if the remote data source normally stores names in lowercase characters.

Create a remote base table called employee (lowercase characters) on an Informix server, and create a nickname named EMPLOYEE (uppercase characters) on that table:

```

CREATE TABLE employee
(EMP_NO CHAR(6) NOT NULL,
...)
OPTIONS
(REMOTE_SERVER 'INFX_SERVER',
REMOTE_TABNAME 'employee')

```

When creating a table at a remote data source that supports delimited identifiers, use the REMOTE_TABNAME option and a character string constant that specifies the table name in the desired case.

Create a remote base table called employee (lowercase characters) on an Informix server, and create a nickname named employee (lowercase characters) on that table:

```

CREATE TABLE "employee"
(EMP_NO CHAR(6) NOT NULL,
...)
OPTIONS
(REMOTE_SERVER 'INFX_SERVER')

```

If the REMOTE_TABNAME option is not specified, and *table-name* is delimited, the remote base table name will be identical to *table-name*.

Example 21: Create a range-clustered table that can be used to locate a student using a student ID. For each student record, include the school ID, program ID, student number, student ID, student first name, student last name, and student grade point average (GPA).


```

CREATE TABLE STUDENTS
(SCHOOL_ID    INTEGER    NOT NULL,
PROGRAM_ID    INTEGER    NOT NULL,
STUDENT_NUM   INTEGER    NOT NULL,
STUDENT_ID    INTEGER    NOT NULL,
FIRST_NAME    CHAR(30),
LAST_NAME     CHAR(30),
GPA           DOUBLE)
ORGANIZE BY KEY SEQUENCE
(STUDENT_ID
STARTING FROM 1
ENDING AT 1000000)
DISALLOW OVERFLOW

```

The size of each record is the sum of the columns, plus alignment, plus the range-clustered table row header. In this case, the row size is 98 bytes: 4 + 4 + 4 + 4 + 30 + 30 + 8 + 3 (for nullable columns) + 1 (for alignment) + 10 (for the header). With a 4-KB page size (or 4096 bytes), after accounting for page overhead, there are 4038 bytes available, enough room for 41 records per page. Allowing for 1 million student records, there is a need for (1 million divided by 41 records per page) 24 391 pages. With two additional pages for table overhead, the final number of 4-KB pages that are allocated when the table is created is 24 393.

Example 22: Create a table named DEPARTMENT with a functional dependency that has no specified constraint name.

```

CREATE TABLE DEPARTMENT
(DEPTNO       SMALLINT    NOT NULL,
DEPTNAME     VARCHAR(36) NOT NULL,
MGRNO        CHAR(6),
ADMNDEPT     SMALLINT    NOT NULL,
LOCATION       CHAR(30),
CHECK (DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED)

```

Example 23: Create a table with protected rows.

```

CREATE TABLE TOASTMASTERS
(PERFORMANCE DB2SECURITYLABEL,
POINTS       INTEGER,
NAME         VARCHAR(50))
SECURITY POLICY CONTRIBUTIONS

```

Example 24: Create a table with protected columns.

```

CREATE TABLE TOASTMASTERS
(PERFORMANCE CHAR(8),
POINTS       INTEGER COLUMN SECURED WITH CLUBPOSITION,
NAME         VARCHAR(50))
SECURITY POLICY CONTRIBUTIONS

```

Example 25: Create a table with protected rows and columns.

```

CREATE TABLE TOASTMASTERS
(PERFORMANCE DB2SECURITYLABEL,
POINTS       INTEGER COLUMN SECURED WITH CLUBPOSITION,
NAME         VARCHAR(50))
SECURITY POLICY CONTRIBUTIONS

```

Example 26: Large objects for a partitioned table reside, by default, in the same table space as the data. This default behavior can be overridden by using the LONG IN clause to specify one or more table spaces for the large objects. Create a table named DOCUMENTS whose large object data is to be stored (in a round-robin fashion for each data partition) in table spaces TBSP1 and TBSP2.

```

CREATE TABLE DOCUMENTS
  (ID INTEGER,
   CONTENTS CLOB)
LONG IN TBSP1, TBSP2
PARTITION BY RANGE (ID)
  (STARTING 1 ENDING 1000
   EVERY 100)

```

Alternatively, use the long form of the syntax to explicitly identify a large table space for each data partition. In this example, the CLOB data for the first data partition is placed in LARGE_TBSP3, and the CLOB data for the remaining data partitions is spread across LARGE_TBSP1 and LARGE_TBSP2 in a round-robin fashion.

```

CREATE TABLE DOCUMENTS
  (ID INTEGER,
   CONTENTS CLOB)
LONG IN LARGE_TBSP1, LARGE_TBSP2
PARTITION BY RANGE (ID)
  (STARTING 1 ENDING 100
   IN TBSP1 LONG IN LARGE_TBSP3,
   STARTING 101 ENDING 1000
   EVERY 100)

```

Example 27: Create a partitioned table named ACCESSNUMBERS having two data partitions. The row (10, NULL) is to be placed in the first partition, and the row (NULL, 100) is to be placed in the second (last) data partition.

```

CREATE TABLE ACCESSNUMBERS
  (AREA INTEGER,
   EXCHANGE INTEGER)
PARTITION BY RANGE (AREA NULLS LAST, EXCHANGE NULLS FIRST)
  (STARTING (1,1) ENDING (10,100),
   STARTING (11,1) ENDING (MAXVALUE,MAXVALUE))

```

Because null values in the second column are sorted first, the row (11, NULL) would sort below the low boundary of the last data partition (11, 1); attempting to insert this row returns an error. The row (12, NULL) would fall within the last data partition.

Example 28: Create a table named RATIO having a single data partition and partitioning column PERCENT.

```

CREATE TABLE RATIO
  (PERCENT INTEGER)
PARTITION BY RANGE (PERCENT)
  (STARTING (MINVALUE) ENDING (MAXVALUE))

```

This table definition allows any integer value for column PERCENT to be inserted. The following definition for the RATIO table allows any integer value between 1 and 100 inclusive to be inserted into column PERCENT.

```

CREATE TABLE RATIO
  (PERCENT INTEGER)
PARTITION BY RANGE (PERCENT)
  (STARTING 0 EXCLUSIVE ENDING 100 INCLUSIVE)

```

Example 29: Create a table named MYDOCS with two columns: one is an identifier, and the other stores XML documents.

```

CREATE TABLE MYDOCS
  (ID INTEGER,
   DOC XML)
IN HLTBSPACE

```

Example 30: Create a table named NOTES with four columns, including one for storing XML-based notes.

```
CREATE TABLE NOTES
  (ID          INTEGER,
   DESCRIPTION VARCHAR(255),
   CREATED     TIMESTAMP,
   NOTE        XML)
```

Example 31: Create a table, EMP_INFO, that contains a phone number and address for each employee. Include a ROW CHANGE TIMESTAMP column in the table to track the modification of employee information.

```
CREATE TABLE EMP_INFO
  (EMPNO          CHAR(6) NOT NULL,
   EMP_INFOCHANGE NOT NULL GENERATED ALWAYS
   FOR EACH ROW ON UPDATE
   AS ROW CHANGE TIMESTAMP,
   EMP_ADDRESS    VARCHAR(300),
   EMP_PHONENO    CHAR(4),
   PRIMARY KEY (EMPNO ) )
```

Example 32: Create a partitioned table named DOCUMENTS having two data partitions:

- The data object in the first partition resides in table space TBSP11. The partitioned index partition on the partition resides in table space TBSP21. The XML data object resides in table space TBSP31.
- The data object in the second partition resides in table space TBSP12. The partitioned index partition on the partition resides in table space TBSP22. The XML data object resides in table space TBSP32.

The table level INDEX IN clause has no impact on table space selection for partitioned indexes.

```
CREATE TABLE DOCUMENTS
  (ID          INTEGER,
   CONTENTS    XML) INDEX IN TBSPX
 PARTITION BY (ID)
 (STARTING 1 ENDING 100 IN TBSP11 INDEX IN TBSP21 LONG IN TBSP31,
  STARTING 101 ENDING 101 IN TBSP21 INDEX IN TBSP22 LONG IN TBSP32 );
```

Example 33: Create a partitioned table named SALES having two data partitions:

- The data object in the first partition resides in table space TBSP11. The partitioned index partition on the partition resides in table space TBSP21.
- The data object in the second partition resides in table space TBSP12. The partitioned index object resides in table space TBSP22.

The table level INDEX IN clause has no impact on table space selection for partitioned indexes.

```
CREATE TABLE SALES
  (SID          INTEGER,
   AMOUNT       INTEGER) INDEX IN TBSPX
 PARTITION BY (SID)
 (STARTING 1 ENDING 100 IN TBSP11 INDEX IN TBSP21,
  STARTING 101 ENDING 101 IN TBSP12 INDEX IN TBSP22);
```

CREATE TABLESPACE

The CREATE TABLESPACE statement defines a new table space within the database, assigns containers to the table space, and records the table space definition and attributes in the catalog.

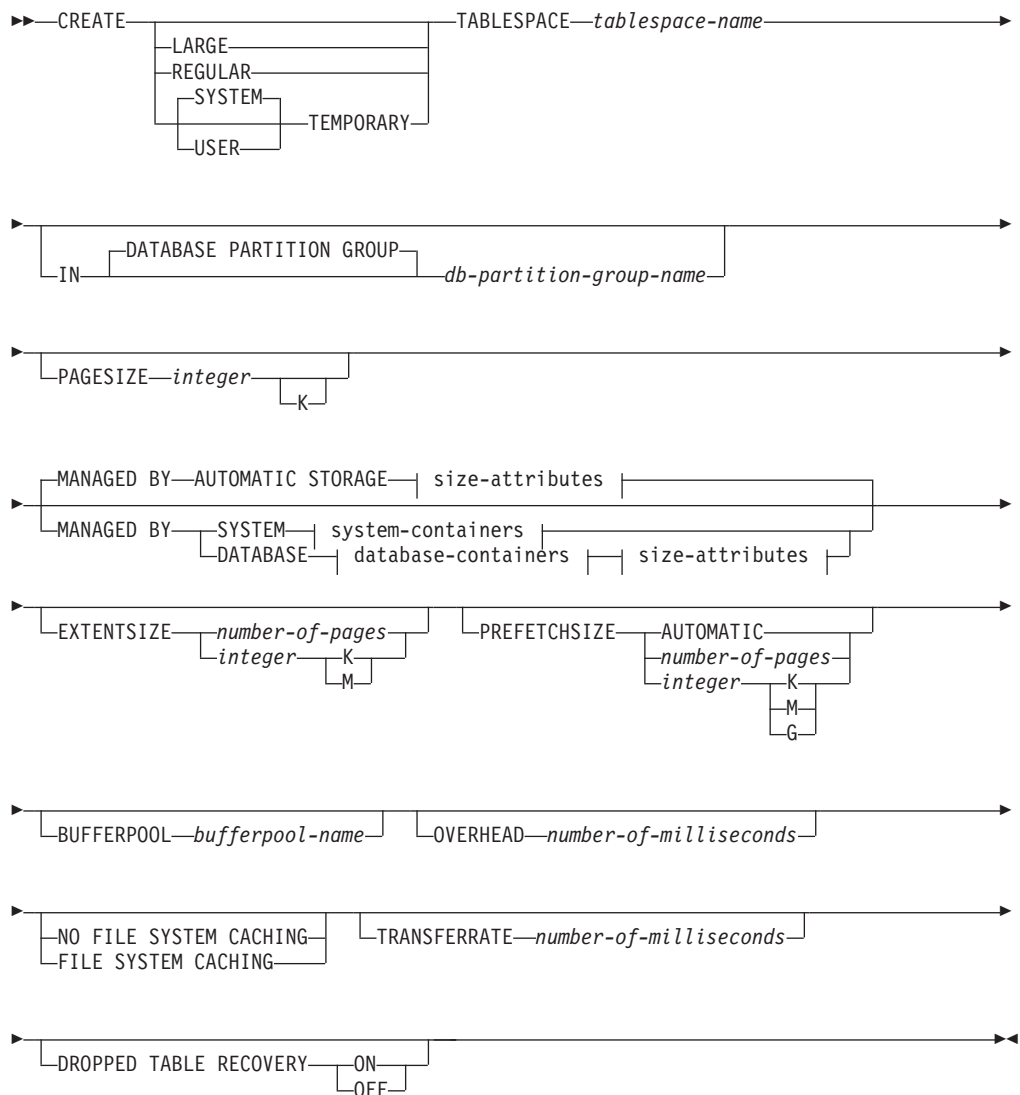
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

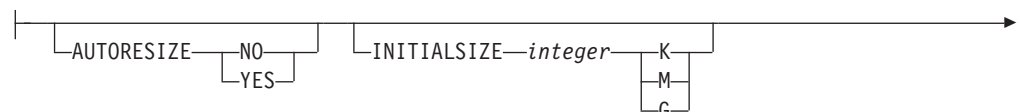
Authorization

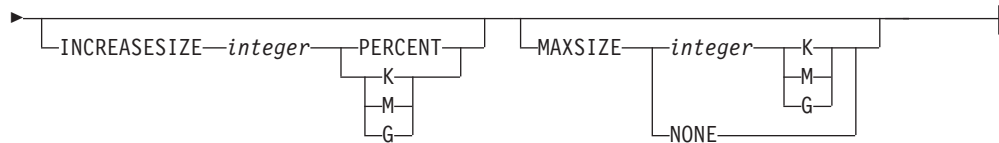
The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax

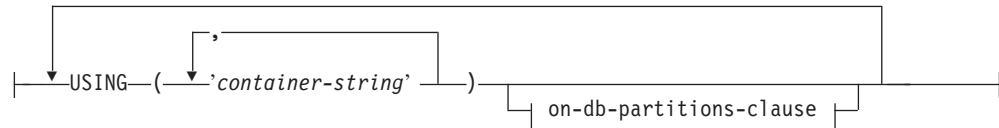


size-attributes:

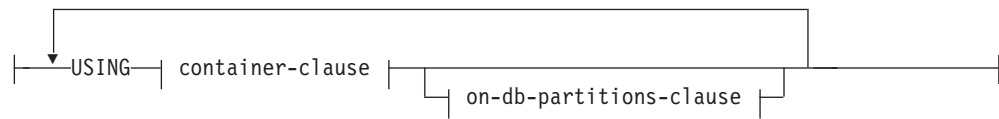




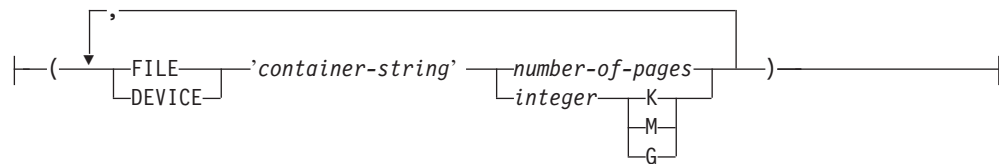
system-containers:



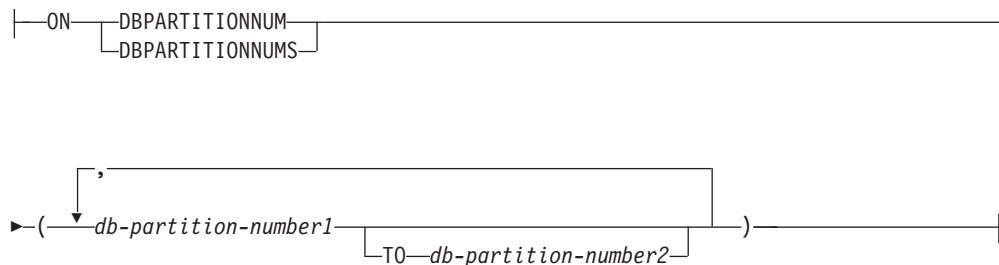
database-containers:



container-clause:



on-db-partitions-clause:



Description

LARGE, REGULAR, SYSTEM TEMPORARY, or USER TEMPORARY

Specifies the type of table space that is to be created. If no type is specified, the default is determined by the MANAGED BY clause.

LARGE

Stores all permanent data. This type is only allowed on database managed space (DMS) table spaces. It is also the default type for DMS table spaces when no type is specified. When a table is placed in a large table space:

- The table can be larger than a table in a regular table space. For details on table and table space limits, see “SQL limits”.

- The table can support more than 255 rows per data page, which can improve space utilization on data pages.
- Indexes that are defined on the table will require an additional 2 bytes per row entry, compared to indexes defined on a table that resides in a regular table space.

REGULAR

Stores all permanent data. This type applies to both DMS and SMS table spaces. This is the only type allowed for SMS table spaces, and it is also the default type for SMS table spaces when no type is specified.

SYSTEM TEMPORARY

Stores temporary tables, work areas used by the database manager to perform operations such as sorts or joins. A database must always have at least one SYSTEM TEMPORARY table space, because temporary tables can only be stored in such a table space. A temporary table space is created automatically when a database is created.

USER TEMPORARY

Stores created temporary tables and declared temporary tables. No user temporary table spaces exist when a database is created. To allow the definition of created temporary tables or declared temporary tables, at least one user temporary table space should be created with appropriate USE privileges.

tablespace-name

Names the table space. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *tablespace-name* must not identify a table space that already exists in the catalog (SQLSTATE 42710). The *tablespace-name* must not begin with the characters 'SYS' (SQLSTATE 42939).

IN DATABASE PARTITION GROUP *db-partition-group-name*

Specifies the database partition group for the table space. The database partition group must exist. The only database partition group that can be specified when creating a SYSTEM TEMPORARY table space is IBMTEMPGROUP. The DATABASE PARTITION GROUP keywords are optional.

If the database partition group is not specified, the default database partition group (IBMDEFAULTGROUP) is used for REGULAR, LARGE, and USER TEMPORARY table spaces. For SYSTEM TEMPORARY table spaces, the default database partition group IBMTEMPGROUP is used.

PAGESIZE *integer* [K]

Defines the size of pages used for the table space. The valid values for *integer* without the suffix K are 4 096, 8 192, 16 384, or 32 768. The valid values for *integer* with the suffix K are 4, 8, 16, or 32. Any number of spaces is allowed between *integer* and K, including no space. An error occurs if the page size is not one of these values (SQLSTATE 428DE), or if the page size is not the same as the page size of the buffer pool that is associated with the table space (SQLSTATE 428CB).

The default value is provided by the **pagesize** database configuration parameter, which is set when the database is created.

MANAGED BY AUTOMATIC STORAGE

Specifies that the table space is to be an automatic storage table space. If automatic storage is not defined for the database, an error is returned (SQLSTATE 55060).

An automatic storage table space is created as either a system managed space (SMS) table space or a database managed space (DMS) table space. When DMS is chosen and the type of table space is not specified, the default behavior is to create a large table space. With an automatic storage table space, the database manager determines which containers are to be assigned to the table space, based upon the storage paths that are associated with the database.

size-attributes

Specify the size attributes for an automatic storage table space or a DMS table space that is not an automatic storage table space. SMS table spaces are not auto-resizable.

AUTORESIZE

Specifies whether or not the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled. Auto-resizable table spaces automatically increase in size when they become full. The default is NO for DMS table spaces and YES for automatic storage table spaces.

NO

Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be disabled.

YES

Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled.

INITIALSIZE *integer* **K | M | G**

Specifies the initial size, per database partition, of an automatic storage table space. This option is only valid for automatic storage table spaces. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain a consistent size across containers in the table space. Moreover, if the table space is auto-resizable and the initial size is not large enough to contain meta-data that must be added to the new table space, the database manager will continue to extend the table space by the value of **INCREASESIZE** until there is enough space. If the **INITIALSIZE** clause is not specified, the database manager determines an appropriate value. The value for *integer* must be at least 48 K.

INCREASESIZE *integer* **PERCENT** or **INCREASESIZE** *integer* **K | M | G**

Specifies the amount, per database partition, by which a table space that is enabled for auto-resize will automatically be increased when the table space is full, and a request for space has been made. The integer value must be followed by:

- **PERCENT** to specify the amount as a percentage of the table space size at the time that a request for space is made. When **PERCENT** is specified, the integer value must be between 0 and 100 (SQLSTATE 42615).
- **K** (for kilobytes), **M** (for megabytes), or **G** (for gigabytes) to specify the amount in bytes

Note that the actual value used might be slightly smaller or larger than what was specified, because the database manager strives to maintain consistent growth across containers in the table space. If the table space is auto-resizable, but the **INCREASESIZE** clause is not specified, the database manager determines an appropriate value.

MAXSIZE *integer* **K | M | G** or **MAXSIZE NONE**

Specifies the maximum size to which a table space that is enabled for auto-resize can automatically be increased. If the table space is auto-resizable, but the MAXSIZE clause is not specified, the default is NONE.

integer

Specifies a hard limit on the size, per database partition, to which a DMS table space or an automatic storage table space can automatically be increased. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

NONE

Specifies that the table space is to be allowed to grow to file system capacity, or to the maximum table space size (described in “SQL limits”).

MANAGED BY SYSTEM

Specifies that the table space is to be an SMS table space. When the type of table space is not specified, the default behavior is to create a regular table space.

system-containers

Specify the containers for an SMS table space.

USING (*'container-string',...*)

For an SMS table space, identifies one or more containers that will belong to the table space and in which the table space data will be stored. The *container-string* cannot exceed 240 bytes in length.

Each *container-string* can be an absolute or relative directory name.

The directory name, if not absolute, is relative to the database directory, and can be a path name alias (a symbolic link on UNIX systems) to storage that is not physically associated with the database directory. For example, *<dbdir>/work/c1* could be a symbolic link to a separate file system.

If any component of the directory name does not exist, it is created by the database manager. When a table space is dropped, all components created by the database manager are deleted. If the directory identified by *container-string* exists, it must not contain any files or subdirectories (SQLSTATE 428B2).

The format of *container-string* is dependent on the operating system. On Windows operating systems, an absolute directory path name begins with a drive letter and a colon (:); on UNIX systems, an absolute path name begins with a forward slash (/). A relative path name on any platform does not begin with an operating system-dependent character.

Remote resources (such as LAN-redirected drives or NFS-mounted file systems) are currently only supported when using Network Appliance Filers, IBM iSCSI, IBM Network Attached Storage, Network Appliance iSCSI, NEC iStorage S2100, S2200, or S4100, or NEC Storage NS Series with a Windows DB2 server. Note that NEC Storage NS Series is only supported with the use of an uninterrupted power supply (UPS); continuous UPS (rather than standby) is recommended. An NFS-mounted file system on AIX must be mounted in uninterruptible mode using the -o nointr option.

on-db-partitions-clause

Specifies the database partition or partitions on which the containers are created in a partitioned database. If this clause is not specified, then the containers are created on the database partitions in the database partition group that are not explicitly specified in any other *on-db-partitions-clauses*. For a SYSTEM TEMPORARY table space defined on database partition group IBMTEMPGROUP, when the *on-db-partitions-clause* is not specified, the containers will also be created on all new database partitions added to the database.

MANAGED BY DATABASE

Specifies that the table space is to be a DMS table space. When the type of table space is not specified, the default behavior is to create a large table space.

database-containers

Specify the containers for a DMS table space.

USING

Introduces a container-clause.

container-clause

Specifies the containers for a DMS table space.

(FILE | DEVICE 'container-string' number-of-pages,...)

For a DMS table space, identifies one or more containers that will belong to the table space and in which the table space data will be stored. The type of the container (either FILE or DEVICE) and its size (in PAGESIZE pages) are specified. The size can also be specified as an integer value followed by K (for kilobytes), M (for megabytes) or G (for gigabytes). If specified in this way, the floor of the number of bytes divided by the pagesize is used to determine the number of pages for the container. A mixture of FILE and DEVICE containers can be specified. The *container-string* cannot exceed 254 bytes in length.

For a FILE container, *container-string* must be an absolute or relative file name. The file name, if not absolute, is relative to the database directory. If any component of the directory name does not exist, it is created by the database manager. If the file does not exist, it will be created and initialized to the specified size by the database manager. When a table space is dropped, all components created by the database manager are deleted.

Note: If the file exists, it is overwritten, and if it is smaller than specified, it is extended. The file will not be truncated if it is larger than specified.

For a DEVICE container, *container-string* must be a device name. The device must already exist.

All containers must be unique across all databases. A container can belong to only one table space. The size of the containers can differ; however, optimal performance is achieved when all containers are the same size. The exact format of *container-string* is dependent on the operating system.

Remote resources (such as LAN-redirected drives or NFS-mounted file systems) are currently only supported when using Network Appliance Filers, IBM iSCSI, IBM Network Attached Storage, Network Appliance iSCSI, NEC iStorage S2100, S2200, or S4100, or NEC Storage NS Series with a Windows DB2 server. Note that NEC Storage NS Series is only

supported with the use of an uninterrupted power supply (UPS); continuous UPS (rather than standby) is recommended..

on-db-partitions-clause

Specifies the database partition or partitions on which the containers are created in a partitioned database. If this clause is not specified, then the containers are created on the database partitions in the database partition group that are not explicitly specified in any other *on-db-partitions-clause*. For a SYSTEM TEMPORARY table space defined on database partition group IBMTEMPGROUP, when the *on-db-partitions-clause* is not specified, the containers will also be created on all new database partitions added to the database.

on-db-partitions-clause

Specifies the database partitions on which containers are created in a partitioned database.

ON DBPARTITIONNUMS

Keywords indicating that individual database partitions are specified. DBPARTITIONNUM is a synonym for DBPARTITIONNUMS.

db-partition-number1

Specify a database partition number.

TO *db-partition-number2*

Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9). Containers are to be created on each database partition between and including the specified values. A specified database partition must be in the database partition group for the table space.

The database partition specified by number, and every database partition within the specified range of database partitions must exist in the database partition group for the table space (SQLSTATE 42729). A database partition number can only appear explicitly or within a range in exactly one *on-db-partitions-clause* for the statement (SQLSTATE 42613).

EXTENTSIZE *number-of-pages*

Specifies the number of PAGESIZE pages that will be written to a container before skipping to the next container. The extent size value can also be specified as an integer value followed by K (for kilobytes) or M (for megabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the value for the extent size. The database manager cycles repeatedly through the containers as data is stored.

The default value is provided by the **dft_extent_sz** database configuration parameter, which has a valid range of 2-256 pages.

PREFETCHSIZE

Specifies to read in data needed by a query prior to it being referenced by the query, so that the query need not wait for I/O to be performed.

The default value is provided by the **dft_prefetch_sz** database configuration parameter.

AUTOMATIC

Specifies that the prefetch size of a table space is to be updated automatically; that is, the prefetch size will be managed by DB2, using the following formula:

Prefetch size =
(number of containers) *
(number of physical disks per container) *
(extent size)

The number of physical disks per container defaults to 1, unless a value is specified through the DB2_PARALLEL_IO registry variable.

DB2 will update the prefetch size automatically whenever the number of containers in a table space changes (following successful execution of an ALTER TABLESPACE statement that adds or drops one or more containers). The prefetch size is updated at database start-up.

number-of-pages

Specifies the number of PAGESIZE pages that will be read from the table space when data prefetching is being performed. The prefetch size value can also be specified as an integer value followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the number of pages value for prefetch size.

BUFFERPOOL *bufferpool-name*

The name of the buffer pool used for tables in this table space. The buffer pool must exist (SQLSTATE 42704). If not specified, the default buffer pool (IBMDEFAULTBP) is used. The page size of the buffer pool must match the page size specified (or defaulted) for the table space (SQLSTATE 428CB). The database partition group of the table space must be defined for the buffer pool (SQLSTATE 42735).

OVERHEAD *number-of-milliseconds*

Specifies the I/O controller overhead and disk seek and latency time. This value is used to determine the cost of I/O during query optimization. The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all containers, the number should be the average for all containers that belong to the table space.

For a database that was created in Version 9 or later, the default I/O controller overhead and disk seek and latency time is 7.5 milliseconds. For a database that was upgraded from a previous version of DB2 to Version 9 or later, the default is 12.67 milliseconds.

FILE SYSTEM CACHING or NO FILE SYSTEM CACHING

Specifies whether or not I/O operations are to be cached at the file system level. If neither option is specified, the default is:

- FILE SYSTEM CACHING for JFS on AIX, Linux System z[®], all non-VxFS file systems on Solaris, HP-UX, SMS temporary table space files on all platforms, and all LOB and large data
- NO FILE SYSTEM CACHING on all other platforms and file system types

FILE SYSTEM CACHING

Specifies that all I/O operations in the target table space are to be cached at the file system level.

NO FILE SYSTEM CACHING

Specifies that all I/O operations are to bypass the file system-level cache.

TRANSFERRATE *number-of-milliseconds*

Specifies the time to read one page into memory. This value is used to

determine the cost of I/O during query optimization. The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all containers, the number should be the average for all containers that belong to the table space.

For a database that was created in Version 9 or later, the default time to read one page into memory is 0.06 milliseconds. For a database that was upgraded from a previous version of DB2 to Version 9 or later, the default is 0.18 milliseconds.

DROPPED TABLE RECOVERY

Indicates whether dropped tables in the specified table space can be recovered using the RECOVER DROPPED TABLE option of the ROLLFORWARD DATABASE command. This clause can only be specified for a regular or large table space (SQLSTATE 42613).

ON

Specifies that dropped tables can be recovered. This has been the default since Version 8.

OFF

Specifies that dropped tables cannot be recovered. This is the default in Version 7.

Rules

- If automatic storage is not defined for the database, an error is returned (SQLSTATE 55060).
- The INITIALSIZE clause cannot be specified with the MANAGED BY SYSTEM or MANAGED BY DATABASE clause (SQLSTATE 42601).
- The AUTORESIZE, INCREASESIZE, or MAXSIZE clause cannot be specified with the MANAGED BY SYSTEM clause (SQLSTATE 42601).
- The AUTORESIZE, INITIALSIZE, INCREASESIZE, or MAXSIZE clause cannot be specified for the creation of a temporary automatic storage table space (SQLSTATE 42601).
- The INCREASESIZE or MAXSIZE clause cannot be specified if the table space is not auto-resizable (SQLSTATE 42601).
- AUTORESIZE cannot be enabled for DMS table spaces that are defined to use raw device containers (SQLSTATE 42601).
- A table space must initially be large enough to hold five extents (SQLSTATE 57011).
- The maximum size of a table space must be larger than its initial size (SQLSTATE 560B0).
- Container operations (ADD, EXTEND, RESIZE, DROP, or BEGIN NEW STRIPE SET) cannot be performed on automatic storage table spaces, because the database manager is controlling the space management of such table spaces (SQLSTATE 42858).
- Each container definition requires 53 bytes plus the number of bytes necessary to store the container name. The combined length of all container names for the table space cannot exceed 20 480 bytes (SQLSTATE 54034).
- For a partitioned database, if more than one database partition resides on the same physical node, the same device or path cannot be specified for more than one database partition (SQLSTATE 42730). In this environment, either specify a unique *container-string* for each database partition, or use a relative path name.

Notes

- Choosing between a database-managed space or a system-managed space for a table space is a fundamental choice involving trade-offs.
- When more than one TEMPORARY table space exists in the database, they are used in round-robin fashion to balance their usage.
- The owner of the table space is granted USE privilege with the WITH GRANT OPTION on the table space when it is created.
- You can specify a database partition expression for container string syntax when creating either SMS or DMS containers. You would typically specify the database partition expression when using multiple logical database partitions in the partitioned database system. This ensures that container names are unique across database partition servers. When the expression is specified, the database partition number is part of the container name or, if additional arguments are specified, the result of the argument is part of the container name.

You use the argument “ \$N” ([blank]\$N) to indicate a database partition expression. A database partition expression can be used anywhere in the container name, and multiple database partition expressions can be specified. Terminate the database partition expression with a space character; whatever follows the space is appended to the container name after the database partition expression is evaluated. If there is no space character in the container name after the database partition expression, it is assumed that the rest of the string is part of the expression. The argument can only be used in one of the following forms.

Table 14. Arguments for Creating Containers. Operators are evaluated from left to right. The database partition number in the examples is assumed to be 5.

Syntax	Example	Value
[blank]\$N	" \$N"	5
[blank]\$N+[number]	" \$N+1011"	1016
[blank]\$N%[number]	" \$N%3" ^a	2
[blank]\$N+[number]%[number]	" \$N+12%13"	4
[blank]\$N%[number]+[number]	" \$N%3+20"	22
^a % represents the modulus operator.		

For example:

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
(device '/dev/rcont $N' 20000)
```

On a two database partition system, the following containers would be created:

```
/dev/rcont0 - on DATABASE PARTITION 0
/dev/rcont1 - on DATABASE PARTITION 1
```

```
CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
(file '/DB2/containers/TS2/container $N+100' 10000)
```

On a four database partition system, the following containers would be created:

```
/DB2/containers/TS2/container100 - on DATABASE PARTITION 0
/DB2/containers/TS2/container101 - on DATABASE PARTITION 1
/DB2/containers/TS2/container102 - on DATABASE PARTITION 2
/DB2/containers/TS2/container103 - on DATABASE PARTITION 3
```

```
CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
```



```
('/TS3/cont $N%2','/TS3/cont $N%2+2')
```

On a two database partition system, the following containers would be created:

```
/TS3/cont0 - On DATABASE PARTITION 0  
/TS3/cont2 - On DATABASE PARTITION 0  
/TS3/cont1 - On DATABASE PARTITION 1  
/TS3/cont3 - On DATABASE PARTITION 1
```

If database partition = 5, the containers:

```
'/dbdir/node $N /cont1'  
'/ $N+1000 /file1'  
' $N%10 /container'  
'/dir/ $N2000 /dmscont'
```

are created as:

```
'/dbdir/node5/cont1'  
'/1005/file1'  
'5/container'  
'/dir/2000/dmscont'
```

- An automatic storage table space is created as either an SMS table space or a DMS table space. DMS is chosen for large and regular table spaces, and SMS is chosen for temporary table spaces. Note that this behavior cannot be depended upon, because it might change in a future release. When DMS is chosen and the type of table space is not specified, the default behavior is to create a large table space.
- The creation of an automatic storage table space does not include container definitions. The database manager automatically determines the location and size, if applicable, of the containers on the basis of the storage paths that are associated with the database. The database manager will attempt to grow large and regular table spaces, as necessary, provided that the maximum size has not been reached. This might involve extending existing containers or adding containers to a new stripe set. Every time that the database is activated, the database manager automatically reconfigures the number and location of the containers for temporary table spaces that are not in an abnormal state.
- A large or regular automatic storage table space will not use new storage paths (see the description of the ALTER DATABASE statement) until there is no more space in one of the existing storage paths that the table space is using. Temporary automatic storage table spaces can only use the new storage paths once the database has been deactivated and then reactivated.
- **Compatibilities:** For compatibility with previous versions of DB2 databases:
 - NODE can be specified in place of DBPARTITIONNUM
 - NODES can be specified in place of DBPARTITIONNUMS
 - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
 - LONG can be specified in place of LARGE

Examples

Example 1: Create a large DMS table space on a UNIX system using three devices of 10 000 4K pages each. Specify their I/O characteristics.

```
CREATE TABLESPACE PAYROLL  
  MANAGED BY DATABASE  
  USING (DEVICE '/dev/rhdisk6' 10000,  
        DEVICE '/dev/rhdisk7' 10000,  
        DEVICE '/dev/rhdisk8' 10000)  
  OVERHEAD 12.67  
  TRANSFERRATE 0.18
```

Example 2: Create a regular SMS table space on Windows using three directories on three separate drives, with a 64-page extent size, and a 32-page prefetch size.

```
CREATE TABLESPACE ACCOUNTING
  MANAGED BY SYSTEM
  USING ('d:\acc_tbsp', 'e:\acc_tbsp', 'f:\acc_tbsp')
  EXTENTSIZE 64
  PREFETCHSIZE 32
```

Example 3: Create a system temporary DMS table space on a UNIX system using two files of 50 000 pages each, and a 256-page extent size.

```
CREATE TEMPORARY TABLESPACE TEMPSPACE2
  MANAGED BY DATABASE
  USING (FILE 'dbtmp/tempspace2.f1' 50000,
        FILE 'dbtmp/tempspace2.f2' 50000)
  EXTENTSIZE 256
```

Example 4: Create a large DMS table space in database partition group ODDNODEGROUP (database partitions 1, 3, and 5) on a UNIX system. Use the device /dev/rhdisk0 for 10 000 4K pages on each database partition. Specify a database partition-specific device with 40 000 4K pages for each database partition.

```
CREATE TABLESPACE PLANS
  MANAGED BY DATABASE
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn1hd01' 40000)
  ON DBPARTITIONNUM (1)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn3hd03' 40000)
  ON DBPARTITIONNUM (3)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn5hd05' 40000)
  ON DBPARTITIONNUM (5)
```

Example 5: Create a large automatic storage table space named DATATS, allowing the system to make all decisions with respect to table space size and growth.

```
CREATE TABLESPACE DATATS
```

or

```
CREATE TABLESPACE DATATS
  MANAGED BY AUTOMATIC STORAGE
```

Example 6: Create a system temporary automatic storage table space named TEMPDATA.

```
CREATE TEMPORARY TABLESPACE TEMPDATA
```

or

```
CREATE TEMPORARY TABLESPACE TEMPDATA
  MANAGED BY AUTOMATIC STORAGE
```

Example 7: Create a large automatic storage table space named USERSPACE3 with an initial size of 100 megabytes and a maximum size of 1 gigabyte.

```
CREATE TABLESPACE USERSPACE3
  INITIALSIZE 100 M
  MAXSIZE 1 G
```

Example 8: Create a large automatic storage table space named LARGEDATA with a growth rate of 10 percent (that is, its total size increases by 10 percent each time that it is automatically resized) and a maximum size of 512 megabytes. Instead of specifying the INITIALSIZE clause, let the database manager determine an appropriate initial size for the table space.


```

CREATE LARGE TABLESPACE LARGEDATA
  INCREASESIZE 10 PERCENT
  MAXSIZE 512 M

```

Example 9: Create a large DMS table space named USERSPACE4 with two file containers (each container being 1 megabyte in size), a growth rate of 2 megabytes, and a maximum size of 100 megabytes.

```

CREATE TABLESPACE USERSPACE4
  MANAGED BY DATABASE USING (FILE '/db2/file1' 1 M, FILE '/db2/file2' 1 M)
  AUTORESIZE YES
  INCREASESIZE 2 M
  MAXSIZE 100 M

```

Example 10: Create large DMS table spaces, using RAW devices on a Windows operating system.

- To specify entire physical drives, use the `\\.\physical-drive` format:

```

CREATE TABLESPACE TS1
  MANAGED BY DATABASE USING (DEVICE '\\.\PhysicalDrive5' 10000,
  DEVICE '\\.\PhysicalDrive6' 10000)

```

- To specify logical partitions by using drive letters:

```

CREATE TABLESPACE TS2
  MANAGED BY DATABASE USING (DEVICE '\\.\G:' 10000,
  DEVICE '\\.\H:' 10000)

```

- To specify logical partitions by using volume global unique identifiers (GUIDs), use the `db2listvolumes` utility to retrieve the volume GUID for each local partition, then copy the GUID for the logical partition that you want into the table space container clause:

```

CREATE TABLESPACE TS3
  MANAGED BY DATABASE USING (
  DEVICE '\\?\Volume{2ca6a0c1-8542-11d8-9734-00096b5322d2}\ 20000M)

```

You might prefer to use volume GUIDs over the drive letter format if you have more partitions than available drive letters on the machine.

- To specify logical partitions by using junction points (or volume mount points), mount the RAW partition to another NTFS-formatted volume as a junction point, then specify the path to the junction point on the NTFS volume as the container path. For example:

```

CREATE TABLESPACE TS4
  MANAGED BY DATABASE USING (DEVICE 'C:\JUNCTION\DISK_1' 10000,
  DEVICE 'C:\JUNCTION\DISK_2' 10000)

```

DB2 first queries the partition to see whether there is a file system on it; if yes, the partition is not treated as a RAW device, and DB2 performs normal file system I/O operations on the partition.

CREATE VIEW

The `CREATE VIEW` statement defines a view on one or more tables, views or nicknames.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if `DYNAMICRULES` run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the view does not exist
- CREATEIN privilege on the schema, if the schema name of the view refers to an existing schema
- DBADM authority

and at least one of the following for each table, view, or nickname identified in any fullselect:

- CONTROL privilege on that table, view, or nickname
- SELECT privilege on that table, view, or nickname
- DATAACCESS authority

If creating a subview:

- The authorization ID of the statement must be the same as the definer of the root table of the table hierarchy, or
- The privileges held by the authorization ID must include DBADM authority

and

- The authorization ID of the statement must have SELECT WITH GRANT privilege on the underlying table of the subview, or the superview must not have SELECT privilege granted to any user other than the view definer, or
- ACCESSCTRL authority and one of the following:
 - SELECT privilege on the underlying table of the subview
 - DATAACCESS authority

If WITH ROW MOVEMENT is specified, the privileges held by the authorization ID of the statement must include at least one of the following:

- UPDATE privilege on that table or view
- DATAACCESS authority

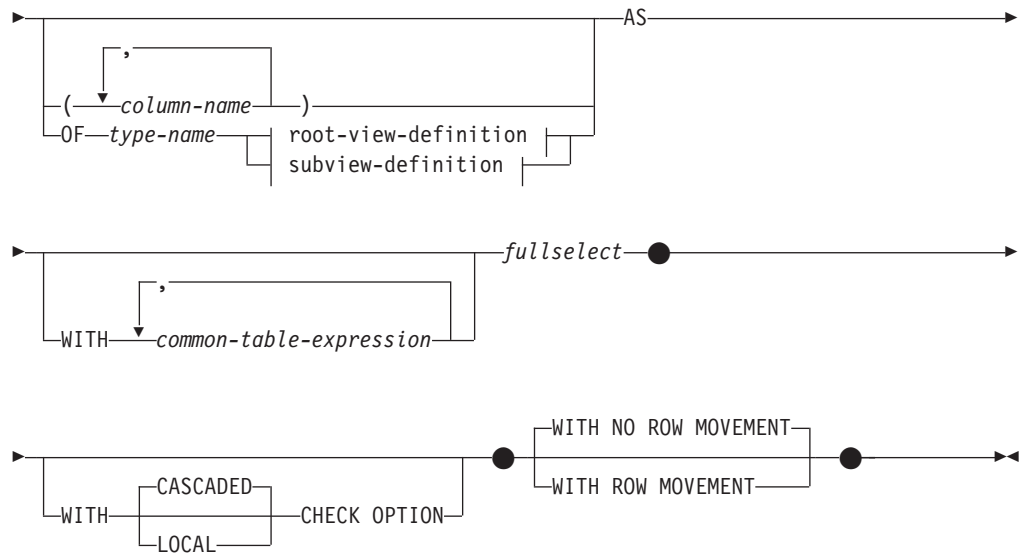
Group privileges are not considered for any table or view specified in the CREATE VIEW statement.

Privileges are not considered when defining a view on a federated database nickname. Authorization requirements of the data source for the table or view referenced by the nickname are applied when the query is processed. The authorization ID of the statement can be mapped to a different remote authorization ID.

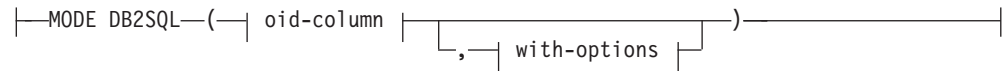
To replace an existing view, the authorization ID of the statement must be the owner of the existing view (SQLSTATE 42501).

Syntax

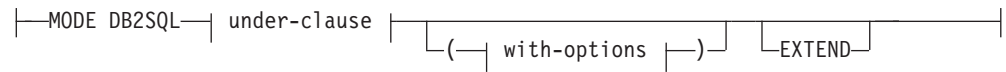
```
►► CREATE [OR REPLACE] VIEW view-name →
```



root-view-definition:



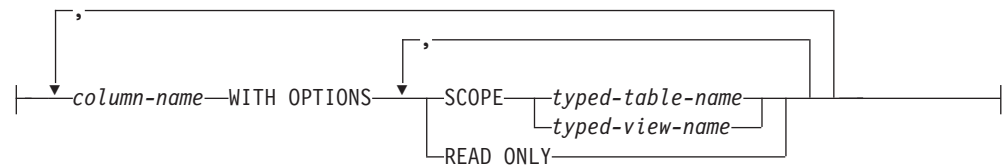
subview-definition:



oid-column:



with-options:



under-clause:



Description

OR REPLACE

Specifies to replace the definition for the view if one exists at the current server. The existing definition is effectively dropped before the new definition

is replaced in the catalog, with the exception that privileges that were granted on the view are not affected. This option is ignored if a definition for the view does not exist at the current server.

view-name

Names the view. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname or alias described in the catalog. The qualifier must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

The name can be the same as the name of an inoperative view (see Inoperative views). In this case the new view specified in the CREATE VIEW statement will replace the inoperative view. The user will get a warning (SQLSTATE 01595) when an inoperative view is replaced. No warning is returned if the application was bound with the bind option SQLWARN set to NO.

column-name

Names the columns in the view. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the view inherit the names of the columns of the result table of the fullselect.

A list of column names must be specified if the result table of the fullselect has duplicate column names or an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

OF *type-name*

Specifies that the columns of the view are based on the attributes of the structured type identified by *type-name*. If *type-name* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The type name must be the name of an existing user-defined type (SQLSTATE 42704) and it must be a structured type that is instantiable (SQLSTATE 428DP).

MODE DB2SQL

This clause is used to specify the mode of the typed view. This is the only valid mode currently supported.

UNDER *superview-name*

Indicates that the view is a subview of *superview-name*. The superview must be an existing view (SQLSTATE 42704) and the view must be defined using a structured type that is the immediate supertype of *type-name* (SQLSTATE 428DB). The schema name of *view-name* and *superview-name* must be the same (SQLSTATE 428DQ). The view identified by *superview-name* must not have any existing subview already defined using *type-name* (SQLSTATE 42742).

The columns of the view include the object identifier column of the superview with its type modified to be REF(*type-name*), followed by columns based on the attributes of *type-name* (remember that the type includes the attributes of its supertype).

INHERIT SELECT PRIVILEGES

Any user or group holding a SELECT privilege on the superview will be granted an equivalent privilege on the newly created subview. The subview definer is considered to be the grantor of this privilege.

OID-column

Defines the object identifier column for the typed view.

REF IS *OID-column-name* USER GENERATED

Specifies that an object identifier (OID) column is defined in the view as the first column. An OID is required for the root view of a view hierarchy (SQLSTATE 428DX). The view must be a typed view (the OF clause must be present) that is not a subview (SQLSTATE 42613). The name for the column is defined as *OID-column-name* and cannot be the same as the name of any attribute of the structured type *type-name* (SQLSTATE 42711). The first column specified in *fullselect* must be of type REF(*type-name*) (you may need to cast it so that it has the appropriate type). If UNCHECKED is not specified, it must be based on a not nullable column on which uniqueness is enforced through an index (primary key, unique constraint, unique index, or OID-column). This column will be referred to as the *object identifier column* or *OID column*. The keywords USER GENERATED indicate that the initial value for the OID column must be provided by the user when inserting a row. Once a row is inserted, the OID column cannot be updated (SQLSTATE 42808).

UNCHECKED

Defines the object identifier column of the typed view definition to assume uniqueness even though the system can not prove this uniqueness. This is intended for use with tables or views that are being defined into a typed view hierarchy where the user knows that the data conforms to this uniqueness rule but it does not comply with the rules that allow the system to prove uniqueness. UNCHECKED option is mandatory for view hierarchies that range over multiple hierarchies or legacy tables or views. By specifying UNCHECKED, the user takes responsibility for ensuring that each row of the view has a unique OID. If the user fails to ensure this property, and a view contains duplicate OID values, then a path-expression or Deref operator involving one of the non-unique OID values may result in an error (SQLSTATE 21000).

with-options

Defines additional options that apply to columns of a typed view.

column-name **WITH OPTIONS**

Specifies the name of the column for which additional options are specified. The *column-name* must correspond to the name of an attribute defined in (not inherited by) the *type-name* of the view. The column must be a reference type (SQLSTATE 42842). It cannot correspond to a column that also exists in the superview (SQLSTATE 428DJ). A column name can only appear in one WITH OPTIONS SCOPE clause in the statement (SQLSTATE 42613).

SCOPE

Identifies the scope of the reference type column. A scope must be specified for any column that is intended to be used as the left operand of a dereference operator or as the argument of the Deref function.

Specifying the scope for a reference type column may be deferred to a subsequent ALTER VIEW statement (if the scope is not inherited) to allow the target table or view to be defined, usually in the case of mutually referencing views and tables. If no scope is specified for a reference type column of the view and the underlying table or view column was scoped, then the underlying column's scope is inherited by the reference type column. The column remains unscoped if the underlying table or view column did not have a scope. See "Notes" on page 233 for more information about scope and reference type columns.

typed-table-name

The name of a typed table. The table must already exist or be the same as the name of the table being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name

The name of a typed view. The view must already exist or be the same as the name of the view being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

READ ONLY

Identifies the column as a read-only column. This option is used to force a column to be read-only so that subview definitions can specify an expression for the same column that is implicitly read-only.

AS

Identifies the view definition.

WITH *common-table-expression*

Defines a common table expression for use with the fullselect that follows. A common table expression cannot be specified when defining a typed view.

fullselect

Defines the view. At any time, the view consists of the rows that would result if the SELECT statement were executed. The fullselect must not reference host variables, parameter markers, or declared temporary tables. However, a parameterized view can be created as an SQL table function.

The fullselect cannot include an SQL data change statement in the FROM clause (SQLSTATE 428FL).

For Typed Views and Subviews: The *fullselect* must conform to the following rules otherwise an error is returned (SQLSTATE 428EA unless otherwise specified).

- The fullselect must not include references to the DBPARTITIONNUM or HASHEDVALUE functions, non-deterministic functions, or functions defined to have external action.
- The body of the view must consist of a single subselect, or a UNION ALL of two or more subselects. Let each of the subselects participating directly in the view body be called a *branch* of the view. A view may have one or more branches.
- The FROM-clause of each branch must consist of a single table or view (not necessarily typed), called the *underlying* table or view of that branch.
- The underlying table or view of each branch must be in a separate hierarchy (that is, a view cannot have multiple branches with their underlying tables or views in the same hierarchy).
- None of the branches of a typed view definition may specify GROUP BY or HAVING.
- If the view body contains UNION ALL, the root view in the hierarchy must specify the UNCHECKED option for its OID column.

For a hierarchy of views and subviews: Let BR1 and BR2 be any branches that appear in the definitions of views in the hierarchy. Let T1 be the underlying table or view of BR1, and let T2 be the underlying table or view of BR2. Then:

- If T1 and T2 are not in the same hierarchy, then the root view in the view hierarchy must specify the UNCHECKED option for its OID column.
- If T1 and T2 are in the same hierarchy, then BR1 and BR2 must contain predicates or ONLY-clauses that are sufficient to guarantee that their row-sets are disjoint.

For typed subviews defined using EXTEND AS: For every branch in the body of the subview:

- The underlying table of each branch must be a (not necessarily proper) subtable of some underlying table of the immediate superview.
- The expressions in the SELECT list must be assignable to the non-inherited columns of the subview (SQLSTATE 42854).

For typed subviews defined using AS without EXTEND:

- For every branch in the body of the subview, the expressions in the SELECT-list must be assignable to the declared types of the inherited and non-inherited columns of the subview (SQLSTATE 42854).
- The OID-expression of each branch over a given hierarchy in the subview must be equivalent (except for casting) to the OID-expression in the branch over the same hierarchy in the root view.
- The expression for a column not defined (implicitly or explicitly) as READ ONLY in a superview must be equivalent in all branches over the same underlying hierarchy in its subviews.

WITH CHECK OPTION

Specifies the constraint that every row that is inserted or updated through the view must conform to the definition of the view. A row that does not conform to the definition of the view is a row that does not satisfy the search conditions of the view.

WITH CHECK OPTION must not be specified if any of the following conditions is true:

- The view is read-only (SQLSTATE 42813). If WITH CHECK OPTION is specified for an updatable view that does not allow inserts, the constraint applies to updates only.
- The view references the DBPARTITIONNUM or HASHEDVALUE function, a non-deterministic function, or a function with external action (SQLSTATE 42997).
- A nickname is the update target of the view.
- A view that has an INSTEAD OF trigger defined on it is the update target of the view (SQLSTATE 428FQ).

If WITH CHECK OPTION is omitted, the definition of the view is not used in the checking of any insert or update operations that use the view. Some checking might still occur during insert or update operations if the view is directly or indirectly dependent on another view that includes WITH CHECK OPTION. Because the definition of the view is not used, rows might be inserted or updated through the view that do not conform to the definition of the view.

CASCADED

The WITH CASCADED CHECK OPTION constraint on a view *V* means that *V* inherits the search conditions as constraints from any updatable

view on which *V* is dependent. Furthermore, every updatable view that is dependent on *V* is also subject to these constraints. Thus, the search conditions of *V* and each view on which *V* is dependent are ANDed together to form a constraint that is applied for an insert or update of *V* or of any view dependent on *V*.

LOCAL

The WITH LOCAL CHECK OPTION constraint on a view *V* means the search condition of *V* is applied as a constraint for an insert or update of *V* or of any view that is dependent on *V*.

The difference between CASCADED and LOCAL is shown in the following example. Consider the following updatable views (substituting for *Y* from column headings of the table that follows):

```
V1 defined on table T
V2 defined on V1 WITH Y CHECK OPTION
V3 defined on V2
V4 defined on V3 WITH Y CHECK OPTION
V5 defined on V4
```

The following table shows the search conditions against which inserted or updated rows are checked:

	Y is LOCAL	Y is CASCADED
V1 checked against:	no view	no view
V2 checked against:	V2	V2, V1
V3 checked against:	V2	V2, V1
V4 checked against:	V2, V4	V4, V3, V2, V1
V5 checked against:	V2, V4	V4, V3, V2, V1

Consider the following updatable view which shows the impact of the WITH CHECK OPTION using the default CASCADED option:

```
CREATE VIEW V1 AS SELECT COL1 FROM T1 WHERE COL1 > 10

CREATE VIEW V2 AS SELECT COL1 FROM V1 WITH CHECK OPTION

CREATE VIEW V3 AS SELECT COL1 FROM V2 WHERE COL1 < 100
```

The following INSERT statement using *V1* will succeed because *V1* does not have a WITH CHECK OPTION and *V1* is not dependent on any other view that has a WITH CHECK OPTION.

```
INSERT INTO V1 VALUES(5)
```

The following INSERT statement using *V2* will result in an error because *V2* has a WITH CHECK OPTION and the insert would produce a row that did not conform to the definition of *V2*.

```
INSERT INTO V2 VALUES(5)
```

The following INSERT statement using *V3* will result in an error even though it does not have WITH CHECK OPTION because *V3* is dependent on *V2* which does have a WITH CHECK OPTION (SQLSTATE 44000).

```
INSERT INTO V3 VALUES(5)
```

The following INSERT statement using *V3* will succeed even though it does not conform to the definition of *V3* (*V3* does not have a WITH CHECK OPTION); it does conform to the definition of *V2* which does have a WITH CHECK OPTION.

```
INSERT INTO V3 VALUES(200)
```

WITH NO ROW MOVEMENT or WITH ROW MOVEMENT

Specifies the action to take for an updatable UNION ALL view when a row is updated in a way that violates a check constraint on the underlying table. The default is WITH NO ROW MOVEMENT.

WITH NO ROW MOVEMENT

Specifies that an error (SQLSTATE 23513) is to be returned if a row is updated in a way that violates a check constraint on the underlying table.

WITH ROW MOVEMENT

Specifies that an updated row is to be moved to the appropriate underlying table, even if it violates a check constraint on that table.

Row movement involves deletion of the rows that violate the check constraint, and insertion of those rows back into the view. The WITH ROW MOVEMENT clause can only be specified for UNION ALL views whose columns are all updatable (SQLSTATE 429BJ). If a row is inserted (perhaps after trigger activation) into the same underlying table from which it was deleted, an error is returned (SQLSTATE 23524). A view defined using the WITH ROW MOVEMENT clause must not contain nested UNION ALL operations, except in the outermost fullselect (SQLSTATE 429BJ).

Notes

- Creating a view with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- View columns inherit the NOT NULL WITH DEFAULT attribute from the base table or view except when columns are derived from an expression. When a row is inserted or updated into an updatable view, it is checked against the constraints (primary key, referential integrity, and check) if any are defined on the base table.
- A new view cannot be created if it uses an inoperative view in its definition. (SQLSTATE 51024).
- If an object referenced in the view body does not exist or is marked invalid, or the definer temporarily doesn't have privileges to access the object, and if the database configuration parameter **auto_reval** is not set to DISABLED, then the view will still be created successfully. The view will be marked invalid and will be revalidated the next time it is referenced.
- This statement does not support declared temporary tables (SQLSTATE 42995).
- **Deletable views:** A view is *deletable* if an INSTEAD OF trigger for the delete operation has been defined for the view, or if all of the following are true:
 - each FROM clause of the outer fullselect identifies only one base table (with no OUTER clause), deletable view (with no OUTER clause), deletable nested table expression, or deletable common table expression (cannot identify a nickname)
 - the outer fullselect does not include a VALUES clause
 - the outer fullselect does not include a GROUP BY clause or HAVING clause
 - the outer fullselect does not include column functions in the select list
 - the outer fullselect does not include SET operations (UNION, EXCEPT or INTERSECT) with the exception of UNION ALL
 - the base tables in the operands of a UNION ALL must not be the same table and each operand must be deletable
 - the select list of the outer fullselect does not include DISTINCT

- **Updatable views:** A column of a view is *updatable* if an INSTEAD OF trigger for the update operation has been defined for the view, or if all of the following are true:
 - the view is deletable (independent of an INSTEAD OF trigger for delete), the column resolves to a column of a base table (not using a dereference operation), and the READ ONLY option is not specified
 - all the corresponding columns of the operands of a UNION ALL have exactly matching data types (including length or precision and scale) and matching default values if the fullselect of the view includes a UNION ALL

A view is updatable if *any* column of the view is updatable.

- **Insertable views:** A view is insertable if an INSTEAD OF trigger for the insert operation has been defined for the view, or at least one column of the view is updatable (independent of an INSTEAD OF trigger for update), and the fullselect of the view does not include UNION ALL.

A given row can be inserted into a view (including a UNION ALL) if, and only if, it fulfills the check constraints of exactly one of the underlying base tables.

To insert into a view that includes non-updatable columns, those columns must be omitted from the column list.

- **Read-only views:** A view is *read-only* if it is *not* deletable, updatable, or insertable.

The READONLY column in the SYSCAT.VIEWS catalog view indicates if a view is read-only without considering INSTEAD OF triggers.

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

- **Inoperative views:** An *inoperative view* is a view that is no longer available for SQL statements. A view becomes inoperative if:

- A privilege, upon which the view definition is dependent, is revoked.
- An object such as a table, nickname, alias or function, upon which the view definition is dependent, is dropped.
- A view, upon which the view definition is dependent, becomes inoperative.
- A view that is the superview of the view definition (the subview) becomes inoperative.

In practical terms, an inoperative view is one in which the view definition has been unintentionally dropped. For example, when an alias is dropped, any view defined using that alias is made inoperative. All dependent views also become inoperative and packages dependent on the view are no longer valid.

Until the inoperative view is explicitly recreated or dropped, a statement using that inoperative view cannot be compiled (SQLSTATE 51024) with the exception of the CREATE ALIAS, CREATE VIEW, DROP VIEW, and COMMENT ON TABLE statements. Until the inoperative view has been explicitly dropped, its qualified name cannot be used to create another table or alias (SQLSTATE 42710).

An inoperative view may be recreated by issuing a CREATE VIEW statement using the definition text of the inoperative view. This view definition text is stored in the TEXT column of the SYSCAT.VIEWS catalog. When recreating an inoperative view, it is necessary to explicitly grant any privileges required on that view by others, due to the fact that all authorization records on a view are deleted if the view is marked inoperative. Note that there is no need to explicitly drop the inoperative view in order to recreate it. Issuing a CREATE VIEW

statement with the same *view-name* as an inoperative view will cause that inoperative view to be replaced, and the CREATE VIEW statement will return a warning (SQLSTATE 01595).

Inoperative views are indicated by an X in the VALID column of the SYSCAT.VIEWS catalog view and an X in the STATUS column of the SYSCAT.TABLES catalog view.

- **Privileges:** The definer of a view always receives the SELECT privilege on the view as well as the right to drop the view. The definer of a view will get CONTROL privilege on the view only if the definer has CONTROL privilege on every base table, view, or nickname identified in the fullselect, or if the definer has each of the following authorities:

- ACCESSCTRL or SECADM
- DATAACCESS
- DBADM

The definer of the view is granted INSERT, UPDATE, column level UPDATE or DELETE privileges on the view if the view is not read-only and the definer has the corresponding privileges on the underlying objects.

For a view defined WITH ROW MOVEMENT, the definer acquires the UPDATE privilege on the view only if the definer has the UPDATE privilege on all columns of the view, as well as INSERT and DELETE privileges on all underlying tables or views.

The definer of a view only acquires privileges if the privileges from which they are derived exist at the time the view is created. The definer must have these privileges either directly or because PUBLIC has these privilege. Privileges are not considered when defining a view on a federated server nickname. However, when using a view on a nickname, the user's authorization ID must have valid select privileges on the table or view that the nickname references at the data source. Otherwise, an error is returned. Privileges held by groups of which the view definer is a member, are not considered.

When a subview is created, the SELECT privileges held on the immediate superview are automatically granted on the subview.

- **Scope and REF columns:** When selecting a reference type column in the fullselect of a view definition, consider the target type and scope that is required.
 - If the required target type and scope is the same as the underlying table or view, the column can simply be selected.
 - If the scope needs to be changed, use the WITH OPTIONS SCOPE clause to define the required scope table or view.
 - If the target type of the reference needs to be changed, the column must be cast first to the representation type of the reference and then to the new reference type. The scope in this case can be specified in the cast to the reference type or using the WITH OPTIONS SCOPE clause. For example, assume you select column Y defined as REF(TYP1) SCOPE TAB1. You want this to be defined as REF(VTYP1) SCOPE VIEW1. The select list item would be as follows:

```
CAST(CAST(Y AS VARCHAR(16) FOR BIT DATA) AS REF(VTYP1) SCOPE VIEW1)
```

- **Identity columns:** A column of a view is considered an identity column, if the element of the corresponding column in the fullselect of the view definition is the name of an identity column of a table, or the name of a column of a view which directly or indirectly maps to the name of an identity column of a base table.

In all other cases, the columns of a view will not get the identity property. For example:

- the select-list of the view definition includes multiple instances of the name of an identity column (that is, selecting the same column more than once)
- the view definition involves a join
- a column in the view definition includes an expression that refers to an identity column
- the view definition includes a UNION

When inserting into a view for which the select list of the view definition directly or indirectly includes the name of an identity column of a base table, the same rules apply as if the INSERT statement directly referenced the identity column of the base table.

- **Federated views:** A federated view is a view that includes a reference to a nickname somewhere in the fullselect. The presence of such a nickname changes the authorization model used for the view when the view is subsequently referenced in a query.

When the view is created, no privilege checking is done to determine whether the view definer has access to the underlying data source table or view of a nickname. Privilege checking of references to tables or views at the federated database are handled as usual, requiring the view definer to have at least SELECT privilege on such objects.

When a federated view is subsequently referenced in a query, the nicknames result in queries against the data source, and the authorization ID that issued the query (or the remote authorization ID to which it maps) must have the necessary privileges to access the data source table or view. The authorization ID that issues the query referencing the federated view is not required to have any additional privileges on tables or views (non-federated) that exist at the federated server.

- **ROW MOVEMENT, triggers and constraints:** When a view that is defined using the WITH ROW MOVEMENT clause is updated, the sequence of trigger and constraints operations is as follows:
 1. BEFORE UPDATE triggers are activated for all rows being updated, including rows that will eventually be moved.
 2. The update operation is processed.
 3. Constraints are processed for all updated rows.
 4. AFTER UPDATE triggers (both row-level and statement-level) are activated in creation order, for all rows that satisfy the constraints after the update operation. Because this is an UPDATE statement, all UPDATE statement-level triggers are activated for all underlying tables.
 5. BEFORE DELETE triggers are activated for all rows that did not satisfy the constraints after the update operation (these are the rows that are to be moved).
 6. The delete operation is processed.
 7. Constraints are processed for all deleted rows.
 8. AFTER DELETE triggers (both row-level and statement-level) are activated in creation order, for all deleted rows. Statement-level triggers are activated for only those tables that are involved in the delete operation.
 9. BEFORE INSERT triggers are activated for all rows being inserted (that is, the rows being moved). The new transition tables for the BEFORE INSERT triggers contain the input data provided by the user.
 10. The insert operation is processed.
 11. Constraints are processed for all inserted rows.

- 12. AFTER INSERT triggers (both row-level and statement-level) are activated in creation order, for all inserted rows. Statement-level triggers are activated for only those tables that are involved in the insert operation.
- **Nested UNION ALL views:** A view defined with UNION ALL and based, either directly or indirectly, on a view that is also defined with UNION ALL cannot be updated if either view is defined using the WITH ROW MOVEMENT clause (SQLSTATE 429BK).
- **Considerations for implicitly hidden columns:** It is possible that the result table of the fullselect will include a column of the base table that is defined as implicitly hidden. This can occur when the implicitly hidden column is explicitly referenced in the fullselect of the view definition. However, the corresponding column of the view does not inherit the implicitly hidden attribute. Columns of a view cannot be defined as hidden.
- **Compatibilities:** For compatibility with previous versions of DB2 databases:
 - The FEDERATED keyword can be specified between the keywords CREATE and VIEW. The FEDERATED keyword is ignored, however, because a warning is no longer returned if federated objects are used in the view definition.
- **Subselect:** The *isolation-clause* cannot be specified in the *fullselect* (SQLSTATE 42601).

Examples

Example 1: Create a view named MA_PROJ upon the PROJECT table that contains only those rows with a project number (PROJNO) starting with the letters 'MA'.

```
CREATE VIEW MA_PROJ AS SELECT *
FROM PROJECT
WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Example 2: Create a view as in example 1, but select only the columns for project number (PROJNO), project name (PROJNAME) and employee in charge of the project (RESPEMP).

```
CREATE VIEW MA_PROJ
AS SELECT PROJNO, PROJNAME, RESPEMP
FROM PROJECT
WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Example 3: Create a view as in example 2, but, in the view, call the column for the employee in charge of the project IN_CHARGE.

```
CREATE VIEW MA_PROJ
(PROJNO, PROJNAME, IN_CHARGE)
AS SELECT PROJNO, PROJNAME, RESPEMP
FROM PROJECT
WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Note: Even though only one of the column names is being changed, the names of all three columns in the view must be listed in the parentheses that follow MA_PROJ.

Example 4: Create a view named PRJ_LEADER that contains the first four columns (PROJNO, PROJNAME, DEPTNO, RESPEMP) from the PROJECT table together with the last name (LASTNAME) of the person who is responsible for the project (RESPEMP). Obtain the name from the EMPLOYEE table by matching EMPNO in EMPLOYEE to RESPEMP in PROJECT.


```

CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO

```

Example 5: Create a view as in example 4, but in addition to the columns PROJNO, PROJNAME, DEPTNO, RESPEMP, and LASTNAME, show the total pay (SALARY + BONUS + COMM) of the employee who is responsible. Also select only those projects with mean staffing (PRSTAFF) greater than one.

```

CREATE VIEW PRJ_LEADER
  (PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, TOTAL_PAY )
AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, SALARY+BONUS+COMM
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO
  AND PRSTAFF > 1

```

Specifying the column name list could be avoided by naming the expression SALARY+BONUS+COMM as TOTAL_PAY in the fullselect.

```

CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP,
             LASTNAME, SALARY+BONUS+COMM AS TOTAL_PAY
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO AND PRSTAFF > 1

```

Example 6: Given the set of tables and views shown in the following figure: User ZORPIE (who does not have ACCESSCTRL, DATAACCESS, or DBADM

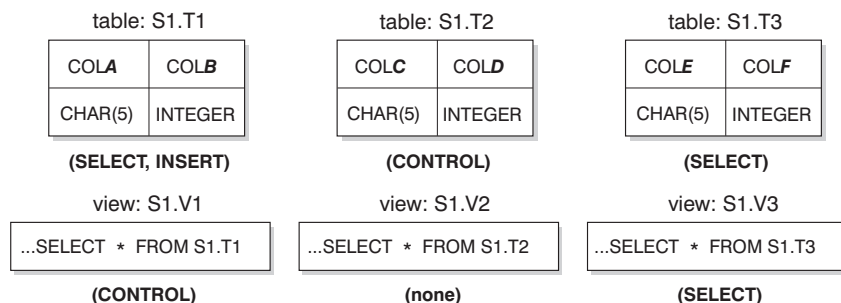


Figure 1. Tables and Views for Example 6

authority) has been granted the privileges shown in brackets below each object:

- ZORPIE will get CONTROL privilege on the view that she creates with:

```

CREATE VIEW VA AS SELECT * FROM S1.V1

```

because she has CONTROL on S1.V1. (CONTROL on S1.V1 must have been granted to ZORPIE by someone with ACCESSCTRL or SECADM authority.) It does not matter which, if any, privileges she has on the underlying base table.

- ZORPIE will not be allowed to create the view:

```

CREATE VIEW VB AS SELECT * FROM S1.V2

```

because she has neither CONTROL nor SELECT on S1.V2. It does not matter that she has CONTROL on the underlying base table (S1.T2).

- ZORPIE will get CONTROL privilege on the view that she creates with:

```

CREATE VIEW VC (COLA, COLB, COLC, COLD)
  AS SELECT * FROM S1.V1, S1.T2
  WHERE COLA = COLC

```


because the fullselect of ZORPIE.VC references view S1.V1 and table S1.T2 and she has CONTROL on both of these. Note that the view VC is read-only, so ZORPIE does not get INSERT, UPDATE or DELETE privileges.

4. ZORPIE will get SELECT privilege on the view that she creates with:

```
CREATE VIEW VD (COLA,COLB, COLE, COLF)
AS SELECT * FROM S1.V1, S1.V3
WHERE COLA = COLE
```

because the fullselect of ZORPIE.VD references the two views S1.V1 and S1.V3, one on which she has only SELECT privilege, and one on which she has CONTROL privilege. She is given the lesser of the two privileges, SELECT, on ZORPIE.VD.

5. ZORPIE will get INSERT, UPDATE and DELETE privilege WITH GRANT OPTION and SELECT privilege on the view VE in the following view definition.

```
CREATE VIEW VE
AS SELECT * FROM S1.V1
WHERE COLA > ANY
(SELECT COLE FROM S1.V3)
```

ZORPIE's privileges on VE are determined primarily by her privileges on S1.V1. Since S1.V3 is only referenced in a subquery, she only needs SELECT privilege on S1.V3 to create the view VE. The definer of a view only gets CONTROL on the view if they have CONTROL on all objects referenced in the view definition. ZORPIE does not have CONTROL on S1.V3, consequently she does not get CONTROL on VE.

DELETE

The DELETE statement deletes rows from a table, nickname, or view, or the underlying tables, nicknames, or views of the specified fullselect. Deleting a row from a nickname deletes the row from the data source object to which the nickname refers. Deleting a row from a view deletes the row from the table on which the view is based if no INSTEAD OF trigger is defined for the delete operation on this view. If such a trigger is defined, the trigger will be executed instead.

There are two forms of this statement:

- The *Searched* DELETE form is used to delete one or more rows (optionally determined by a search condition).
- The *Positioned* DELETE form is used to delete exactly one row (as determined by the current position of a cursor).

Invocation

A DELETE statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

To execute either form of this statement, the privileges held by the authorization ID of the statement must include at least one of the following:

- DELETE privilege on the table, view, or nickname from which rows are to be deleted

- CONTROL privilege on the table, view, or nickname from which rows are to be deleted
- DATAACCESS authority

To execute a Searched DELETE statement, the privileges held by the authorization ID of the statement must also include at least one of the following for each table, view, or nickname referenced by a subquery:

- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

If the package used to process the statement is precompiled with SQL92 rules (option LANGLEVEL with a value of SQL92E or MIA), and the searched form of a DELETE statement includes a reference to a column of the table or view in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

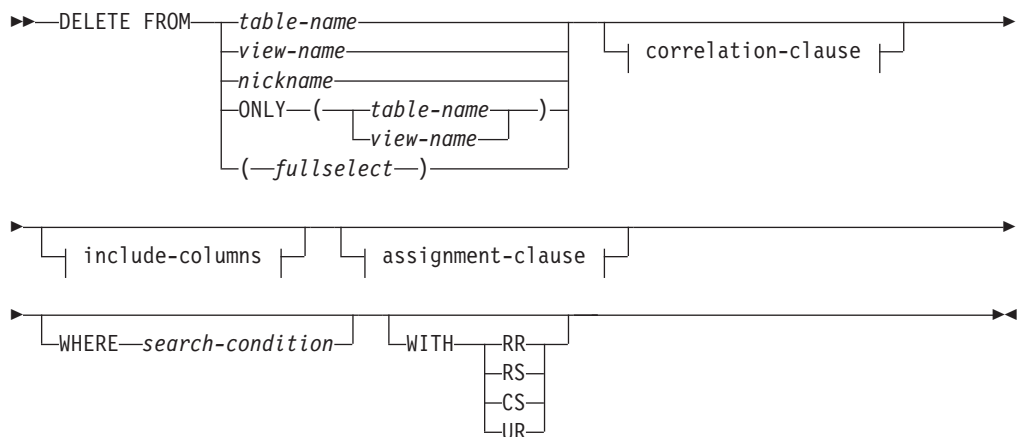
If the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

Group privileges are not checked for static DELETE statements.

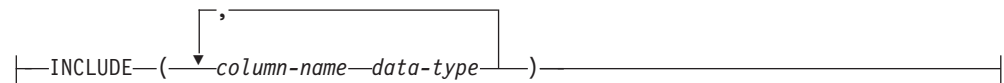
If the target of the delete operation is a nickname, the privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges required for the operation on the object at the data source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

Syntax

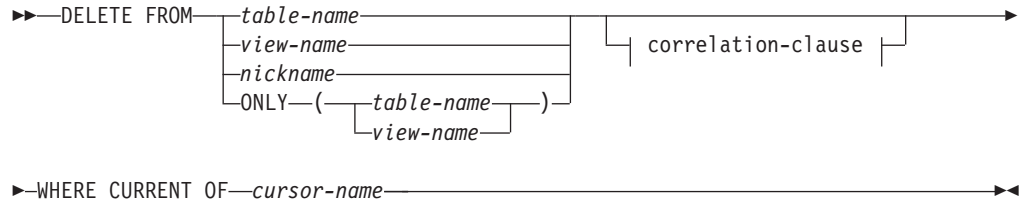
searched-delete:



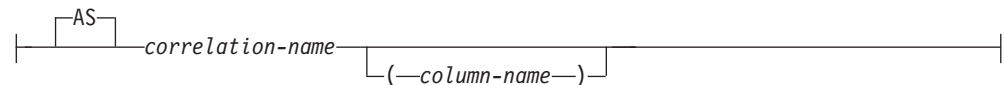
include-columns:



positioned-delete:



correlation-clause:



Description

FROM *table-name*, *view-name*, *nickname*, or (*fullselect*)

Identifies the object of the delete operation. The name must identify a table or view that exists in the catalog, but it must not identify a catalog table, a catalog view, a system-maintained materialized query table, or a read-only view.

If *table-name* is a typed table, rows of the table or any of its proper subtables may get deleted by the statement.

If *view-name* is a typed view, rows of the underlying table or underlying tables of the view's proper subviews may get deleted by the statement. If *view-name* is a regular view with an underlying table that is a typed table, rows of the typed table or any of its proper subtables may get deleted by the statement.

If the object of the delete operation is a fullselect, the fullselect must be deletable, as defined in the "Deletable views" Notes item in the description of the CREATE VIEW statement.

Only the columns of the specified table can be referenced in the WHERE clause. For a positioned DELETE, the associated cursor must also have specified the table or view in the FROM clause without using ONLY.

FROM ONLY (*table-name*)

Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be deleted by the statement. For a positioned DELETE, the associated cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

FROM ONLY (*view-name*)

Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews

cannot be deleted by the statement. For a positioned DELETE, the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

correlation-clause

Can be used within the *search-condition* to designate a table, view, nickname, or fullselect. For a description of *correlation-clause*, see “table-reference” in the description of “Subselect”.

include-columns

Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the DELETE statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE

Specifies a list of columns to be included in the intermediate result table of the DELETE statement.

column-name

Specifies a column of the intermediate result table of the DELETE statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type

Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

assignment-clause

See the description of *assignment-clause* under the UPDATE statement. The same rules apply. The *include-columns* are the only columns that can be set using the *assignment-clause* (SQLSTATE 42703).

WHERE

Specifies a condition that selects the rows to be deleted. The clause can be omitted, a search condition specified, or a cursor named. If the clause is omitted, all rows of the table or view are deleted.

search-condition

Each *column-name* in the search condition, other than in a subquery must identify a column of the table or view.

The *search-condition* is applied to each row of the table, view, or nickname, and the deleted rows are those for which the result of the *search-condition* is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the *search condition* is applied to a row, and the results used in applying the *search condition*. In actuality, a subquery with no correlated references is executed once, whereas a subquery with a correlated reference may have to be executed once for each row. If a subquery refers to the object table of a DELETE statement or a dependent table with a delete rule of CASCADE or SET NULL, the subquery is completely evaluated before any rows are deleted.

CURRENT OF *cursor-name*

Identifies a cursor that is defined in a DECLARE CURSOR statement of the program. The DECLARE CURSOR statement must precede the DELETE statement.

The table, view, or nickname named must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see “DECLARE CURSOR”.)

When the DELETE statement is executed, the cursor must be positioned on a row: that row is the one deleted. After the deletion, the cursor is positioned before the next row of its result table. If there is no next row, the cursor is positioned after the last row.

WITH

Specifies the isolation level used when locating the rows to be deleted.

RR

Repeatable Read

RS

Read Stability

CS

Cursor Stability

UR

Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. The WITH clause has no effect on nicknames, which always use the default isolation level of the statement.

Rules

- **Triggers:** DELETE statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the deleted rows. If a DELETE statement on a view causes an INSTEAD OF trigger to fire, referential integrity will be checked against the updates performed in the trigger, and not against the underlying tables of the view that caused the trigger to fire.
- **Referential integrity:** If the identified table or the base table of the identified view is a parent, the rows selected for delete must not have any dependents in a relationship with a delete rule of RESTRICT, and the DELETE must not cascade to descendent rows that have dependents in a relationship with a delete rule of RESTRICT.

If the delete operation is not prevented by a RESTRICT delete rule, the selected rows are deleted. Any rows that are dependents of the selected rows are also affected:

- The nullable columns of the foreign keys of any rows that are their dependents in a relationship with a delete rule of SET NULL are set to the null value.
- Any rows that are their dependents in a relationship with a delete rule of CASCADE are also deleted, and the above rules apply, in turn, to those rows.

The delete rule of NO ACTION is checked to enforce that any non-null foreign key refers to an existing parent row after the other referential constraints have been enforced.

- **Security policy:** If the identified table or the base table of the identified view is protected with a security policy, the session authorization ID must have the label-based access control (LBAC) credentials that allow:
 - Write access to all protected columns (SQLSTATE 42512)

- Read and write access to all of the rows that are selected for deletion (SQLSTATE 42519)

Notes

- If an error occurs during the execution of a multiple row DELETE, no changes are made to the database.
- Unless appropriate locks already exist, one or more exclusive locks are acquired during the execution of a successful DELETE statement. Issuing a COMMIT or ROLLBACK statement will release the locks. Until the locks are released by a commit or rollback operation, the effect of the delete operation can only be perceived by:
 - The application process that performed the deletion
 - Another application process using isolation level UR.

The locks can prevent other application processes from performing operations on the table.

- If an application process deletes a row on which any of its cursors are positioned, those cursors are positioned before the next row of their result table. Let C be a cursor that is positioned before row R (as a result of an OPEN, a DELETE through C, a DELETE through some other cursor, or a searched DELETE). In the presence of INSERT, UPDATE, and DELETE operations that affect the base table from which R is derived, the next FETCH operation referencing C does not necessarily position C on R. For example, the operation can position C on R', where R' is a new row that is now the next row of the result table.
- SQLERRD(3) in the SQLCA shows the number of rows that qualified for the delete operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. SQLERRD(5) in the SQLCA shows the number of rows affected by referential constraints and by triggered statements. It includes rows that were deleted as a result of a CASCADE delete rule and rows in which foreign keys were set to NULL as the result of a SET NULL delete rule. With regards to triggered statements, it includes the number of rows that were inserted, updated, or deleted.
- If an error occurs that prevents deleting all rows matching the search condition and all operations required by existing referential constraints, no changes are made to the table and the error is returned.
- For nicknames, the external server option iud_app_svpt_enforce poses an additional limitation. Refer to the Federated documentation for more information.
- For some data sources, the SQLCODE -20190 may be returned on a delete against a nickname because of potential data inconsistency. Refer to the Federated documentation for more information.
- **Compatibilities:** The following syntax is supported, but is non-standard and should not be used:
 - The FROM keyword can be omitted.

Examples

Example 1: Delete department (DEPTNO) 'D11' from the DEPARTMENT table.

```
DELETE FROM DEPARTMENT
WHERE DEPTNO = 'D11'
```

Example 2: Delete all the departments from the DEPARTMENT table (that is, empty the table).

```
DELETE FROM DEPARTMENT
```

Example 3: Delete from the EMPLOYEE table any sales rep or field rep who didn't make a sale in 1995.

```
DELETE FROM EMPLOYEE
WHERE LASTNAME NOT IN
(SELECT SALES_PERSON
 FROM SALES
 WHERE YEAR(SALES_DATE)=1995)
AND JOB IN ('SALESREP', 'FIELDREP')
```

Example 4: Delete all the duplicate employee rows from the EMPLOYEE table. An employee row is considered to be a duplicate if the last names match. Keep the employee row with the smallest first name in lexical order.

```
DELETE FROM
(SELECT ROWNUMBER() OVER (PARTITION BY LASTNAME ORDER BY FIRSTNAME)
 FROM EMPLOYEE) AS E(RN)
WHERE RN = 1
```

DROP

The DROP statement deletes an object. Any objects that are directly or indirectly dependent on that object are either deleted or made inoperative. Whenever an object is deleted, its description is deleted from the catalog, and any packages that reference the object are invalidated.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

When dropping objects that allow two-part names, the privileges held by the authorization ID of the statement must include at least one of the following:

- DROPIN privilege on the schema for the object
- Owner of the object, as recorded in the OWNER column of the catalog view for the object
- CONTROL privilege on the object (applicable only to indexes, index specifications, nicknames, packages, tables, and views)
- Owner of the user-defined type, as recorded in the OWNER column of the SYSCAT.DATATYPES catalog view (applicable only when dropping a method that is associated with a user-defined type)
- DBADM authority

When dropping a table or view hierarchy, the privileges held by the authorization ID of the statement must include one of the above privileges for each of the tables or views in the hierarchy.

When dropping an audit policy, the privileges held by the authorization ID of the statement must include SECADM authority.

| When dropping a buffer pool, database partition group or table space, the
| privileges held by the authorization ID of the statement must include SYSADM or
| SYSCTRL authority.

| When dropping a data type mapping, function mapping, server definition, or
| wrapper, the privileges held by the authorization ID of the statement must include
| DBADM authority.

| When dropping an event monitor the privilege held by the authorization ID of the
| statement must include SQLADM or DBADM authority.

| When dropping a role the privileges held by the authorization ID of the statement
| must include SECADM authority.

| When dropping a schema, the privileges held by the authorization ID of the
| statement must include DBADM authority, or be the schema owner, as recorded in
| the OWNER column of the SYSCAT.SCHEMATA catalog view.

| When dropping a security label, a security label component, or a security policy,
| the privileges held by the authorization ID of the statement must include SECADM
| authority.

| When dropping a service class, work action set, work class set, workload,
| threshold, or histogram template, the privileges held by the authorization ID of the
| statement must include WLMADM or DBADM authority.

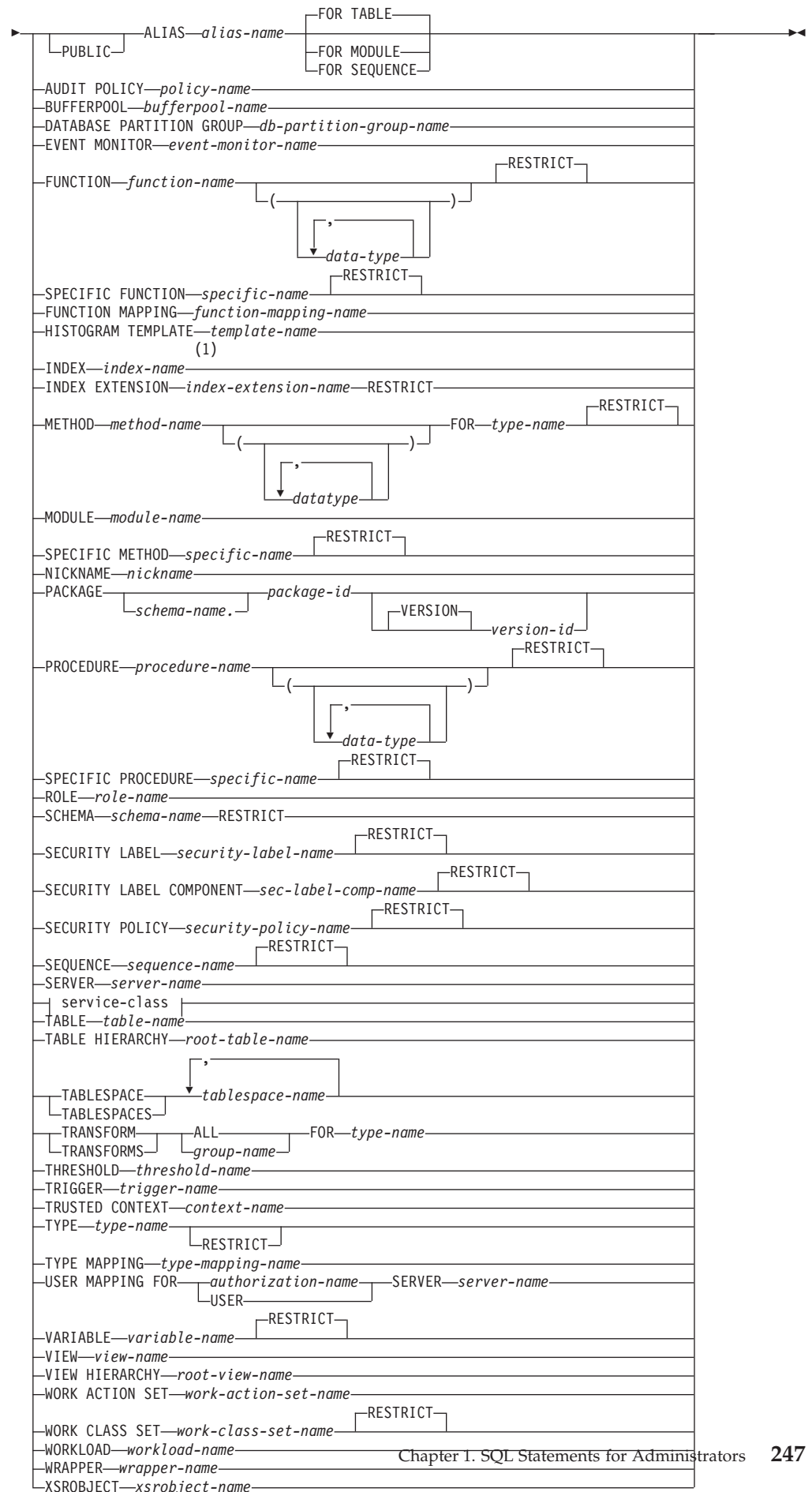
| When dropping a transform, the privileges held by the authorization ID of the
| statement must include DBADM authority, or must be the owner of *type-name*.

| When dropping a trusted context, the privileges held by the authorization ID of
| the statement must include SECADM authority.

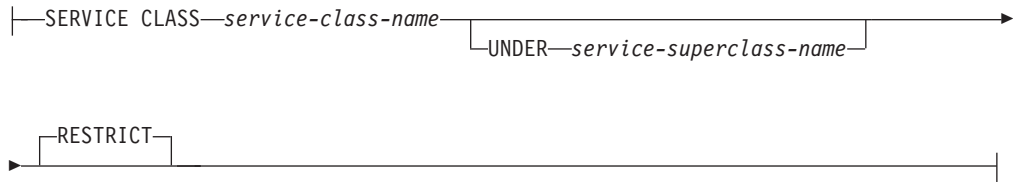
| When dropping a user mapping, the privileges held by the authorization ID of the
| statement must include DBADM authority, if this authorization ID is different from
| the federated database authorization name within the mapping. Otherwise, if the
| authorization ID and the authorization name match, no authorities or privileges
| are required.

Syntax

►►—DROP—————►



service-class:



Notes:

- 1 *Index-name* can be the name of either an index or an index specification.

Description

ALIAS *alias-name*

Identifies the alias that is to be dropped. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704). The specified alias is deleted.

FOR TABLE, FOR MODULE, or FOR SEQUENCE

Specifies the object type for the alias.

FOR TABLE

The alias is for a table, view, or nickname.

FOR MODULE

The alias is for a module.

FOR SEQUENCE

The alias is for a sequence.

The specified alias is deleted. All tables, views, and triggers that reference the alias are made inoperative. This includes both the table referenced in the ON clause of the CREATE TRIGGER statement, and all tables referenced within the triggered SQL statements.

If PUBLIC is specified, the *alias-name* must identify a public alias that exists at the current server (SQLSTATE 42704).

AUDIT POLICY *policy-name*

Identifies the audit policy that is to be dropped. The *policy-name* must identify an audit policy that exists at the current server (SQLSTATE 42704). The audit policy must not be associated with any database objects (SQLSTATE 42893). The specified audit policy is deleted from the catalog.

BUFFERPOOL *bufferpool-name*

Identifies the buffer pool that is to be dropped. The *bufferpool-name* must identify a buffer pool that is described in the catalog (SQLSTATE 42704). There can be no table spaces assigned to the buffer pool (SQLSTATE 42893). The IBMDEFAULTBP buffer pool cannot be dropped (SQLSTATE 42832). Buffer pool memory is released immediately, to be used by DB2. Disk storage may not be released until the next connection to the database.

DATABASE PARTITION GROUP *db-partition-group-name*

Identifies the database partition group that is to be dropped. The *db-partition-group-name* parameter must identify a database partition group that is described in the catalog (SQLSTATE 42704). This is a one-part name.

Dropping a database partition group drops all table spaces defined in the database partition group. All existing database objects with dependencies on the tables in the table spaces (such as packages, referential constraints, and so on) are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

System-defined database partition groups cannot be dropped (SQLSTATE 42832).

If a DROP DATABASE PARTITION GROUP statement is issued against a database partition group that is currently undergoing a data redistribution, the drop database partition group operation fails, and an error is returned (SQLSTATE 55038). However, a partially redistributed database partition group can be dropped. A database partition group can become partially redistributed if a REDISTRIBUTE DATABASE PARTITION GROUP command does not execute to completion. This can happen if it is interrupted by either an error or a FORCE APPLICATION ALL command. (For a partially redistributed database partition group, the REBALANCE_PMAP_ID in the SYSCAT.DBPARTITIONGROUPS catalog is not -1.)

EVENT MONITOR *event-monitor-name*

Identifies the event monitor that is to be dropped. The *event-monitor-name* must identify an event monitor that is described in the catalog (SQLSTATE 42704).

If the identified event monitor is active, an error is returned (SQLSTATE 55034); otherwise, the event monitor is deleted. Note that if an event monitor has been previously activated using the SET EVENT MONITOR STATE statement, and the database has been deactivated and subsequently reactivated, use the SET EVENT MONITOR STATE statement to deactivate the event monitor before issuing the DROP statement.

If there are event files in the target path of a WRITE TO FILE event monitor that is being dropped, the event files are not deleted. However, if a new event monitor that specifies the same target path is created, the event files are deleted.

When dropping WRITE TO TABLE event monitors, table information is removed from the SYSCAT.EVENTTABLES catalog view, but the tables themselves are not dropped.

FUNCTION

Identifies an instance of a user-defined function (either a complete function or a function template) that is to be dropped. The function instance specified must be a user-defined function described in the catalog. Functions implicitly generated by the CREATE TYPE (Distinct) statement cannot be dropped.

There are several different ways available to identify the function instance:

FUNCTION *function-name*

Identifies the particular function, and is valid only if there is exactly one function instance with the *function-name*. The function thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the function in the named or implied schema, an error is returned (SQLSTATE 42725).

FUNCTION *function-name (data-type,...)*

Provides the function signature, which uniquely identifies the function to be dropped. The function selection algorithm is not used.

function-name

Gives the function name of the function to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)

Must match the data types that were specified on the CREATE FUNCTION statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific function instance which is to be dropped.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(n) does not need to match the defined value for n since $0 < n < 25$ means REAL and $24 < n < 54$ means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

RESTRICT

The RESTRICT keyword enforces the rule that the function is not to be dropped if any of the following dependencies exists:

- Another routine is sourced on the function.
- A view uses the function.
- A trigger uses the function.
- A materialized query table uses the function in its definition.

RESTRICT is the default behavior.

If no function with the specified signature exists in named or implied schema, an error is returned (SQLSTATE 42883).

SPECIFIC FUNCTION *specific-name*

Identifies the particular user-defined function that is to be dropped, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

RESTRICT

The RESTRICT keyword enforces the rule that the function is not to be dropped if any of the following dependencies exists:

- Another routine is sourced on the function.
- A view uses the function.
- A trigger uses the function.

RESTRICT is the default behavior.

It is not possible to drop a function that is in the SYSIBM, SYSFUN, or the SYSPROC schema (SQLSTATE 42832).

Other objects can be dependent upon a function. All such dependencies must be removed before the function can be dropped, with the exception of packages which are marked inoperative. An attempt to drop a function with such dependencies will result in an error (SQLSTATE 42893). See “Rules” on page 264 for a list of these dependencies.

If the function can be dropped, it is dropped.

Any package dependent on the specific function being dropped is marked as inoperative. Such a package is not implicitly rebound. It must either be rebound by use of the BIND or REBIND command, or it must be re-prepared by use of the PREP command.

FUNCTION MAPPING *function-mapping-name*

Identifies the function mapping that is to be dropped. The *function-mapping-name* must identify a user-defined function mapping that is described in the catalog (SQLSTATE 42704). The function mapping is deleted from the database.

Default function mappings cannot be dropped, but can be disabled by using the CREATE FUNCTION MAPPING statement. Dropping a user-defined function mapping that was created to override a default function mapping reinstates the default function mapping.

Packages having a dependency on a dropped function mapping are invalidated.

HISTOGRAM TEMPLATE *template-name*

Identifies the histogram template that is to be dropped. The *template-name* must identify a histogram template that exists at the current server (SQLSTATE 42704). The *template-name* cannot be SYSDEFAULTHISTOGRAM (SQLSTATE 42832). The histogram template cannot be dropped if a service class or a work action is dependent on it (SQLSTATE 42893). The specified histogram template is deleted from the catalog.

INDEX *index-name*

Identifies the index or index specification that is to be dropped. The *index-name* must identify an index or index specification that is described in the catalog (SQLSTATE 42704). It cannot be an index that is required by the system for a primary key or unique constraint, for a replicated materialized query table, or for an XML column (SQLSTATE 42917). The specified index or index specification is deleted.

Packages having a dependency on a dropped index or index specification are invalidated.

INDEX EXTENSION *index-extension-name* RESTRICT

Identifies the index extension that is to be dropped. The *index-extension-name* must identify an index extension that is described in the catalog (SQLSTATE

42704). The RESTRICT keyword enforces the rule that no index can be defined that depends on this index extension definition (SQLSTATE 42893).

METHOD

Identifies a method body that is to be dropped. The method body specified must be a method described in the catalog (SQLSTATE 42704). Method bodies that are implicitly generated by the CREATE TYPE statement cannot be dropped.

DROP METHOD deletes the body of a method, but the method specification (signature) remains as a part of the definition of the subject type. After dropping the body of a method, the method specification can be removed from the subject type definition by ALTER TYPE DROP METHOD.

There are several ways available to identify the method body to be dropped:

METHOD *method-name*

Identifies the particular method to be dropped, and is valid only if there is exactly one method instance with name *method-name* and subject type *type-name*. Thus, the method identified may have any number of parameters. If no method by this name exists for the type *type-name*, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the method for the named data type, an error is returned (SQLSTATE 42725).

METHOD *method-name (data-type,...)*

Provides the method signature, which uniquely identifies the method to be dropped. The method selection algorithm is not used.

method-name

The method name of the method to be dropped for the specified type. The name must be an unqualified identifier.

(data-type, ...)

Must match the data types that were specified in the corresponding positions of the method-specification of the CREATE TYPE or ALTER TYPE statement. The number of data types and the logical concatenation of the data types are used to identify the specific method instance which is to be dropped.

If the data-type is unqualified, the type name is resolved by searching the schemas on the SQL path.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

A type of FLOAT(n) does not need to match the defined value for n since $0 < n < 25$ means REAL and $24 < n < 54$ means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no method with the specified signature exists for the named data type, an error is returned (SQLSTATE 42883).

FOR *type-name*

Names the type for which the specified method is to be dropped. The

name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names.

RESTRICT

The RESTRICT keyword enforces the rule that the method is not to be dropped if any of the following dependencies exists:

- Another routine is sourced on the method.
- A view uses the method.
- A trigger uses the method.

RESTRICT is the default behavior.

MODULE *module-name*

Identifies the module that is to be dropped. The *module-name* must identify a module that exists at the current server (SQLSTATE 42704). The specified module is dropped from the schema, including all module objects. All privileges on the module are also dropped.

SPECIFIC METHOD *specific-name*

Identifies the particular method that is to be dropped, using a name either specified or defaulted to at CREATE TYPE or ALTER TYPE time. If the specific name is unqualified, the CURRENT SCHEMA special register is used as a qualifier for an unqualified specific name in dynamic SQL. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for an unqualified specific name. The *specific-name* must identify a method; otherwise, an error is returned (SQLSTATE 42704).

RESTRICT

The RESTRICT keyword enforces the rule that the method is not to be dropped if any of the following dependencies exists:

- Another routine is sourced on the method.
- A view uses the method.
- A trigger uses the function.

RESTRICT is the default method.

Other objects can be dependent upon a method. All such dependencies must be removed before the method can be dropped, with the exception of packages which will be marked inoperative if the drop is successful. An attempt to drop a method with such dependencies will result in an error (SQLSTATE 42893).

If the method can be dropped, it will be dropped.

Any package dependent on the specific method being dropped is marked as inoperative. Such a package is not implicitly re-bound. Either it must be re-bound by use of the BIND or REBIND command, or it must be re-prepared by use of the PREP command.

If the specific method being dropped overrides another method, all packages dependent on the overridden method — and on methods that override this method in supertypes of the specific method being dropped — are invalidated.

NICKNAME *nickname*

Identifies the nickname that is to be dropped. The nickname must be listed in the catalog (SQLSTATE 42704). The nickname is deleted from the database.

All information about the columns and indexes associated with the nickname is deleted from the catalog. Any materialized query tables that are dependent on the nickname are dropped. Any index specifications that are dependent on the nickname are dropped. Any views that are dependent on the nickname are marked inoperative. Any packages that are dependent on the dropped index specifications or inoperative views are invalidated. The data source table that the nickname references is not affected.

If an SQL function or method is dependent on a nickname, that nickname cannot be dropped (SQLSTATE 42893).

PACKAGE *schema-name.package-id*

Identifies the package that is to be dropped. If a schema name is not specified, the package identifier is implicitly qualified by the default schema. The schema name and package identifier, together with the implicitly or explicitly specified version identifier, must identify a package that is described in the catalog (SQLSTATE 42704). The specified package is deleted. If the package being dropped is the only package identified by *schema-name.package-id* (that is, there are no other versions), all privileges on the package are also deleted.

VERSION *version-id*

Identifies which package version is to be dropped. If a value is not specified, the version defaults to the empty string. If multiple packages with the same package name but different versions exist, only one package version can be dropped in one invocation of the DROP statement. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

PROCEDURE

Identifies an instance of a procedure that is to be dropped. The procedure instance specified must be a procedure described in the catalog.

There are several different ways available to identify the procedure instance:

PROCEDURE *procedure-name*

Identifies the particular procedure to be dropped, and is valid only if there is exactly one procedure instance with the *procedure-name* in the schema. The procedure thus identified may have any number of parameters defined for it. If no procedure by this name exists in the named or implied schema, an error is returned (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If there is more than one specific instance of the procedure in the named or implied schema, an error (SQLSTATE 42725) is returned.

PROCEDURE *procedure-name (data-type,...)*

Provides the procedure signature, which uniquely identifies the procedure to be dropped. The procedure selection algorithm is not used. For federated procedures, the signature information is not specified on the CREATE PROCEDURE statement, but the information is available in the system catalog.

procedure-name

Gives the procedure name of the procedure to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)

Must match the data types that were specified on the CREATE PROCEDURE statement in the corresponding position, except for federated procedures, where the data type must match what is stored in the local catalog for the corresponding parameter. The number of data types, and the logical concatenation of the data types is used to identify the specific procedure instance which is to be dropped.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement or, for federated procedures, it must exactly match what is stored in the local catalog for the corresponding parameter.

A type of FLOAT(n) does not need to match the defined value for n since $0 < n < 25$ means REAL and $24 < n < 54$ means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

RESTRICT

The RESTRICT keyword prevents the procedure from being dropped if a trigger definition, an SQL function, or an SQL method contains a CALL statement with the name of the procedure. RESTRICT is the default behavior.

If no procedure with the specified signature exists in named or implied schema, an error (SQLSTATE 42883) is returned.

SPECIFIC PROCEDURE *specific-name*

Identifies the particular procedure that is to be dropped, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

RESTRICT

The RESTRICT keyword prevents the procedure from being dropped if a trigger definition, an SQL function, or an SQL method contains a CALL statement with the name of the procedure. RESTRICT is the default behavior.

It is not possible to drop a procedure that is in the SYSIBM, SYSFUN, or the SYSPROC schema (SQLSTATE 42832).

ROLE *role-name*

Identifies the role that is to be dropped. The *role-name* must identify a role that already exists at the current server (SQLSTATE 42704). The *role-name* must not identify a role, or a role that contains *role-name*, if the role has either EXECUTE privilege on a routine or USAGE privilege on a sequence, and an SQL object other than a package is dependent on the routine or sequence (SQLSTATE 42893). The owner of the SQL object is either *authorization-name* or any user who is a member of *authorization-name*, where *authorization-name* is a role.

A DROP ROLE statement fails (SQLSTATE 42893) if any of the following is true for the role to be dropped:

- A workload exists such that one of the values for the connection attribute SESSION_USER ROLE is *role-name*
- A trusted context using *role-name* exists

The specified role is deleted from the catalog.

SCHEMA *schema-name* **RESTRICT**

Identifies the particular schema to be dropped. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704). The RESTRICT keyword enforces the rule that no objects can be defined in the specified schema for the schema to be deleted from the database (SQLSTATE 42893).

SECURITY LABEL *security-label-name*

Identifies the security label to be dropped. The name must be qualified with a security policy (SQLSTATE 42704) and must identify a security label that exists at the current server (SQLSTATE 42704).

RESTRICT

This option, which is the default, prevents the security label from being dropped if any of the following dependencies exist (SQLSTATE 42893):

- One or more authorization IDs currently hold the security label for read access
- One or more authorization IDs currently hold the security label for write access
- The security label is currently being used to protect one or more columns

SECURITY LABEL COMPONENT *sec-label-comp-name*

Identifies the security label component to be dropped. The *sec-label-comp-name* must identify a security label component that is described in the catalog (SQLSTATE 42704).

RESTRICT

This option, which is the default, prevents the security label component from being dropped if any of the following dependencies exist (SQLSTATE 42893):

- One or more security policies that include the security label component are currently defined

SECURITY POLICY *security-policy-name*

Identifies the security policy to be dropped. The *security-policy-name* must identify a security policy that exists at the current server (SQLSTATE 42704).

RESTRICT

This option, which is the default, prevents the security policy from being dropped if any of the following dependencies exist (SQLSTATE 42893):

- One or more tables are associated with this security policy
- One or more authorization IDs hold an exemption on one of the rules in this security policy
- One or more security labels are defined for this security policy

SEQUENCE *sequence-name*

Identifies the particular sequence that is to be dropped. The *sequence-name*, along with the implicit or explicit schema name, must identify an existing sequence at the current server. If no sequence by this name exists in the explicitly or implicitly specified schema, an error is returned (SQLSTATE 42704).

RESTRICT

This option, which is the default, prevents the sequence from being dropped if any of the following dependencies exist:

- A trigger exists such that a NEXT VALUE or PREVIOUS VALUE expression in the trigger specifies the sequence (SQLSTATE 42893).
- An SQL function or an SQL method exists such that a NEXT VALUE expression in the routine body specifies the sequence (SQLSTATE 42893).

SERVER *server-name*

Identifies the data source whose definition is to be dropped from the catalog. The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The definition of the data source is deleted.

All nicknames for tables and views residing at the data source are dropped. Any index specifications dependent on these nicknames are dropped. Any user-defined function mappings, user-defined type mappings, and user mappings that are dependent on the dropped server definition are also dropped. All packages dependent on the dropped server definition, function mappings, nicknames, and index specifications are invalidated. All federated procedures that are dependent on the server definition are also dropped.

SERVICE CLASS *service-class-name*

Identifies the service class to be dropped. The *service-class-name* must identify a service class that is described in the catalog (SQLSTATE 42704). To drop a service subclass, the *service-superclass-name* must be specified using the UNDER clause.

UNDER *service-superclass-name*

Specifies the service superclass of the service subclass when dropping a service subclass. The *service-superclass-name* must identify a service superclass that is described in the catalog (SQLSTATE 42704).

RESTRICT

This keyword enforces the rule that the service class is not to be dropped if any of the following dependencies exists:

- The service class is a service superclass and there is a user defined service subclass under the service class (SQLSTATE 5U031). The service subclass must first be dropped.
- The service class is a service superclass and there is a work action set mapping to the service class (SQLSTATE 5U031). The work action set must first be dropped.

- The service class is a service subclass and there is a work action mapping to the service class (SQLSTATE 5U031). The work action must first be dropped.
- The service class has a workload mapping (SQLSTATE 5U031). The workload mapping must first be removed. Remove the workload mapping by dropping the workload or altering the workload to not map to the service class.
- The service class has an associated threshold (SQLSTATE 5U031). The threshold must first be dropped.
- The service class is the target of a REMAP ACTIVITY action in a threshold (SQLSTATE 5U031). Alter the threshold to set a different service subclass as the target of the REMAP ACTIVITY action or drop the threshold.
- The service class is not disabled (SQLSTATE 5U031). The service class must first be disabled.

RESTRICT is the default behavior.

TABLE *table-name*

Identifies the base table, created temporary table, or declared temporary table that is to be dropped. The *table-name* must identify a table that is described in the catalog or, if it is a declared temporary table, the *table-name* must be qualified by the schema name SESSION and exist in the application (SQLSTATE 42704). The subtables of a typed table are dependent on their supertables. All subtables must be dropped before a supertable can be dropped (SQLSTATE 42893). The specified table is deleted from the database.

All indexes, primary keys, foreign keys, check constraints, materialized query tables, and staging tables referencing the table are dropped. All views and triggers that reference the table are made inoperative. (This includes both the table referenced in the ON clause of the CREATE TRIGGER statement, and all tables referenced within the triggered SQL statements.) All packages depending on any object dropped or marked inoperative will be invalidated. This includes packages dependent on any supertables above the subtable in the hierarchy. Any reference columns for which the dropped table is defined as the scope of the reference become unscoped.

Packages are not dependent on declared temporary tables, and therefore are not invalidated when such a table is dropped. Packages are, however, dependent on created temporary tables, and are invalidated when such a table is dropped.

In a federated system, a remote table that was created using transparent DDL can be dropped. Dropping a remote table also drops the nickname associated with that table, and invalidates any packages that are dependent on that nickname.

When a subtable is dropped from a table hierarchy, the columns associated with the subtable are no longer accessible although they continue to be considered with respect to limits on the number of columns and size of the row. Dropping a subtable has the effect of deleting all the rows of the subtable from the supertables. This may result in activation of triggers or referential integrity constraints defined on the supertables.

When a created temporary table or declared temporary table is dropped, and its creation preceded the active unit of work or savepoint, then the table will be functionally dropped and the application will not be able to access the table. However, the table will still reserve some space in its table space and will

prevent that USER TEMPORARY table space from being dropped or the database partition group of the USER TEMPORARY table space from being redistributed until the unit of work is committed or savepoint is ended. Dropping a created temporary table or declared temporary table causes the data in the table to be destroyed, regardless of whether DROP is committed or rolled back.

A table cannot be dropped if it has the RESTRICT ON DROP attribute.

A newly detached table is initially inaccessible. This prevents the table from being read, modified, or dropped until the SET INTEGRITY statement can be run to incrementally refresh MQTs or to complete any processing for foreign key constraints. After the SET INTEGRITY statement executes against all dependent tables, the table is fully accessible, its detached attribute is reset, and it can be dropped.

TABLE HIERARCHY *root-table-name*

Identifies the typed table hierarchy that is to be dropped. The *root-table-name* must identify a typed table that is the root table in the typed table hierarchy (SQLSTATE 428DR). The typed table identified by *root-table-name* and all of its subtables are deleted from the database.

All indexes, materialized query tables, staging tables, primary keys, foreign keys, and check constraints referencing the dropped tables are dropped. All views and triggers that reference the dropped tables are made inoperative. All packages depending on any object dropped or marked inoperative will be invalidated. Any reference columns for which one of the dropped tables is defined as the scope of the reference become unscoped.

Unlike dropping a single subtable, dropping the table hierarchy does not result in the activation of delete triggers of any tables in the hierarchy nor does it log the deleted rows.

TABLESPACE or TABLESPACES *tablespace-name*

Identifies the table spaces that are to be dropped; *tablespace-name* must identify a table space that is described in the catalog (SQLSTATE 42704). This is a one-part name.

The table spaces will not be dropped (SQLSTATE 55024) if there is any table that stores at least one of its parts in a table space being dropped, and has one or more of its parts in another table space that is not being dropped (these tables would need to be dropped first), or if any table that resides in the table space has the RESTRICT ON DROP attribute.

Objects whose names are prefixed with 'SYS' are system-defined objects and, with the exception of the SYSTOOLSPACE and SYSTOOLSTMPSPACE table spaces, cannot be dropped (SQLSTATE 42832).

A SYSTEM TEMPORARY table space cannot be dropped (SQLSTATE 55026) if it is the only temporary table space that exists in the database. A USER TEMPORARY table space cannot be dropped if there is an instance of a created temporary table or a declared temporary table created in it (SQLSTATE 55039). Even if a created temporary table has been dropped, the USER TEMPORARY table space will still be considered to be in use until all instances of the created temporary table are dropped. Instances of a created temporary table are dropped when the session terminates or when the created temporary table is referenced in the session. Even if a declared temporary table has been dropped, the USER TEMPORARY table space will still be considered to be in use until the unit of work containing the DROP TABLE statement has been committed.

Dropping a table space drops all objects that are defined in the table space. All existing database objects with dependencies on the table space, such as packages, referential constraints, and so on, are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

Containers that were created by a user are not deleted. Any directories in the path of the container name that were created by the database manager during CREATE TABLESPACE execution are deleted. All containers that are below the database directory are deleted. When the DROP TABLESPACE statement is committed, the DMS file containers or SMS containers for the specified table space are deleted, if possible. If the containers cannot be deleted (because they are being kept open by another agent, for example), the files are truncated to zero length. After all connections are terminated, or the DEACTIVATE DATABASE command is issued, these zero-length files are deleted.

THRESHOLD *threshold-name*

Identifies the threshold that is to be dropped. The *threshold-name* must identify a threshold that exists at the current server (SQLSTATE 42704). This is a one-part name. Thresholds with a queue, for example TOTALSCPARTITIONCONNECTIONS and CONCURRENTDBCOORDACTIVITIES, must be disabled before they can be dropped (SQLSTATE 5U025). The specified threshold is deleted from the catalog.

TRIGGER *trigger-name*

Identifies the trigger that is to be dropped. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704). The specified trigger is deleted.

Dropping triggers causes certain packages to be marked invalid.

If *trigger-name* specifies an INSTEAD OF trigger on a view, another trigger may depend on that trigger through an update against the view.

TRANSFORM ALL FOR *type-name*

Indicates that all transforms groups defined for the user-defined data type *type-name* are to be dropped. The transform functions referenced in these groups are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704).

If there are not transforms defined for *type-name*, an error is returned (SQLSTATE 42740).

DROP TRANSFORM is the inverse of CREATE TRANSFORM. It causes the transform functions associated with certain groups, for a given data type, to become undefined. The functions formerly associated with these groups still exist and can still be called explicitly, but they no longer have the transform property, and are no longer invoked implicitly for exchanging values with the host language environment.

The transform group is not dropped if there is a user-defined function (or method) written in a language other than SQL that has a dependency on one of the group's transform functions defined for the user-defined type *type-name* (SQLSTATE 42893). Such a function has a dependency on the transform function associated with the referenced transform group defined for type *type-name*. Packages that depend on a transform function associated with the named transform group are marked inoperative.

TRANSFORMS *group-name* **FOR** *type-name*

Indicates that the specified transform group for the user-defined data type *type-name* is to be dropped. The transform functions referenced in this group are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704), and the *group-name* must identify an existing transform group for *type-name*.

TRIGGER *trigger-name*

Identifies the trigger that is to be dropped. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704). The specified trigger is deleted.

Dropping triggers causes certain packages to be marked invalid.

If *trigger-name* specifies an INSTEAD OF trigger on a view, another trigger may depend on that trigger through an update against the view.

TRUSTED CONTEXT *context-name*

Identifies the trusted context that is to be dropped. The *context-name* must identify a trusted context that exists at the current server (SQLSTATE 42704). If the trusted context is dropped while trusted connections for this context are active, those connections remain trusted until they terminate or until the next reuse attempt. If an attempt is made to switch the user on these trusted connections, an error is returned (SQLSTATE 42517). The specified trusted context is deleted from the catalog.

TYPE *type-name*

Identifies the user-defined type to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. For a structured type, the associated reference type is also dropped. The *type-name* must identify a user-defined type described in the catalog.

RESTRICT

The type is not dropped (SQLSTATE 42893) if any of the following is true:

- The type is used as the type of a column of a table or view.
- The type has a subtype.
- The type is a structured type used as the data type of a typed table or a typed view.
- The type is an attribute of another structured type.
- There exists a column of a table whose type might contain an instance of *type-name*. This can occur if *type-name* is the type of the column or is used elsewhere in the column's associated type hierarchy. More formally, for any type T, T cannot be dropped if there exists a column of a table whose type directly or indirectly uses *type-name*.
- The type is the target type of a reference-type column of a table or view, or a reference-type attribute of another structured type.
- The type, or a reference to the type, is a parameter type or a return value type of a function or method.
- The type is a parameter type or is used in the body of an SQL procedure.

- The type, or a reference to the type, is used in the body of an SQL function or method, but it is not a parameter type or a return value type.
- The type is used in a check constraint, trigger, view definition, or index extension.

If RESTRICT is not specified, the behavior is the same as RESTRICT, except for functions and methods that use the type.

Functions that use the type: If the user-defined type can be dropped, then for every function, F (with specific name SF), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following DROP FUNCTION statement is effectively executed:

```
DROP SPECIFIC FUNCTION SF
```

It is possible that this statement also would cascade to drop dependent functions. If all of these functions are also in the list to be dropped because of a dependency on the user-defined type, the drop of the user-defined type will succeed (otherwise it fails with SQLSTATE 42893).

Methods that use the type: If the user-defined type can be dropped, then for every method, M of type T1 (with specific name SM), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following statements are effectively executed:

```
DROP SPECIFIC METHOD SM  
ALTER TYPE T1 DROP SPECIFIC METHOD SM
```

The existence of objects that are dependent on these methods may cause the DROP TYPE operation to fail.

All packages that are dependent on methods defined in supertypes of the type being dropped, and that are eligible for overriding, are invalidated.

TYPE MAPPING *type-mapping-name*

Identifies the user-defined data type mapping to be dropped. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704). The data type mapping is deleted from the database.

No additional objects are dropped.

USER MAPPING FOR *authorization-name* | **USER SERVER** *server-name*

Identifies the user mapping to be dropped. This mapping associates an authorization name that is used to access the federated database with an authorization name that is used to access a data source. The first of these two authorization names is either identified by the *authorization-name* or referenced by the special register USER. The *server-name* identifies the data source that the second authorization name is used to access.

The *authorization-name* must be listed in the catalog (SQLSTATE 42704). The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The user mapping is deleted.

No additional objects are dropped.

VARIABLE *variable-name*

Identifies the global variable that is to be dropped. The *variable-name* must identify a global variable that exists at the current server (SQLSTATE 42704).

RESTRICT

This keyword enforces the rule that the global variable cannot be dropped

if it is referenced in a function, method, trigger, or view (SQLSTATE 42893). RESTRICT is the default behavior.

VIEW *view-name*

Identifies the view that is to be dropped. The *view-name* must identify a view that is described in the catalog (SQLSTATE 42704). The subviews of a typed view are dependent on their superviews. All subviews must be dropped before a superview can be dropped (SQLSTATE 42893).

The specified view is deleted. The definition of any view or trigger that is directly or indirectly dependent on that view is marked inoperative. Any materialized query table or staging table that is dependent on any view that is marked inoperative is dropped. Any packages dependent on a view that is dropped or marked inoperative will be invalidated. This includes packages dependent on any superviews above the subview in the hierarchy. Any reference columns for which the dropped view is defined as the scope of the reference become unscoped.

VIEW HIERARCHY *root-view-name*

Identifies the typed view hierarchy that is to be dropped. The *root-view-name* must identify a typed view that is the root view in the typed view hierarchy (SQLSTATE 428DR). The typed view identified by *root-view-name* and all of its subviews are deleted from the database.

The definition of any view or trigger that is directly or indirectly dependent on any of the dropped views is marked inoperative. Any packages dependent on any view or trigger that is dropped or marked inoperative will be invalidated. Any reference columns for which a dropped view or view marked inoperative is defined as the scope of the reference become unscoped.

WORK ACTION SET *work-action-set-name*

Identifies the work action set that is to be dropped. The *work-action-set-name* must identify a work action set that exists at the current server (SQLSTATE 42704). All work actions that are contained by the *work-action-set-name* are also dropped.

WORK CLASS SET *work-class-set-name*

Identifies the work class set that is to be dropped. The *work-class-set-name* must identify a work class set that exists at the current server (SQLSTATE 42704). All work classes that are contained by the *work-class-set-name* are also dropped.

RESTRICT

This keyword enforces the rule that the work class set is not to be dropped if it is associated with any work action set (SQLSTATE 42893). RESTRICT is the default behavior.

WORKLOAD *workload-name*

Identifies the workload that is to be dropped. This is a one-part name. The *workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). SYSDEFAULTUSERWORKLOAD or SYSDEFAULTADMWORKLOAD cannot be dropped (SQLSTATE 42832). A workload must be disabled and must not have active workload occurrences associated with it before it can be dropped (SQLSTATE 5U023). The specified workload is deleted from the catalog.

WRAPPER *wrapper-name*

Identifies the wrapper to be dropped. The *wrapper-name* must identify a wrapper that is described in the catalog (SQLSTATE 42704). The wrapper is deleted.

All server definitions, user-defined function mappings, and user-defined data type mappings that are dependent on the wrapper are dropped. All user-defined function mappings, nicknames, user-defined data type mappings, and user mappings that are dependent on the dropped server definitions are also dropped. Any index specifications dependent on the dropped nicknames are dropped, and any views dependent on these nicknames are marked inoperative. All packages dependent on the dropped objects and inoperative views are invalidated. All federated procedures that are dependent on the dropped server definitions are also dropped.

XSRBJECT *xsobject-name*

Identifies the XSR object to be dropped. The *xsobject-name* must identify an XSR object that is described in the catalog (SQLSTATE 42704).

Check constraints that reference the XSR object are dropped. All triggers and views referencing the XSR object are marked inoperative. Packages having a dependency on a dropped XSR object are invalidated.

In a partitioned database environment, you can issue this statement against an XSR object by connecting to any partition.

Rules

Dependencies: Table 15 on page 265 shows the dependencies that objects have on each other. Not all dependencies are explicitly recorded in the catalog. For example, there is no record of the constraints on which a package has dependencies. Four different types of dependencies are shown:

- R** Restrict semantics. The underlying object cannot be dropped as long as the object that depends on it exists.
- C** Cascade semantics. Dropping the underlying object causes the object that depends on it (the depending object) to be dropped as well. However, if the depending object cannot be dropped because it has a Restrict dependency on some other object, the drop of the underlying object will fail.
- X** Inoperative semantics. Dropping the underlying object causes the object that depends on it to become inoperative. It remains inoperative until a user takes some explicit action.
- A** Automatic Invalidation/Revalidation semantics. Dropping the underlying object causes the object that depends on it to become invalid. The database manager attempts to revalidate the invalid object.

A package used by a function or a method, or by a procedure that is called directly or indirectly from a function or method, will only be automatically revalidated if the routine is defined as MODIFIES SQL DATA. If the routine is not MODIFIES SQL DATA, an error is returned (SQLSTATE 56098).

Some of the dependencies shown in Table 15 on page 265 change to "A" (Automatic Invalidation/Revalidation semantics) when the database configuration parameter **auto_reval** is set to IMMEDIATE or DEFERRED. Table 16 on page 272 summarizes the dependent objects that are impacted. Objects listed in the "Impacted Dependent Objects" column will be invalidated when the corresponding statement listed in the "Statement" column is executed.

In general, the database manager attempts to revalidate the invalid objects the next time the object is used. However, in situations when **auto_reval** is

set to IMMEDIATE, the impacted dependent objects will be revalidated immediately after they become invalid. Those situations are:

- ALTER TABLE ... ALTER COLUMN
- ALTER TABLE ... DROP COLUMN
- ALTER TABLE ... RENAME COLUMN
- ALTER TYPE ... ADD ATTRIBUTE
- ALTER TYPE ... DROP ATTRIBUTE
- Any CREATE statement that specifies "OR REPLACE"

Some DROP statement parameters and objects are not shown in Table 15 because they would result in blank rows or columns:

- EVENT MONITOR, PACKAGE, PROCEDURE, SCHEMA, TYPE MAPPING, and USER MAPPING DROP statements do not have object dependencies.
- Alias, buffer pool, distribution key, privilege, and procedure object types do not have DROP statement dependencies.
- A DROP SERVER, DROP FUNCTION MAPPING, or DROP TYPE MAPPING statement in a given unit of work (UOW) cannot be processed under either of the following conditions:
 - The statement references a single data source, and the UOW already includes a SELECT statement that references a nickname for a table or view within this data source (SQLSTATE 55006).
 - The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes a SELECT statement that references a nickname for a table or view within one of these data sources (SQLSTATE 55006).

Table 15. Dependencies

	Object Type																													
	C	O	N	F	S	T	R	A	N	I	B	D	O	M	U	G	E	S	L	C	L	E	P	N	N	E	O	E	A	C
ALTER FUNCTION	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER NICKNAME, altering the local name or the local type	R ³³	R	-	-	-	-	-	R	-	-	A	-	-	R	-	-	-	-	-	-	-	-	-	-	R	-	-	-	-	

Table 15. Dependencies (continued)

Statement	Object Type																																				
	C	O	N	F	S	T	R	A	I	N	O	N	G	E	X	N	D	E	P	E ³¹	R	S	E	S	L	C	L	E	P	N	G	W	N	T	D	T	
ALTER NICKNAME, altering a column option or a nickname option	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	A	-	-	R	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ALTER NICKNAME, adding, altering, or dropping a constraint	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		
ALTER PROCEDURE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		
ALTER SERVER	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		
ALTER TABLE ALTER COLUMN	-	A	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	A	-	-	-	A	-	-	-	-	-	X ³⁴		
ALTER TABLE DROP COLUMN	C	C	-	C	C	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	C	-	-	-	C	-	-	-	-	X ³⁴		
ALTER TABLE DROP CONSTRAINT	C	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	A ¹	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		
ALTER TABLE DROP PARTITIONING KEY	-	-	-	-	-	-	-	-	-	R ²⁰	A ¹	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ALTER TYPE ADD ATTRIBUTE	-	-	-	-	-	R	-	-	-	A ²³	-	-	R ²⁴	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	R ¹⁴	-	-	-	-	-	-		
ALTER TYPE ALTER METHOD	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ALTER TYPE DROP ATTRIBUTE	-	-	-	-	-	R	-	-	-	A ²³	-	-	R ²⁴	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	R ¹⁴	-	-	-	-	-	-	-	
ALTER TYPE ADD METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ALTER TYPE DROP METHOD	-	-	-	-	-	-	R ²⁷	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CREATE METHOD	-	-	-	-	-	-	-	-	-	A ²⁸	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CREATE TYPE	-	-	-	-	-	-	-	-	-	A ²⁹	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 15. Dependencies (continued)

	Object Type																																	
	C	O	N	F	S	T	R	A	I	N	O	M	U	G	E	S	L	C	L	E	P	N	N	E	O	E	A	C	T					
DROP VIEW HIERARCHY	-	R	-	R	-	-	-	-	-	-	-	-	-	A ²	-	-	-	-	-	-	X ¹⁶	-	-	-	X ¹⁶	-	-	-	-					
DROP WORK CLASS SET	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	R ³⁶	-	-					
DROP WRAPPER	-	-	C	-	-	-	-	-	-	-	-	-	-	C	-	-	-	-	-	-	-	-	C	-	-	-	-	-	-	-				
DROP XSROBJECT	C	-	-	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	X	-	-	-	X	-	-	-	-					
REVOKE a privilege ¹⁰	-	CR ²⁵	-	-	-	-	-	-	CR ²⁵	-	-	-	-	A ¹	-	-	-	-	-	-	CX ⁸	-	-	-	X	-	-	-	-	X ⁸	-	-	-	-

- 1 This dependency is implicit in depending on a table with these constraints, triggers, or a distribution key.
- 2 If a package has an INSERT, UPDATE, or DELETE statement acting upon a view, then the package has an insert, update or delete usage on the underlying base table of the view. In the case of UPDATE, the package has an update usage on each column of the underlying base table that is modified by the UPDATE.

If a package has a statement acting on a typed view, creating or dropping any view in the same view hierarchy will invalidate the package.
- 3 If a package, materialized query table, staging table, view, or trigger uses an alias, it becomes dependent both on the alias and the object that the alias references. If the alias is in a chain, a dependency is created on each alias in the chain.

Aliases themselves are not dependent on anything. It is possible for an alias to be defined on an object that does not exist.
- 4 A user-defined type T can depend on another user-defined type B, if T:
 - names B as the data type of an attribute
 - has an attribute of REF(B)
 - has B as a supertype.
- 5 Dropping a user-defined type cascades to dropping the functions and methods that use the type as a parameter, a result type, or in the function or method body. If the user-defined type is a structured type, any methods that are associated with the type are also dropped. Dropping these

functions and methods will not be prevented by the fact that the type and function or method depend on each other.

6 Dropping a table space or a list of table spaces causes all the tables that are completely contained within the given table space or list to be dropped. However, if a table spans table spaces (indexes, long columns, or data partitions in different table spaces) and those table spaces are not in the list being dropped, the table spaces cannot be dropped as long as the table exists.

7 A function can depend on another specific function if the depending function names the base function in a *SOURCE* clause. A function or method can also depend on another specific function or method if the depending routine is written in SQL and uses the base routine in its body. An external method, or an external function with a structured type parameter or returns type will also depend on one or more transform functions.

8 Only loss of *SELECT* privilege will cause a materialized query table to be dropped or a view to become inoperative. If the view that is made inoperative is included in a typed view hierarchy, all of its subviews also become inoperative.

9 If a package has an *INSERT*, *UPDATE*, or *DELETE* statement acting on table *T*, then the package has an insert, update or delete usage on *T*. In the case of *UPDATE*, the package has an update usage on each column of *T* that is modified by the *UPDATE*.

If a package has a statement acting on a typed table, creating or dropping any table in the same table hierarchy will invalidate the package.

10 Dependencies do not exist at the column level because privileges on columns cannot be revoked individually.

If a package, trigger or view includes the use of *OUTER(Z)* in the *FROM* clause, there is a dependency on the *SELECT* privilege on every subtable or subview of *Z*. Similarly, if a package, trigger, or view includes the use of *DEREF(Y)* where *Y* is a reference type with a target table or view *Z*, there is a dependency on the *SELECT* privilege on every subtable or subview of *Z*.

11 A materialized query table is dependent on the underlying tables or nicknames specified in the fullselect of the table definition.

Cascade semantics apply to dependent materialized query tables.

A subtable is dependent on its supertables up to the root table. A supertable cannot be dropped until all of its subtables are dropped.

12 A package can depend on structured types as a result of using the *TYPE* predicate or the subtype-treatment expression (*TREAT expression AS data-type*). The package has a dependency on the subtypes of each structured type specified in the right side of the *TYPE* predicate, or the right side of the *TREAT* expression. Dropping or creating a structured type that alters the subtypes on which the package is dependent causes invalidation.

All packages that are dependent on methods defined in supertypes of the type being dropped, and that are eligible for overriding, are invalidated.

13 A check constraint or trigger is dependent on a type if the type is used

anywhere in the constraint or trigger. There is no dependency on the subtypes of a structured type used in a TYPE predicate within a check constraint or trigger.

- 14 A view is dependent on a type if the type is used anywhere in the view definition (this includes the type of typed view). There is no dependency on the subtypes of a structured type used in a TYPE predicate within a view definition.
- 15 A subview is dependent on its superview up to the root view. A superview cannot be dropped until all its subviews are dropped. Refer to ¹⁶ for additional view dependencies.
- 16 A trigger or view is also dependent on the target table or target view of a dereference operation or Deref function. A trigger or view with a FROM clause that includes OUTER(Z) is dependent on all the subtables or subviews of Z that existed at the time the trigger or view was created.
- 17 A typed view can depend on the existence of a unique index to ensure the uniqueness of the object identifier column.
- 18 A table may depend on a user defined data type (distinct or structured) because the type is:
- used as the type of a column
 - used as the type of the table
 - used as an attribute of the type of the table
 - used as the target type of a reference type that is the type of a column of the table or an attribute of the type of the table
 - directly or indirectly used by a type that is the column of the table.
- 19 Dropping a server cascades to drop the function mappings and type mappings created for that named server.
- 20 If the distribution key is defined on a table in a multiple partition database partition group, the distribution key is required.
- 21 If a dependent OLE DB table function has "R" dependent objects (see DROP FUNCTION), then the server cannot be dropped.
- 22 An SQL function or method can depend on the objects referenced by its body.
- 23 When an attribute A of type TA of *type-name* T is dropped, the following DROP statements are effectively executed:
- ```
Mutator method: DROP METHOD A (TA) FOR T
Observer method: DROP METHOD A () FOR T
ALTER TYPE T
 DROP METHOD A(TA)
 DROP METHOD A()
```
- 24 A table may depend on an attribute of a user-defined structured data type in the following cases:
1. The table is a typed table that is based on *type-name* or any of its subtypes.
  2. The table has an existing column of a type that directly or indirectly refers to *type-name*.
- 25 A REVOKE of SELECT privilege on a table or view that is used in the body of an SQL function or method body causes an attempt to drop the function or method body, if the function or method body defined no longer has the SELECT privilege. If such a function or method body is used in a

view, trigger, function, or method body, it cannot be dropped, and the REVOKE is restricted as a result. Otherwise, the REVOKE cascades and drops such functions.

26 A trigger depends on an INSTEAD OF trigger when it modifies the view on which the INSTEAD OF trigger is defined, and the INSTEAD OF trigger fires.

27 A method declaration of an original method that is overridden by other methods cannot be dropped (SQLSTATE 42893).

28 If the method of the method body being created is declared to override another method, all packages dependent on the overridden method, and on methods that override this method in supertypes of the method being created, are invalidated.

29 When a new subtype of an existing type is created, all packages dependent on methods that are defined in supertypes of the type being created, and that are eligible for overriding (for example, no mutators or observers), are invalidated.

30 If the specific method of the method body being dropped is declared to override another method, all packages dependent on the overridden method, and on methods that override this method in supertypes of the specific method being dropped, are invalidated.

31 Cached dynamic SQL has the same semantics as packages.

32 When a remote base table is dropped using the DROP TABLE statement, both the nickname and the remote base table are dropped.

33 A primary key or unique keys that are not referenced by a foreign key do not restrict the altering of a nickname local name or local type.

34 An XSROBJECT can become inoperative for decomposition as a result of changes to a table that is associated with the XML schema for decomposition. Changes that could impact decomposition are: dropping the table or dropping a column of the table, or changing a column of the table. The decomposition status of the XML schema can be reset by issuing an ALTER XSROBJECT statement to enable or disable decomposition for the XML schema.

35

- A service class cannot be dropped if any threshold is mapped to it (SQLSTATE 5U031).
- A service class cannot be dropped if any workload is mapped to it (SQLSTATE 5U031).
- A service superclass cannot be dropped until all of its user-defined service subclasses have been dropped (SQLSTATE 5U031).
- A service superclass cannot be dropped if any work action set is mapped to it (SQLSTATE 5U031).
- A service subclass cannot be dropped if any work action is mapped to it (SQLSTATE 5U031).

36 A work class set cannot be dropped until the work action set that is defined on it has been dropped.

Table 16. Dependent Objects Impacted by **auto\_reval**

| Statement                                                  | Impacted Dependent Objects                                                                                         |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| ALTER NICKNAME (altering the local name or the local type) | Anchor Type, Function, Method, Procedure, User Defined Type, Variable, View                                        |
| ALTER TABLE ALTER COLUMN                                   | Anchor Type, Function, Method, Procedure, Trigger, User Defined Type, Variable, View, XSROBJECT                    |
| ALTER TABLE DROP COLUMN <sup>2</sup>                       | Anchor Type, Function, Method, Index, Procedure, Trigger, User Defined Type, Variable, View, XSROBJECT             |
| ALTER TABLE RENAME COLUMN <sup>1, 3</sup>                  | Anchor Type, Function, Method, Index, Procedure, Trigger, User Defined Type, Variable, View, XSROBJECT             |
| ALTER TYPE ADD ATTRIBUTE                                   | View                                                                                                               |
| ALTER TYPE DROP ATTRIBUTE                                  | View                                                                                                               |
| DROP ALIAS                                                 | Anchor Type, Function, Method, Procedure, Trigger, User Defined Type, Variable, View                               |
| DROP FUNCTION (ALTER MODULE DROP FUNCTION)                 | Function, Function Mapping, Index Extension, Method, Procedure, Trigger, Variable, View                            |
| DROP METHOD                                                | Function, Function Mapping, Index Extension, Method, Procedure, Trigger, View, Variable                            |
| DROP NICKNAME                                              | Anchor Type, Function, Method, Procedure, Trigger, User Defined Type, Variable, View                               |
| DROP PROCEDURE (ALTER MODULE DROP PROCEDURE)               | Function, Method, Procedure, Trigger                                                                               |
| DROP SEQUENCE                                              | Function, Method, Procedure, Trigger, Variable, View                                                               |
| DROP TABLE                                                 | Anchor Type, Function, Method, Procedure, Trigger, User Defined Type, Variable, View, XSROBJECT                    |
| DROP TABLE HIERARCHY                                       | Function, Method, Procedure, Trigger, Variable, View                                                               |
| DROP TRIGGER                                               | Trigger                                                                                                            |
| DROP TYPE (ALTER MODULE DROP TYPE)                         | Anchor Type, Cursor Type, Function, Method, Procedure, Index Extension, Trigger, User Defined Type, Variable, View |
| DROP VARIABLE (ALTER MODULE DROP VARIABLE)                 | Anchor Type, Function, Function Mapping, Method, Procedure, Trigger, User Defined Type, Variable, View             |
| DROP VIEW                                                  | Anchor Type, Function, Method, Procedure, Trigger, User Defined Type, Variable, View                               |
| DROP VIEW HIERARCHY                                        | Function, Procedure, Trigger, Variable, View                                                                       |
| DROP XSROBJECT                                             | Trigger, View                                                                                                      |
| RENAME TABLE                                               | Anchor Type, Function, Method, Procedure, Trigger, User Defined Type, View, Variable, XSROBJECT                    |
| REVOKE a privilege                                         | Function, Method, Procedure, Trigger, Variable, View                                                               |
| CREATE OR REPLACE ALIAS <sup>1</sup>                       | Function, Trigger, Procedure, Variable, View                                                                       |
| CREATE OR REPLACE VIEW <sup>1</sup>                        | Anchor Type, Function, Method, Procedure, Trigger, User Defined Type, Variable, View                               |
| CREATE OR REPLACE FUNCTION <sup>1</sup>                    | Function, Function Mapping, Index Extension, method, Procedure, Variable, View                                     |
| CREATE OR REPLACE PROCEDURE <sup>1</sup>                   | Function, Method, Procedure, Trigger                                                                               |
| CREATE OR REPLACE NICKNAME <sup>1</sup>                    | Function, method, Procedure, Variable, View                                                                        |
| CREATE OR REPLACE SEQUENCE <sup>1</sup>                    | Function, Method, Procedure, Trigger, Variable, View                                                               |

Table 16. Dependent Objects Impacted by **auto\_reval** (continued)

| Statement                               | Impacted Dependent Objects                                              |
|-----------------------------------------|-------------------------------------------------------------------------|
| CREATE OR REPLACE VARIABLE <sup>1</sup> | Function, Method, Procedure, Trigger, User Defined Type, Variable, View |
| CREATE OR REPLACE TRIGGER <sup>1</sup>  | Trigger                                                                 |

<sup>1</sup> REVALIDATION IMMEDIATE semantics apply for these statements (for the CREATE statements, only if OR REPLACE is specified) regardless of the setting of the **auto\_reval** database configuration parameter.

<sup>2</sup> The dependent objects listed will be revalidated the next time the object is used, except for the following objects, which will be revalidated immediately as part of the statement:

- ANCHOR TYPE
- CURSOR TYPE
- VIEW (where the select list consists only of SELECT \*, and does not contain any explicitly defined view columns).

For an immediate view revalidation, the list of column names for the select list will be re-established during revalidation.

<sup>3</sup> The dependent objects listed will be revalidated the next time the object is used except for the following, which will be revalidated immediately as part of the statement:

- User Defined Type
- VIEW (where the select list consists only of SELECT \*, and does not contain any explicitly defined view columns).

For an immediate view revalidation, the list of column names for the select list will be re-established during revalidation.

The DROP DATABASE PARTITION GROUP statement might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This statement might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

## Notes

- It is valid to drop a user-defined function while it is in use. Also, a cursor can be open over a statement which contains a reference to a user-defined function, and while this cursor is open the function can be dropped without causing the cursor fetches to fail.
- If a package which depends on a user-defined function is executing, it is not possible for another authorization ID to drop the function until the package completes its current unit of work. At that point, the function is dropped and the package becomes inoperative. The next request for this package results in an error indicating that the package must be explicitly rebound.
- The removal of a function body (this is very different from dropping the function) can occur while an application which needs the function body is executing. This may or may not cause the statement to fail, depending on whether the function body still needs to be loaded into storage by the database manager on behalf of the statement.
- In addition to the dependencies recorded for any explicitly specified UDF, the following dependencies are recorded when transforms are implicitly required:



1. When the structured type parameter or result of a function or method requires a transform, a dependency is recorded for the function or method on the required TO SQL or FROM SQL transform function.
2. When an SQL statement included in a package requires a transform function, a dependency is recorded for the package on the designated TO SQL or FROM SQL transform function.

Since the above describes the only circumstances under which dependencies are recorded due to implicit invocation of transforms, no objects other than functions, methods, or packages can have a dependency on implicitly invoked transform functions. On the other hand, explicit calls to transform functions (in views and triggers, for example) do result in the usual dependencies of these other types of objects on transform functions. As a result, a DROP TRANSFORM statement may also fail due to these "explicit" type dependencies of objects on the transform(s) being dropped (SQLSTATE 42893).

- Since the dependency catalogs do not distinguish between depending on a function as a transform versus depending on a function by explicit function call, it is suggested that explicit calls to transform functions are not written. In such an instance, the transform property on the function cannot be dropped, or packages will be marked inoperative, simply because they contain explicit invocations in an SQL expression.
- System created sequences for IDENTITY columns cannot be dropped using the DROP SEQUENCE statement.
- When a sequence is dropped, all privileges on the sequence are also dropped and any packages that refer to the sequence are invalidated.
- For relational nicknames, the DROP NICKNAME statement within a given unit of work (UOW) cannot be processed under either of the following conditions (SQLSTATE 55007):
  - A nickname referenced in this statement has a cursor open on it in the same UOW
  - Either an INSERT, DELETE, or UPDATE statement is already issued in the same UOW against the nickname that is referenced in this statement
- For non-relational nicknames, the DROP NICKNAME statement within a given unit of work (UOW) cannot be processed under any of the following conditions (SQLSTATE 55007):
  - A nickname referenced in this statement has a cursor open on it in the same UOW
  - A nickname referenced in this statement is already referenced by a SELECT statement in the same UOW
  - Either an INSERT, DELETE, or UPDATE statement has already been issued in the same UOW against the nickname that is referenced in this statement
- A DROP SERVER statement (SQLSTATE 55006), or a DROP FUNCTION MAPPING or DROP TYPE MAPPING statement (SQLSTATE 55007) within a given unit of work (UOW) cannot be processed under either of the following conditions:
  - The statement references a single data source, and the UOW already includes one of the following:
    - A SELECT statement that references a nickname for a table or view within this data source
    - An open cursor on a nickname for a table or view within this data source
    - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within this data source

- The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes one of the following:
  - A SELECT statement that references a nickname for a table or view within one of these data sources
  - An open cursor on a nickname for a table or view within one of these data sources
  - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within one of these data sources
- The DROP WORKLOAD statement does not take effect until it is committed, even for the connection that issues the statement.
- Only one of these statements can be issued by any application at a time, and only one of these statements is allowed within any one unit of work. Each statement must be followed by a COMMIT or a ROLLBACK statement before another one of these statements can be issued (SQLSTATE 5U021).
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (HISTOGRAM TEMPLATE)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)
  - CREATE WORK ACTION, ALTER WORK ACTION, or DROP (WORK ACTION)
  - CREATE WORK CLASS, ALTER WORK CLASS, or DROP (WORK CLASS)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
  - GRANT (Workload Privileges) or REVOKE (Workload Privileges)
- **Soft invalidation:** After the drop or change of a database object done by the following statements, active access to the dropped or changed object continues until the access is complete.
  - ALTER FUNCTION
  - ALTER MODULE ... DROP FUNCTION
  - ALTER MODULE ... DROP VARIABLE
  - ALTER TABLE ... DETACH PARTITION
  - ALTER VIEW
  - DROP ALIAS
  - DROP FUNCTION
  - DROP TRIGGER
  - DROP VARIABLE
  - DROP VIEW
  - All of the CREATE OR REPLACE statements except CREATE OR REPLACE SEQUENCE.

This is the case when the database registry variable `DB2_DLL_SOFT_INVALID` is set to ON. When it is set to OFF, the drop or change of these objects will only complete after all active access to the object to be dropped or changed is complete.

- **Compatibilities:**
  - For compatibility with previous versions of DB2 databases:
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

- DISTINCT TYPE *type-name* can be specified in place of TYPE *type-name*
- DATA TYPE *type-name* can be specified in place of TYPE *type-name*
- SYNONYM can be specified in place of ALIAS
- For compatibility with DB2 UDB for OS/390 and z/OS:
  - PROGRAM can be specified in place of PACKAGE

## Examples

*Example 1:* Drop table TDEPT.

```
DROP TABLE TDEPT
```

*Example 2:* Drop the view VDEPT.

```
DROP VIEW VDEPT
```

*Example 3:* The authorization ID HEDGES attempts to drop an alias.

```
DROP ALIAS A1
```

The alias HEDGES.A1 is removed from the catalogs.

*Example 4:* Hedges attempts to drop an alias, but specifies T1 as the alias-name, where T1 is the name of an existing table (not the name of an alias).

```
DROP ALIAS T1
```

This statement fails (SQLSTATE 42809).

*Example 5:*

Drop the BUSINESS\_OPS database partition group. To drop the database partition group, the two table spaces (ACCOUNTING and PLANS) in the database partition group must first be dropped.

```
DROP TABLESPACE ACCOUNTING
DROP TABLESPACE PLANS
DROP DATABASE PARTITION GROUP BUSINESS_OPS
```

*Example 6:* Pellow wants to drop the CENTRE function, which he created in his PELLOW schema, using the signature to identify the function instance to be dropped.

```
DROP FUNCTION CENTRE (INT,FLOAT)
```

*Example 7:* McBride wants to drop the FOCUS92 function, which she created in the PELLOW schema, using the specific name to identify the function instance to be dropped.

```
DROP SPECIFIC FUNCTION PELLOW.FOCUS92
```

*Example 8:* Drop the function ATOMIC\_WEIGHT from the CHEM schema, where it is known that there is only one function with that name.

```
DROP FUNCTION CHEM.ATOMIC_WEIGHT
```

*Example 9:* Drop the trigger SALARY\_BONUS, which caused employees under a specified condition to receive a bonus to their salary.

```
DROP TRIGGER SALARY_BONUS
```

*Example 10:* Drop the distinct data type named shoesize, if it is not currently in use.

**DROP TYPE SHOESIZE**

*Example 11:* Drop the SMITHPAY event monitor.

**DROP EVENT MONITOR SMITHPAY**

*Example 12:* Drop the schema from Example 2 under CREATE SCHEMA using RESTRICT. Notice that the table called PART must be dropped first.

**DROP TABLE PART  
DROP SCHEMA INVENTORY RESTRICT**

*Example 13:* Macdonald wants to drop the DESTROY procedure, which he created in the EIGLER schema, using the specific name to identify the procedure instance to be dropped.

**DROP SPECIFIC PROCEDURE EIGLER.DESTROY**

*Example 14:* Drop the procedure OSMOSIS from the BIOLOGY schema, where it is known that there is only one procedure with that name.

**DROP PROCEDURE BIOLOGY.OSMOSIS**

*Example 15:* User SHAWN used one authorization ID to access the federated database and another to access the database at an Oracle data source called ORACLE1. A mapping was created between the two authorizations, but SHAWN no longer needs to access the data source. Drop the mapping.

**DROP USER MAPPING FOR SHAWN SERVER ORACLE1**

*Example 16:* An index of a data source table that a nickname references has been deleted. Drop the index specification that was created to let the optimizer know about this index.

**DROP INDEX INDEXSPEC**

*Example 17:* Drop the MYSTRUCT1 transform group.

**DROP TRANSFORM MYSTRUCT1 FOR POLYGON**

*Example 18:* Drop the method BONUS for the EMP data type in the PERSONNEL schema.

**DROP METHOD BONUS (SALARY DECIMAL(10,2)) FOR PERSONNEL.EMP**

*Example 19:* Drop the sequence ORG\_SEQ, with restrictions.

**DROP SEQUENCE ORG\_SEQ**

*Example 20:* A remote table EMPLOYEE was created in a federated system using transparent DDL. Access to the table is no longer needed. Drop the remote table EMPLOYEE.

**DROP TABLE EMPLOYEE**

*Example 21:* Drop the function mapping BONUS\_CALC and reinstate the default function mapping (if one exists).

**DROP FUNCTION MAPPING BONUS\_CALC**

*Example 22:* Drop the security label component LEVEL.

**DROP SECURITY LABEL COMPONENT LEVEL**

*Example 23:* Drop the security label EMPLOYEESECLABEL of the security policy DATA\_ACCESS.

```
DROP SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABEL
```

*Example 24:* Drop the security policy DATA\_ACCESS.

```
DROP SECURITY POLICY DATA_ACCESS
```

*Example 25:* Drop the security label component GROUPS.

```
DROP SECURITY LABEL COMPONENT GROUPS
```

*Example 26:* Drop the XML schema EMPLOYEE located in the SQL schema HR.

```
DROP XSROBJECT HR.EMPLOYEE
```

*Example 27:* Drop service subclass DOGSALES under service superclass PETSALLES.

```
DROP SERVICE CLASS DOGSALES UNDER PETSALLES
```

*Example 28:* Drop service superclass PETSALLES, which has no user-defined service subclasses. The default subclass for service class PETSALLES is automatically dropped.

```
DROP SERVICE CLASS PETSALLES
```

---

## GRANT (Database Authorities)

This form of the GRANT statement grants authorities that apply to the entire database (rather than privileges that apply to specific objects within the database).

### Invocation

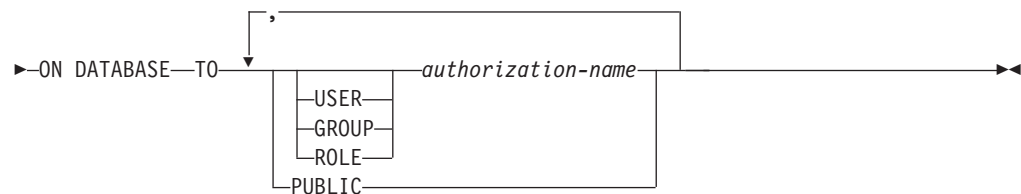
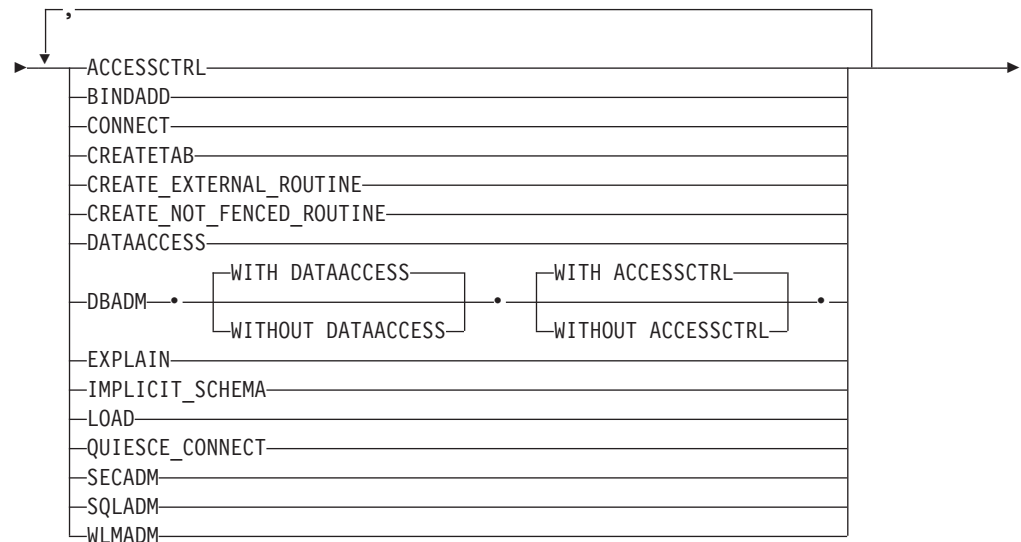
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

To grant ACCESSCTRL, DATAACCESS, DBADM, or SEADM authority, SECADM authority is required. To grant other authorities ACCESSCTRL or SECADM authority is required.

### Syntax

▶▶ GRANT →



## Description

### ACCESSCTRL

Grants the access control authority. The ACCESSCTRL authority allows the holder to:

- Grant and revoke the following database authorities: BINDADD, CONNECT, CREATETAB, CREATE\_EXTERNAL\_ROUTINE, CREATE\_NOT\_FENCED\_ROUTINE, EXPLAIN, IMPLICIT\_SCHEMA, LOAD, QUIESE\_CONNECT, SQLADM, WLMADM
- Grant and revoke all object level privileges

The ACCESSCTRL authority cannot be granted to PUBLIC (SQLSTATE 42508).

### BINDADD

Grants the authority to create packages. The creator of a package automatically has the CONTROL privilege on that package and retains this privilege even if the BINDADD authority is subsequently revoked.

### CONNECT

Grants the authority to access the database.

### CREATETAB

Grants the authority to create base tables. The creator of a base table automatically has the CONTROL privilege on that table. The creator retains this privilege even if the CREATETAB authority is subsequently revoked.

There is no explicit authority required for view creation. A view can be created at any time if the authorization ID of the statement used to create the view has either CONTROL or SELECT privilege on each base table of the view.

### CREATE\_EXTERNAL\_ROUTINE

Grants the authority to register external routines. Care must be taken that

routines so registered will not have adverse side effects. (For more information, see the description of the THREADSAFE clause on the CREATE or ALTER routine statements.)

Once an external routine has been registered, it continues to exist, even if CREATE\_EXTERNAL\_ROUTINE is subsequently revoked.

#### **CREATE\_NOT\_FENCED\_ROUTINE**

Grants the authority to register routines that execute in the database manager's process. Care must be taken that routines so registered will not have adverse side effects. (For more information, see the description of the FENCED clause on the CREATE or ALTER routine statements.)

Once a routine has been registered as not fenced, it continues to run in this manner, even if CREATE\_NOT\_FENCED\_ROUTINE is subsequently revoked.

CREATE\_EXTERNAL\_ROUTINE is automatically granted to an *authorization-name* that is granted CREATE\_NOT\_FENCED\_ROUTINE authority.

#### **DATAACCESS**

Grants the authority to access data. The DATAACCESS authority allows the holder to:

- Select, insert, update, delete, and load data
- Execute any package
- Execute any routine (except audit routines)

The DATAACCESS authority cannot be granted to PUBLIC (SQLSTATE 42508).

#### **DBADM**

Grants the database administrator authority. A database administrator holds nearly all privileges on nearly all objects in the database. The only exceptions are those privileges that are part of the access control, data access, and security administrator authorities.

#### **EXPLAIN**

Grants the authority to explain statements. The EXPLAIN authority allows the holder to explain, prepare, and describe dynamic and static SQL statements without requiring access to data.

#### **IMPLICIT\_SCHEMA**

Grants the authority to implicitly create a schema.

#### **LOAD**

Grants the authority to load in this database. This authority gives a user the right to use the LOAD utility in this database. DATAACCESS and DBADM also have this authority by default. However, if a user only has LOAD authority (not DATAACCESS), the user is also required to have table-level privileges. In addition to LOAD privilege, the user is required to have:

- INSERT privilege on the table for LOAD with mode INSERT, TERMINATE (to terminate a previous LOAD INSERT), or RESTART (to restart a previous LOAD INSERT)
- INSERT and DELETE privilege on the table for LOAD with mode REPLACE, TERMINATE (to terminate a previous LOAD REPLACE), or RESTART (to restart a previous LOAD REPLACE)
- INSERT privilege on the exception table, if such a table is used as part of LOAD

#### **QUIESCE\_CONNECT**

Grants the authority to access the database while it is quiesced.



## SECADM

Grants the security administrator authority. The authority allows the holder to:

- Create and drop security objects such as audit policies, roles, security labels, security label components, security policies, and trusted contexts
- Grant and revoke authorities, exemptions, privileges, roles, and security labels
- Grant and revoke the SETSESSIONUSER privilege
- Execute TRANSFER OWNERSHIP on objects owned by others

The SECADM authority cannot be granted to PUBLIC (SQLSTATE 42508).

## SQLADM

Grants the authority to manage SQL statement execution. The SQLADM authority allows the holder to:

- Create, drop, flush, and set event monitors
- Explain, prepare, and describe dynamic and static SQL statements without requiring access to data
- Flush optimization profile cache
- Flush package cache
- Execute the runstats utility

## WLMADM

Grants the authority to manage workloads. The WLMADM authority allows the holder to:

- Create, drop, and alter service classes, work action sets, work class sets, or workloads.

## TO

Specifies to whom the authorities are granted.

### USER

Specifies that the *authorization-name* identifies a user.

### GROUP

Specifies that the *authorization-name* identifies a group name.

### ROLE

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

### PUBLIC

Grants the authorities to a set of users (authorization IDs). For more information, see “Authorization, privileges and object ownership”. DBADM cannot be granted to PUBLIC.

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).

- If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
- If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
- If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
- If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
- If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

## Notes

- DBADM authority cannot be granted to the special group PUBLIC. Therefore, granting DBADM authority to a role *role-name* fails if *role-name* is granted to PUBLIC either directly or indirectly (SQLSTATE 42508).

- Role *role-name* is granted directly to PUBLIC if the following statement has been issued:

```
GRANT ROLE role-name TO PUBLIC
```

- Role *role-name* is granted indirectly to PUBLIC if the following statements have been issued:

```
GRANT ROLE role-name TO ROLE role-name2
GRANT ROLE role-name2 TO PUBLIC
```

- *Compatibilities:*

- For compatibility with previous versions of DB2:
  - CREATE\_NOT\_FENCED can be specified in place of CREATE\_NOT\_FENCED\_ROUTINE
- For compatibility with DB2 for z/OS:
  - SYSTEM can be specified in place of DATABASE

## Examples

*Example 1:* Give the users WINKEN, BLINKEN, and NOD the authority to connect to the database.

```
GRANT CONNECT ON DATABASE TO USER WINKEN, USER BLINKEN, USER NOD
```

*Example 2:* Grant BINDADD authority on the database to a group named D024. There is both a group and a user called D024 in the system.

```
GRANT BINDADD ON DATABASE TO GROUP D024
```

Observe that, the GROUP keyword must be specified; otherwise, an error will occur since both a user and a group named D024 exist. Any member of the D024 group will be allowed to bind packages in the database, but the D024 user will not be allowed (unless this user is also a member of the group D024, had been granted BINDADD authority previously, or BINDADD authority had been granted to another group of which D024 was a member).

*Example 3:* Give user Walid security administrator authority.

```
GRANT SECADM ON DATABASE TO USER Walid
```

## GRANT (Exemption)

This form of the GRANT statement grants to a user, group, or role an exemption on an access rule for a specified label-based access control (LBAC) security policy. When the user holding the exemption accesses data in a table protected by that security policy the indicated rule will not be enforced when deciding if they can access the data.

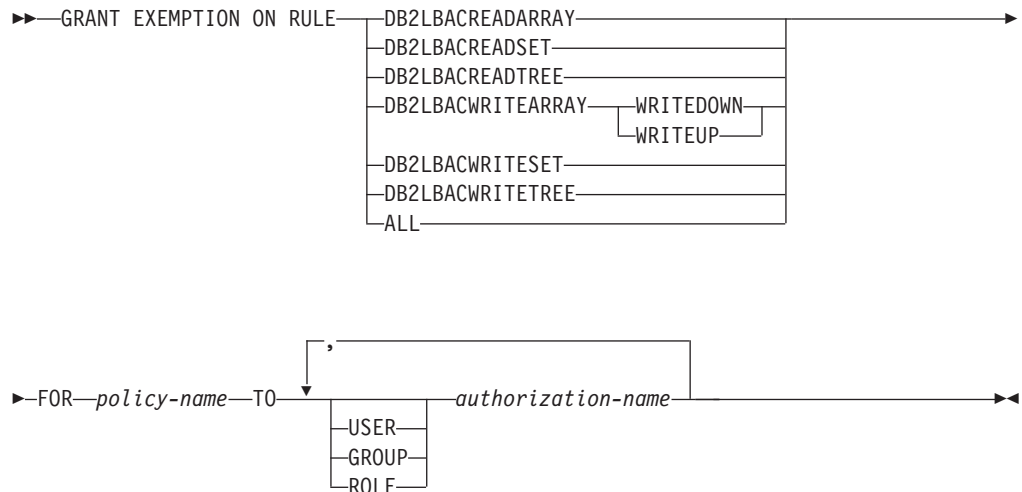
### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

### Syntax



### Description

#### EXEMPTION ON RULE

Grants an exemption on an access rule.

#### DB2LBACREADARRAY

Grants an exemption on the predefined DB2LBACREADARRAY rule.

#### DB2LBACREADSET

Grants an exemption on the predefined DB2LBACREADSET rule.

#### DB2LBACREADTREE

Grants an exemption on the predefined DB2LBACREADTREE rule.

#### DB2LBACWRITEARRAY

Grants an exemption on the predefined DB2LBACWRITEARRAY rule.

#### WRITEDOWN

Specifies that the exemption only applies to write down.

**WRITEUP**

Specifies that the exemption only applies to write up.

**DB2LBACWRITESSET**

Grants an exemption on the predefined DB2LBACWRITESSET rule.

**DB2LBACWRITETREE**

Grants an exemption on the predefined DB2LBACWRITETREE rule.

**ALL**

Grants an exemption on all of the predefined rules.

**FOR** *policy-name*

Identifies the security policy for which the exemption is being granted. The exemption will only be effective for tables that are protected by this security policy. The name must identify a security policy already described in the catalog (SQLSTATE 42704).

**TO**

Specifies to whom the exemption is granted.

**USER**

Specifies that the *authorization-name* identifies a user.

**GROUP**

Specifies that the *authorization-name* identifies a group name.

**ROLE**

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

**Rules**

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- If the security policy is not defined to consider access through groups or roles, any exemption granted to a group or role is ignored when access is attempted.

**Notes**

- By default when a security policy is created, only exemptions granted to an individual user are considered. To have groups or roles considered for the



**ON INDEX** *index-name*

Identifies the index for which the CONTROL privilege is to be granted.

**TO**

Specifies to whom the privileges are granted.

**USER**

Specifies that the *authorization-name* identifies a user.

**GROUP**

Specifies that the *authorization-name* identifies a group name.

**ROLE**

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

**PUBLIC**

Grants the privileges to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

**Rules**

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

**Example**

```
GRANT CONTROL ON INDEX DEPTIDX TO USER KIESLER
```

---

## GRANT (Package Privileges)

This form of the GRANT statement grants privileges on a package.

**Invocation**

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

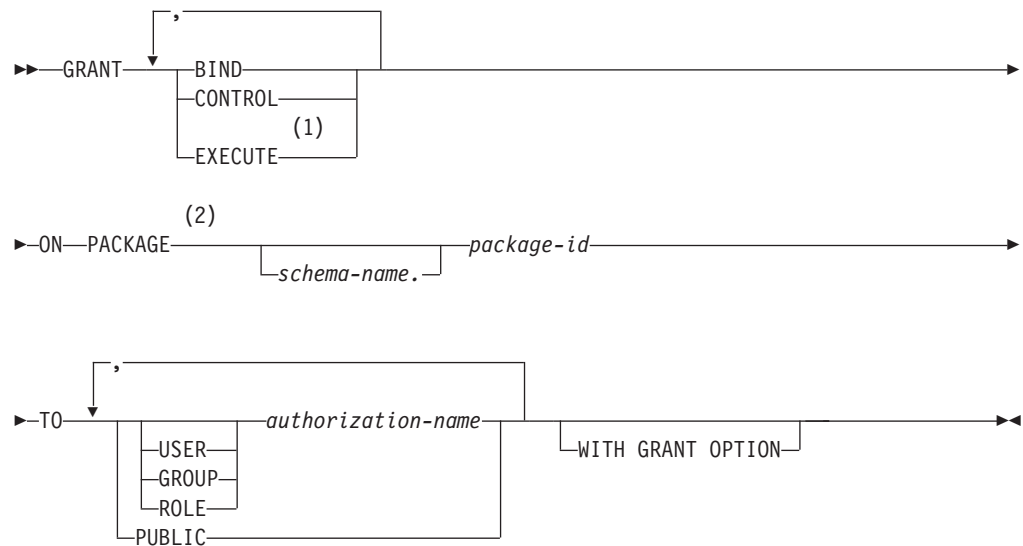
## Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced package
- The WITH GRANT OPTION for each identified privilege on *package-name*
- ACCESSCTRL or SECADM authority

ACCESSCTRL or SECADM authority is required to grant the CONTROL privilege.

## Syntax



### Notes:

- 1 RUN can be used as a synonym for EXECUTE.
- 2 PROGRAM can be used as a synonym for PACKAGE.

## Description

### BIND

Grants the privilege to bind a package. The BIND privilege allows a user to re-issue the BIND command against that package, or to issue the REBIND command. It also allows a user to create a new version of an existing package.

In addition to the BIND privilege, a user must hold the necessary privileges on each table referenced by static DML statements contained in a program. This is necessary, because authorization on static DML statements is checked at bind time.

### CONTROL

Grants the privilege to rebind, drop, or execute the package, and extend package privileges to other users. The CONTROL privilege for packages is automatically granted to creators of packages. A package owner is the package binder, or the ID specified with the OWNER option at bind/precompile time.

BIND and EXECUTE are automatically granted to an *authorization-name* that is granted CONTROL privilege.



CONTROL grants the ability to grant the above privileges (except for CONTROL) to others.

#### **EXECUTE**

Grants the privilege to execute the package.

#### **ON PACKAGE** *schema-name.package-id*

Specifies the name of the package on which privileges are to be granted. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The granting of a package privilege applies to all versions of the package (that is, to all packages that share the same package ID and package schema).

#### **TO**

Specifies to whom the privileges are granted.

#### **USER**

Specifies that the *authorization-name* identifies a user.

#### **GROUP**

Specifies that the *authorization-name* identifies a group name.

#### **ROLE**

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

#### **PUBLIC**

Grants the privileges to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

#### **WITH GRANT OPTION**

Allows the specified *authorization-name* to GRANT the privileges to others.

If the specified privileges include CONTROL, the WITH GRANT OPTION applies to all of the applicable privileges except for CONTROL (SQLSTATE 01516).

### **Rules**

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

## Notes

- Package privileges apply to all versions of a package (that is, all packages that share the same package ID and package schema). It is not possible to restrict access to only one version. Because CONTROL privilege is implicitly granted to the binder of a package, if two different users bind two versions of a package, then both users will implicitly be granted access to each other's package.

## Examples

*Example 1:* Grant the EXECUTE privilege on PACKAGE CORPDATA.PKGA to PUBLIC.

```
GRANT EXECUTE
ON PACKAGE CORPDATA.PKGA
TO PUBLIC
```

*Example 2:* GRANT EXECUTE privilege on package CORPDATA.PKGA to a user named EMPLOYEE. There is neither a group nor a user called EMPLOYEE.

```
GRANT EXECUTE ON PACKAGE
CORPDATA.PKGA TO EMPLOYEE
```

or

```
GRANT EXECUTE ON PACKAGE
CORPDATA.PKGA TO USER EMPLOYEE
```

---

## GRANT (Role)

This form of the GRANT statement grants roles to users, groups, or to other roles.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

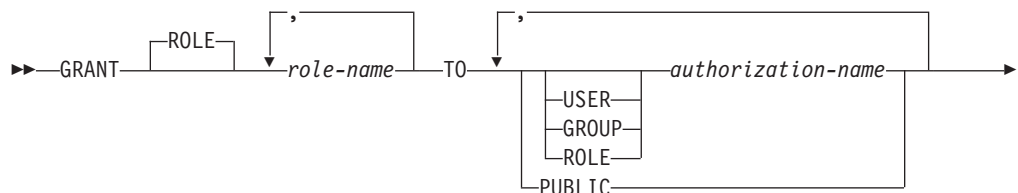
### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- The WITH ADMIN OPTION on the role
- SECADM authority

SECADM authority is required to grant the WITH ADMIN OPTION to an *authorization-name*.

### Syntax



## Description

### ROLE *role-name*,...

Identifies one or more roles to be granted. Each *role-name* must identify an existing role at the current server (SQLSTATE 42704).

### TO

Specifies to whom the role is granted.

### USER

Specifies that the *authorization-name* identifies a user.

### GROUP

Specifies that the *authorization-name* identifies a group.

### ROLE

Specifies that the *authorization-name* identifies an existing role at the current server (SQLSTATE 42704).

### *authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

### PUBLIC

Grants the specified roles to a set of users (authorization IDs). For more information, see “Authorization, privileges, and object ownership”.

### WITH ADMIN OPTION

Allows the specified *authorization-name* to grant or revoke the *role-name* to or from others, or to associate a comment with the role. It does not allow the specified *authorization-name* to drop the role.

## Rules

- For each *authorization-name* specified, if none of the keywords USER, GROUP, or ROLE is specified:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database and as either GROUP or USER in the operating system, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as both USER and GROUP according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as USER only according to the security plug-in in effect, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined as GROUP only according to the security plug-in in effect, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- Hierarchies of roles can be built by granting one role to another role. However, cycles are not allowed (SQLSTATE 428GF). For example, if role R1 is granted to

another role R2, then role R2 (or some other role R<sub>n</sub> that contains R2) cannot be granted back to R1, because this would produce a cycle.

## Notes

- When role R1 is granted to another role R2, then R2 contains R1.
- DBADM authority cannot be granted to PUBLIC. Therefore:
  - Granting role R1 to PUBLIC fails (SQLSTATE 42508) if role R1 holds DBADM authority either directly or indirectly.
    - Role R1 holds DBADM authority directly if the following statement has been issued:  
**GRANT DBADM ON DATABASE TO ROLE R1**
    - Role R1 holds DBADM authority indirectly if the following statements have been issued:  
**GRANT DBADM ON DATABASE TO ROLE R2**  
  
**GRANT ROLE R2 TO ROLE R1**
  - Granting role R1, which holds DBADM authority, to role R2 fails (SQLSTATE 42508) if role R2 is granted to PUBLIC either directly or indirectly.
    - Role R2 is granted to PUBLIC directly if the following statement has been issued:  
**GRANT ROLE R2 TO PUBLIC**
    - Role R2 is granted to PUBLIC indirectly if the following statements have been issued:  
**GRANT ROLE R2 TO ROLE R3**  
  
**GRANT ROLE R3 TO PUBLIC**

## Examples

*Example 1:* Grant role INTERN to role DOCTOR and role DOCTOR to role SPECIALIST.

```
GRANT ROLE INTERN TO ROLE DOCTOR
GRANT ROLE DOCTOR TO ROLE SPECIALIST
```

*Example 2:* Grant role INTERN to PUBLIC.

```
GRANT ROLE INTERN TO PUBLIC
```

*Example 3:* Grant role SPECIALIST to user BOB and group TORONTO.

```
GRANT ROLE SPECIALIST TO USER BOB, GROUP TORONTO
```

---

## GRANT (routine privileges)

This form of the GRANT statement grants privileges on a routine (function, method, or procedure) that is not defined in a module.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

## Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

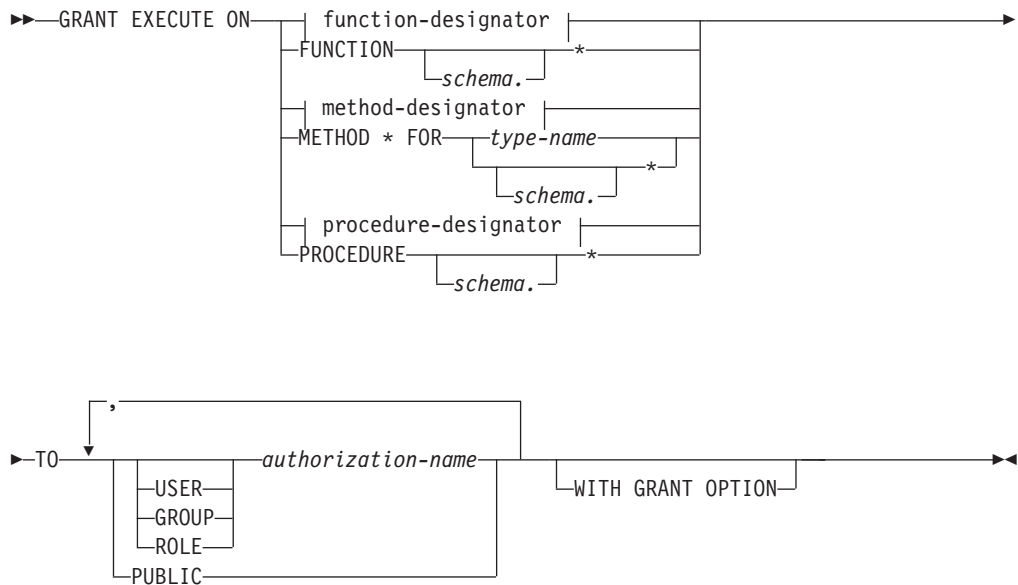
- The WITH GRANT OPTION for EXECUTE on the routine
- ACCESSCTRL or SECADM authority

To grant all routine EXECUTE privileges in the schema or type, the privileges held by the authorization ID of the statement must include at least one of the following:

- The WITH GRANT OPTION for EXECUTE on all existing and future routines (of the specified type) in the specified schema
- ACCESSCTRL or SECADM authority

To grant EXECUTE privilege on the audit procedures and table functions SECADM authority is required. EXECUTE privilege WITH GRANT OPTION cannot be granted for these routines (SQLSTATE 42501)

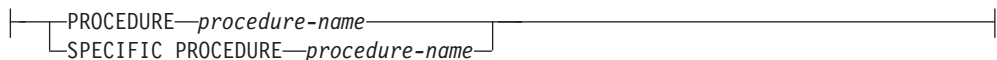
## Syntax



### function-designator:



### procedure-designator:



## Description

### EXECUTE

Grants the privilege to run the identified user-defined function, method, or procedure.

#### *function-designator*

Uniquely identifies the function on which the privilege is granted. For more information, see “Function, method, and procedure designators” on page 497.

### FUNCTION *schema.\**

Identifies all the functions in the schema, including any functions that may be created in the future. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

#### *method-designator*

Uniquely identifies the method on which the privilege is granted. For more information, see “Function, method, and procedure designators” on page 497.

### METHOD \*

Identifies all the methods for the type *type-name*, including any methods that may be created in the future.

#### FOR *type-name*

Names the type in which the specified method is found. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the value of the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names. An asterisk (\*) can be used in place of *type-name* to identify all types in the schema, including any types that may be created in the future.

#### *procedure-designator*

Uniquely identifies the procedure on which the privilege is granted. For more information, see “Function, method, and procedure designators” on page 497.

### PROCEDURE *schema.\**

Identifies all the procedures in the schema, including any procedures that may be created in the future. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

### TO

Specifies to whom the EXECUTE privilege is granted.

#### USER

Specifies that the *authorization-name* identifies a user.

#### GROUP

Specifies that the *authorization-name* identifies a group name.

#### ROLE

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

#### *authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

## PUBLIC

Grants the EXECUTE privilege to a set of users (authorization IDs). For more information, see “Authorization, privileges and object ownership”.

## WITH GRANT OPTION

Allows the specified *authorization-names* to GRANT the EXECUTE privilege to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only grant the EXECUTE privilege to others if they:

- have SYSADM or DBADM authority or
- received the ability to grant the EXECUTE privilege from some other source.

## Rules

- It is not possible to grant the EXECUTE privilege on a function or method defined with schema 'SYSIBM' or 'SYSFUN' (SQLSTATE 42832).
- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, and no privileges were granted, a warning is returned (SQLSTATE 01007). If the grantor has no privileges on the object of the grant operation, an error is returned (SQLSTATE 42501).

## Notes

- Privileges for a routine defined in a module are granted at the module level using the GRANT (module privileges) statement. The EXECUTE privilege on the module allows access to all objects in the module.

## Examples

*Example 1:* Grant the EXECUTE privilege on function CALC\_SALARY to user JONES. Assume that there is only one function in the schema with function name CALC\_SALARY.

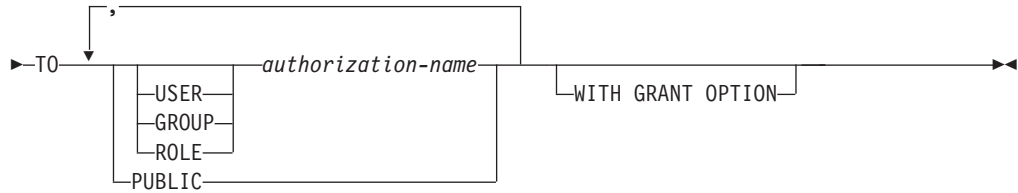
```
GRANT EXECUTE ON FUNCTION CALC_SALARY TO JONES
```

*Example 2:* Grant the EXECUTE privilege on procedure VACATION\_ACCR to all users at the current server.

```
GRANT EXECUTE ON PROCEDURE VACATION_ACCR TO PUBLIC
```







## Description

### ALTERIN

Grants the privilege to alter or comment on all objects in the schema. The owner of an explicitly created schema automatically receives ALTERIN privilege.

### CREATEIN

Grants the privilege to create objects in the schema. Other authorities or privileges required to create the object (such as CREATETAB) are still required. The owner of an explicitly created schema automatically receives CREATEIN privilege. An implicitly created schema has CREATEIN privilege automatically granted to PUBLIC.

### DROPIN

Grants the privilege to drop all objects in the schema. The owner of an explicitly created schema automatically receives DROPIN privilege.

### ON SCHEMA *schema-name*

Identifies the schema on which the privileges are to be granted.

### TO

Specifies to whom the privileges are granted.

#### USER

Specifies that the *authorization-name* identifies a user.

#### GROUP

Specifies that the *authorization-name* identifies a group name.

#### ROLE

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

#### *authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

#### PUBLIC

Grants the privileges to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

### WITH GRANT OPTION

Allows the specified *authorization-names* to GRANT the privileges to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-names* can only grant the privileges to others if they:

- have DBADM authority or
- received the ability to grant privileges from some other source.

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E for MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.)

## Notes

- **Grant on SYSPUBLIC:** Privileges can be granted on the reserved schema SYSPUBLIC. Granting CREATEIN privilege allows the user to create a public alias and granting DROPIN privilege allows the user to drop any public alias.

## Examples

*Example 1:* Grant user JSINGLETON to the ability to create objects in schema CORPDATA.

```
GRANT CREATEIN ON SCHEMA CORPDATA TO JSINGLETON
```

*Example 2:* Grant user IHAKES the ability to create and drop objects in schema CORPDATA.

```
GRANT CREATEIN, DROPIN ON SCHEMA CORPDATA TO IHAKES
```

---

## GRANT (Security Label)

This form of the GRANT statement grants a label-based access control (LBAC) security label to a user, group, or role for read access, write access, or for both read and write access.

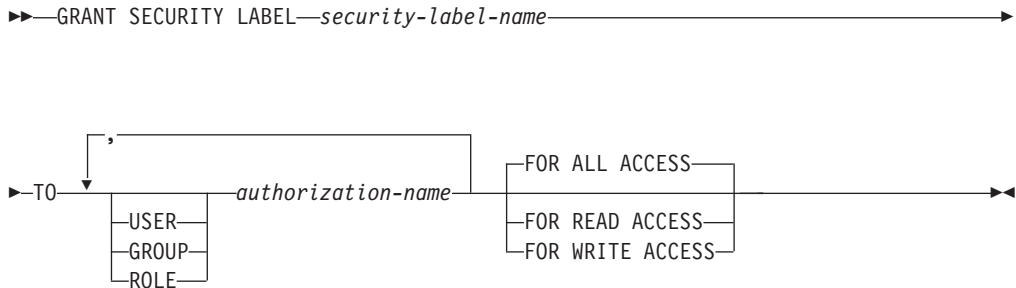
## Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

## Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

### Syntax



### Description

#### **SECURITY LABEL** *security-label-name*

Grants the security label *security-label-name*. The name must be qualified with a security policy (SQLSTATE 42704) and must identify a security label that exists at the current server (SQLSTATE 42704).

#### **TO**

Specifies to whom the specified security label is granted.

#### **USER**

Specifies that the *authorization-name* identifies a user.

#### **GROUP**

Specifies that the *authorization-name* identifies a group name.

#### **ROLE**

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

#### **FOR ALL ACCESS**

Indicates that the security label is to be granted for both read access and write access.

#### **FOR READ ACCESS**

Indicates that the security label is to be granted for read access only.

#### **FOR WRITE ACCESS**

Indicates that the security label is to be granted for write access only.

### Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).

- If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
- If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
- If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
- If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- For any given security policy, an *authorization-name* can be granted at most one security label from that policy for read access and one for write access. If the grantee already holds a security label for the type of access (read or write) indicated and that is part of the security policy that qualifies *security-label-name*, an error is returned (SQLSTATE 428GR).
- If the security policy is not defined to consider access through groups or roles, any security label granted to a group or role is ignored when access is attempted.
- If an *authorization-name* holds different security labels for read access and write access, the security labels must meet the following criteria (SQLSTATE 428GQ):
  - If any component in the security labels is of type ARRAY then the value for that component must be the same in both security labels.
  - If any component in the security labels is of type SET then every element in the value for that component in the write security label must also be part of the value for that component in the read security label.
  - If any component in the security labels is of type TREE then every element in the value for that component in the write security label must be the same as or a descendent of one of the elements in the value for that same component in the read security label.

## Notes

- By default when a security policy is created, only security labels granted to an individual user are considered. To have groups or roles considered for the security policy, you must issue the ALTER SECURITY POLICY statement and specify USE GROUP AUTHORIZATION or USE ROLE AUTHORIZATION as applicable.

## Examples

*Example 1:* The following statement grants two security labels to user GUYLAINE. The security label EMPLOYEESECLABELREAD is granted for read access and the security label EMPLOYEESECLABELWRITE is granted for write access. Both security labels are part of the security policy DATA\_ACCESS.

```
GRANT SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELREAD
 TO USER GUYLAINE FOR READ ACCESS
```

```
GRANT SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELWRITE
 TO USER GUYLAINE FOR WRITE ACCESS
```

The same user is now granted the security label BEGINNER for both read and write access. This does not cause an error, because BEGINNER is part of the security policy CLASSPOLICY, and the security labels already held are part of the security policy DATA\_ACCESS.

```
GRANT SECURITY LABEL CLASSPOLICY.BEGINNER
 TO USER GUYLAINE FOR ALL ACCESS
```

---

## GRANT (Sequence Privileges)

This form of the GRANT statement grants privileges on a sequence.

### Invocation

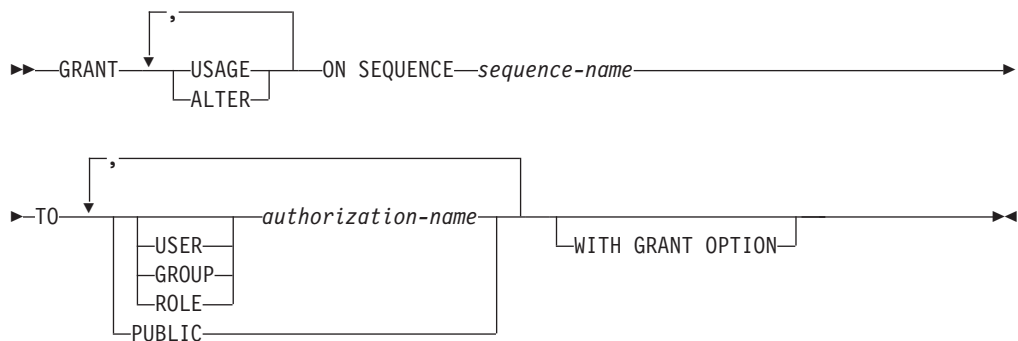
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- The WITH GRANT OPTION for each identified privilege on *sequence-name*
- ACCESSCTRL or SECADM authority

### Syntax



### Description

#### USAGE

Grants the privilege to reference a sequence using *nextval-expression* or *prevval-expression*.

#### ALTER

Grants the privilege to alter sequence properties using the ALTER SEQUENCE statement.

#### ON SEQUENCE *sequence-name*

Identifies the sequence on which the specified privileges are to be granted. The sequence name, including an implicit or explicit schema qualifier, must uniquely identify an existing sequence at the current server. If no sequence by this name exists, an error (SQLSTATE 42704) is returned.

#### TO

Specifies to whom the specified privileges are granted.

#### USER

Specifies that the *authorization-name* identifies a user.

#### GROUP

Specifies that the *authorization-name* identifies a group name.

## ROLE

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles.

## PUBLIC

Grants the specified privileges to a set of users (authorization IDs). For more information, see “Authorization, privileges and object ownership”.

## WITH GRANT OPTION

Allows the specified *authorization-name* to grant the specified privileges to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only grant the specified privileges to others if they:

- have SYSADM or DBADM authority or
- received the ability to grant the specified privileges from some other source.

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges is not granted. If no privileges are granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.)

## Example

*Example 1:* Grant any user the USAGE privilege on a sequence called ORG\_SEQ.

```
GRANT USAGE ON SEQUENCE ORG_SEQ TO PUBLIC
```

*Example 2:* Grant user BOBBY the ability to alter a sequence called GENERATE\_ID, and to grant this privilege to others.

```
GRANT ALTER ON SEQUENCE GENERATE_ID TO BOBBY WITH GRANT OPTION
```



---

## GRANT (Server Privileges)

This form of the GRANT statement grants the privilege to access and use a specified data source in pass-through mode.

### Invocation

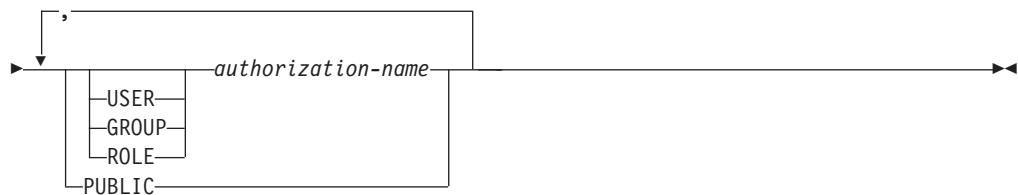
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

### Syntax

►► GRANT PASSTHRU ON SERVER—*server-name*—TO—



### Description

*server-name*

Names the data source for which the privilege to use in pass-through mode is being granted. *server-name* must identify a data source that is described in the catalog.

**TO**

Specifies to whom the privilege is granted.

**USER**

Specifies that the *authorization-name* identifies a user.

**GROUP**

Specifies that the *authorization-name* identifies a group name.

**ROLE**

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

## PUBLIC

Grants to a set of users (authorization IDs) the privilege to pass through to *server-name*. For more information, see “Authorization, privileges and object ownership”.

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

## Examples

*Example 1:* Give R. Smith and J. Jones the privilege to pass through to data source SERVALL. Their authorization IDs are RSMITH and JJONES.

```
GRANT PASSTHRU ON SERVER SERVALL
TO USER RSMITH,
USER JJONES
```

*Example 2:* Grant the privilege to pass through to data source EASTWING to a group whose authorization ID is D024. There is a user whose authorization ID is also D024.

```
GRANT PASSTHRU ON SERVER EASTWING TO GROUP D024
```

The GROUP keyword must be specified; otherwise, an error will occur because D024 is a user’s ID as well as the specified group’s ID (SQLSTATE 56092). Any member of group D024 will be allowed to pass through to EASTWING. Therefore, if user D024 belongs to the group, this user will be able to pass through to EASTWING.

---

## GRANT (SETSESSIONUSER Privilege)

This form of the GRANT statement grants the SETSESSIONUSER privilege to one or more authorization IDs. The privilege allows the holder to use the SET SESSION AUTHORIZATION statement to set the session authorization to one of a set of specified authorization IDs.

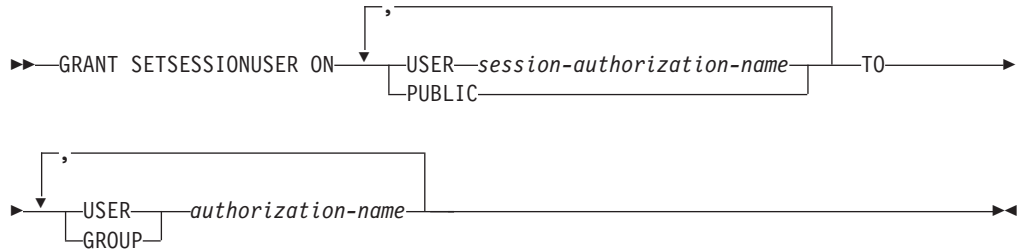
## Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

## Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

## Syntax



## Description

### SETSESSIONUSER ON

Grants the privilege to assume the identity of a new authorization ID.

### USER *session-authorization-name*

Specifies the authorization ID that the *authorization-name* will be able to assume, using the SET SESSION AUTHORIZATION statement. The *session-authorization-name* must identify a user, not a group.

### PUBLIC

Specifies that the grantee will be able to assume any valid authorization ID, using the SET SESSION AUTHORIZATION statement.

### TO

Specifies to whom the privilege is granted.

### USER

Specifies that the *authorization-name* identifies a user.

### GROUP

Specifies that the *authorization-name* identifies a group.

*authorization-name*,...

Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

## Rules

- For each *authorization-name* specified, if neither USER nor GROUP is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.

## Examples

*Example 1:* The following statement grants user PAUL the ability to set the session authorization to user WALID and therefore to execute statements as WALID.

```
GRANT SETSESSIONUSER ON USER WALID
 TO USER PAUL
```

*Example 2:* The following statement grants user GUYLAINE the ability to set the session authorization to user BOBBY. It also grants her the ability to set the session authorization to users RICK and KEVIN.

```
GRANT SETSESSIONUSER ON USER BOBBY, USER RICK, USER KEVIN
 TO USER GUYLAINE
```

*Example 3:* The following statement grants user WALID and everyone in the groups ADMINS and ACCTG the ability to set the session authorization to any user.

```
GRANT SETSESSIONUSER ON PUBLIC TO USER WALID, GROUP ADMINS, ACCTG
```

---

## GRANT (Table Space Privileges)

This form of the GRANT statement grants privileges on a table space.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

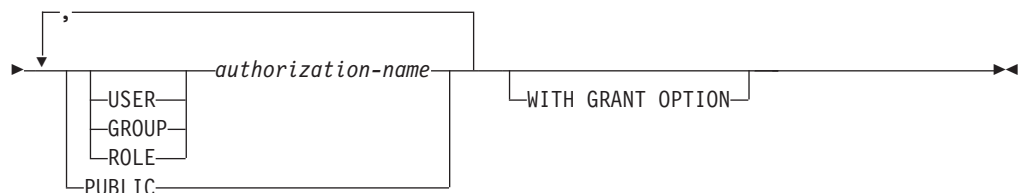
### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- The WITH GRANT OPTION for use of the table space
- ACCESSCTRL, SECADM, SYSADM, or SYSCTRL authority

### Syntax

```
►► GRANT USE OF TABLESPACE tablespace-name TO _____►
```



### Description

#### USE

Grants the privilege to specify or default to the table space when creating a table. The creator of a table space automatically receives USE privilege with grant option.

**OF TABLESPACE** *tablespace-name*

Identifies the table space on which the USE privilege is to be granted. The table space cannot be SYSCATSPACE (SQLSTATE 42838) or a system temporary table space (SQLSTATE 42809).

**TO**

Specifies to whom the USE privilege is granted.

**USER**

Specifies that the *authorization-name* identifies a user.

**GROUP**

Specifies that the *authorization-name* identifies a group name.

**ROLE**

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name*

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

**PUBLIC**

Grants the USE privilege to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

**WITH GRANT OPTION**

Allows the specified *authorization-name* to GRANT the USE privilege to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only GRANT the USE privilege to others if they:

- have SYSADM or DBADM authority or
- received the ability to GRANT the USE privilege from some other source.

**Rules**

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

**Examples**

*Example 1:* Grant user BOBBY the ability to create tables in table space PLANS and to grant this privilege to others.

**GRANT USE OF TABLESPACE PLANS TO BOBBY WITH GRANT OPTION**

## GRANT (Table, View, or Nickname Privileges)

This form of the GRANT statement grants privileges on a table, view, or nickname.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

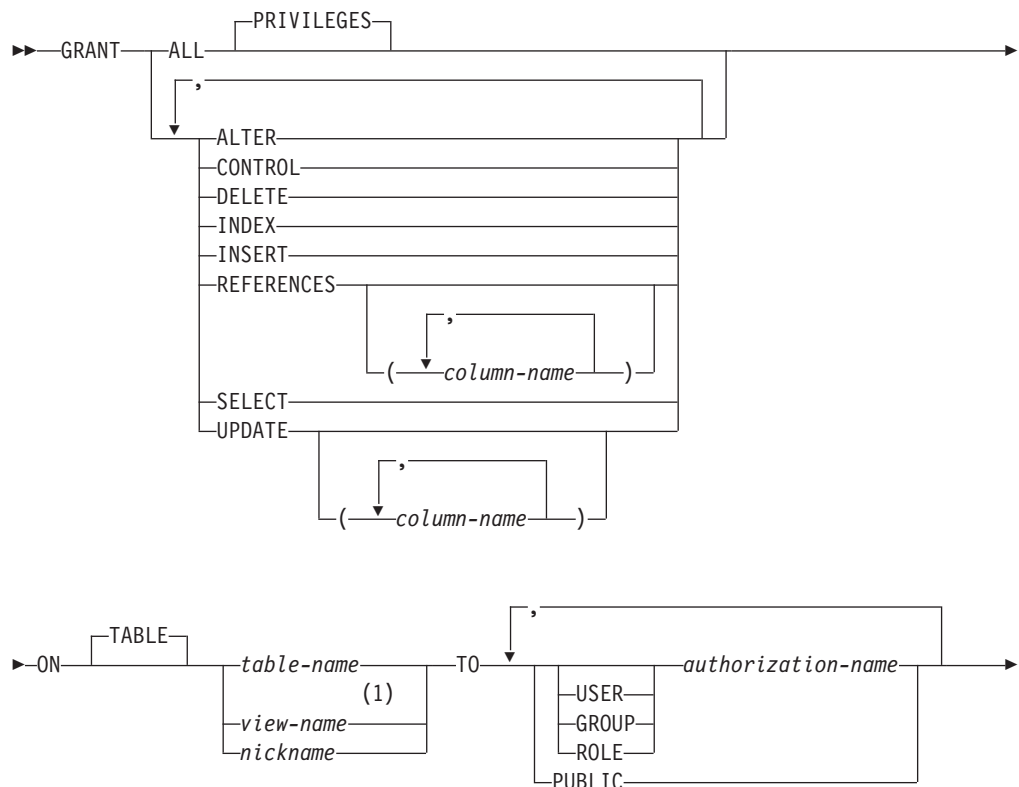
### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced table, view, or nickname
- The WITH GRANT OPTION for each identified privilege. If ALL is specified, the authorization ID must have some grantable privilege on the identified table, view, or nickname.
- ACCESSCTRL or SECADM authority

ACCESSCTRL or SECADM authority is required to grant the CONTROL privilege, or to grant privileges on catalog tables and views.

### Syntax



**Notes:**

- 1 ALTER, INDEX, and REFERENCES privileges are not applicable to views.

**Description****ALL or ALL PRIVILEGES**

Grants all the appropriate privileges, except CONTROL, on the base table, view, or nickname named in the ON clause.

If the authorization ID of the statement has CONTROL privilege on the table, view, or nickname, or ACCESSCTRL or SECADM authority, then all the privileges applicable to the object (except CONTROL) are granted. Otherwise, the privileges granted are all those grantable privileges that the authorization ID of the statement has on the identified table, view, or nickname.

If ALL is not specified, one or more of the keywords in the list of privileges must be specified.

**ALTER**

Grants the privilege to:

- Add columns to a base table definition.
- Create or drop a primary key or unique constraint on a base table.
- Create or drop a foreign key on a base table.

The REFERENCES privilege on each column of the parent table is also required.

- Create or drop a check constraint on a base table.
- Create a trigger on a base table.
- Add, reset, or drop a column option for a nickname.
- Change a nickname column name or data type.
- Add or change a comment on a base table or a nickname.

**CONTROL**

Grants:

- All of the appropriate privileges in the list, that is:
  - ALTER, CONTROL, DELETE, INSERT, INDEX, REFERENCES, SELECT, and UPDATE to base tables
  - CONTROL, DELETE, INSERT, SELECT, and UPDATE to views
  - ALTER, CONTROL, INDEX, and REFERENCES to nicknames
- The ability to grant the above privileges (except for CONTROL) to others.
- The ability to drop the base table, view, or nickname.

This ability cannot be extended to others on the basis of holding CONTROL privilege. The only way that it can be extended is by granting the CONTROL privilege itself and that can only be done by an authorization ID with ACCESSCTRL or SECADM authority.

- The ability to execute the RUNSTATS utility on the table and indexes.
- The ability to execute the REORG utility on the table.
- The ability to issue the SET INTEGRITY statement against a base table, materialized query table, or staging table.



The definer of a base table, materialized query table, staging table, or nickname automatically receives the CONTROL privilege.

The definer of a view automatically receives the CONTROL privilege if the definer holds the CONTROL privilege on all tables, views, and nicknames identified in the fullselect.

#### **DELETE**

Grants the privilege to delete rows from the table or updatable view.

#### **INDEX**

Grants the privilege to create an index on a table, or an index specification on a nickname. This privilege cannot be granted on a view. The creator of an index or index specification automatically has the CONTROL privilege on the index or index specification (authorizing the creator to drop the index or index specification). In addition, the creator retains the CONTROL privilege even if the INDEX privilege is revoked.

#### **INSERT**

Grants the privilege to insert rows into the table or updatable view and to run the IMPORT utility.

#### **REFERENCES**

Grants the privilege to create and drop a foreign key referencing the table as the parent.

If the authorization ID of the statement has one of:

- ACCESSCTRL or SECADM authority
- CONTROL privilege on the table
- REFERENCES WITH GRANT OPTION on the table

then the grantee(s) can create referential constraints using all columns of the table as parent key, even those added later using the ALTER TABLE statement. Otherwise, the privileges granted are all those grantable column REFERENCES privileges that the authorization ID of the statement has on the identified table.

The privilege can be granted on a nickname, although foreign keys cannot be defined to reference nicknames.

#### **REFERENCES (column-name,...)**

Grants the privilege to create and drop a foreign key using only those columns specified in the column list as a parent key. Each *column-name* must be an unqualified name that identifies a column of the table identified in the ON clause. Column level REFERENCES privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

#### **SELECT**

Grants the privilege to:

- Retrieve rows from the table or view.
- Create views on the table.
- Run the EXPORT utility against the table or view.

#### **UPDATE**

Grants the privilege to use the UPDATE statement on the table or updatable view identified in the ON clause.

If the authorization ID of the statement has one of:

- ACCESSCTRL or SECADM authority
- CONTROL privilege on the table or view
- UPDATE WITH GRANT OPTION on the table or view

then the grantee(s) can update all updatable columns of the table or view on which the grantor has with grant privilege as well as those columns added later using the ALTER TABLE statement. Otherwise, the privileges granted are all those grantable column UPDATE privileges that the authorization ID of the statement has on the identified table or view.

**UPDATE** (*column-name,...*)

Grants the privilege to use the UPDATE statement to update only those columns specified in the column list. Each *column-name* must be an unqualified name that identifies a column of the table or view identified in the ON clause. Column level UPDATE privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

**ON TABLE** *table-name* **or** *view-name* **or** *nickname*

Specifies the table, view, or nickname on which privileges are to be granted.

No privileges may be granted on an inoperative view or an inoperative materialized query table (SQLSTATE 51024). No privileges may be granted on a declared temporary table (SQLSTATE 42995).

**TO**

Specifies to whom the privileges are granted.

**USER**

Specifies that the *authorization-name* identifies a user.

**GROUP**

Specifies that the *authorization-name* identifies a group name.

**ROLE**

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

A privilege that is granted to a group is not used for authorization checking:

- On static DML statements in a package
- On a base table while processing a CREATE VIEW statement
- On a base table while processing a CREATE TABLE statement for a materialized query table

In DB2 Database for Linux, UNIX, and Windows, table privileges granted to groups only apply to statements that are dynamically prepared. For example, if the INSERT privilege on the PROJECT table has been granted to group D204 but not UBIQUITY (a member of D204) UBIQUITY could issue the statement:

```
EXEC SQL EXECUTE IMMEDIATE :INSERT_STRING;
```

where the content of the string is:

```
INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

but could not precompile or bind a program with the statement:

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

**PUBLIC**

Grants the privileges to a set of users (authorization IDs). For more information, see “Authorization, privileges and object ownership”.

(Previous restrictions on the use of privileges granted to PUBLIC for static SQL statements and the CREATE VIEW statement have been removed.)

## WITH GRANT OPTION

Allows the specified *authorization-names* to GRANT the privileges to others.

If the specified privileges include CONTROL, the WITH GRANT OPTION applies to all the applicable privileges except for CONTROL (SQLSTATE 01516).

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.) If CONTROL privilege is specified, privileges will only be granted if the authorization ID of the statement has ACCESSCTRL or SECADM authority (SQLSTATE 42501).

## Notes

- Privileges may be granted independently at every level of a table hierarchy. A user with a privilege on a supertable may affect the subtables. For example, an update specifying the supertable *T* may show up as a change to a row in the subtable *S* of *T* done by a user with UPDATE privilege on *T* but without UPDATE privilege on *S*. A user can only operate directly on the subtable if the necessary privilege is held on the subtable.
- Granting nickname privileges has no effect on data source object (table or view) privileges. Typically, data source privileges are required for the table or view that a nickname references when attempting to retrieve data.
- **Compatibilities:** For compatibility with DB2 for z/OS:
  - The following syntax is tolerated and ignored:
    - PUBLIC AT ALL LOCATIONS

## Examples

*Example 1:* Grant all privileges on the table WESTERN\_CR to PUBLIC.

**GRANT ALL ON WESTERN\_CR  
TO PUBLIC**

*Example 2:* Grant the appropriate privileges on the CALENDAR table so that users PHIL and CLAIRE can read it and insert new entries into it. Do not allow them to change or remove any existing entries.

**GRANT SELECT, INSERT ON CALENDAR  
TO USER PHIL, USER CLAIRE**

*Example 3:* Grant all privileges on the COUNCIL table to user FRANK and the ability to extend all privileges to others.

**GRANT ALL ON COUNCIL  
TO USER FRANK WITH GRANT OPTION**

*Example 4:* GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a user named JOHN. There is a user called JOHN and no group called JOHN.

**GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN**

or

**GRANT SELECT  
ON CORPDATA.EMPLOYEE TO USER JOHN**

*Example 5:* GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a group named JOHN. There is a group called JOHN and no user called JOHN.

**GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN**

or

**GRANT SELECT ON CORPDATA.EMPLOYEE TO GROUP JOHN**

*Example 6:* GRANT INSERT and SELECT on table T1 to both a group named D024 and a user named D024.

**GRANT INSERT, SELECT ON TABLE T1  
TO GROUP D024, USER D024**

In this case, both the members of the D024 group and the user D024 would be allowed to INSERT into and SELECT from the table T1. Also, there would be two rows added to the SYSCAT.TABAUTH catalog view.

*Example 7:* GRANT INSERT, SELECT, and CONTROL on the CALENDAR table to user FRANK. FRANK must be able to pass the privileges on to others.

**GRANT CONTROL ON TABLE CALENDAR  
TO FRANK WITH GRANT OPTION**

| The result of this statement is a warning (SQLSTATE 01516) that CONTROL was  
| not given the WITH GRANT OPTION. Frank now has the ability to grant any  
| privilege on CALENDAR including INSERT and SELECT as required. FRANK  
| cannot grant CONTROL on CALENDAR to other users unless he has  
| ACCESSCTRL or SECADM authority.

*Example 8:* User JON created a nickname for an Oracle table that had no index. The nickname is ORAREM1. Later, the Oracle DBA defined an index for this table. User SHAWN now wants DB2 to know that this index exists, so that the optimizer can devise strategies to access the table more efficiently. SHAWN can inform DB2 of the index by creating an index specification for ORAREM1. Give SHAWN the index privilege on this nickname, so that he can create the index specification.

## INSERT

The INSERT statement inserts rows into a table, nickname, or view, or the underlying tables, nicknames, or views of the specified fullselect. Inserting a row into a nickname inserts the row into the data source object to which the nickname refers. Inserting a row into a view also inserts the row into the table on which the view is based, if no INSTEAD OF trigger is defined for the insert operation on this view. If such a trigger is defined, the trigger will be executed instead.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- INSERT privilege on the target table, view, or nickname
- CONTROL privilege on the target table, view, or nickname
- DATAACCESS authority

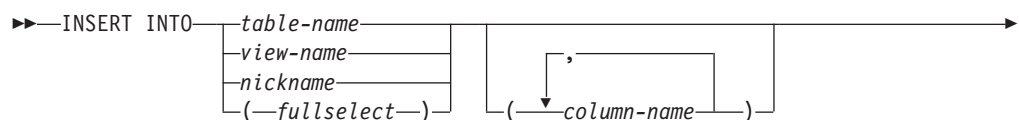
In addition, for each table, view, or nickname referenced in any fullselect used in the INSERT statement, the privileges held by the authorization ID of the statement must include at least one of the following:

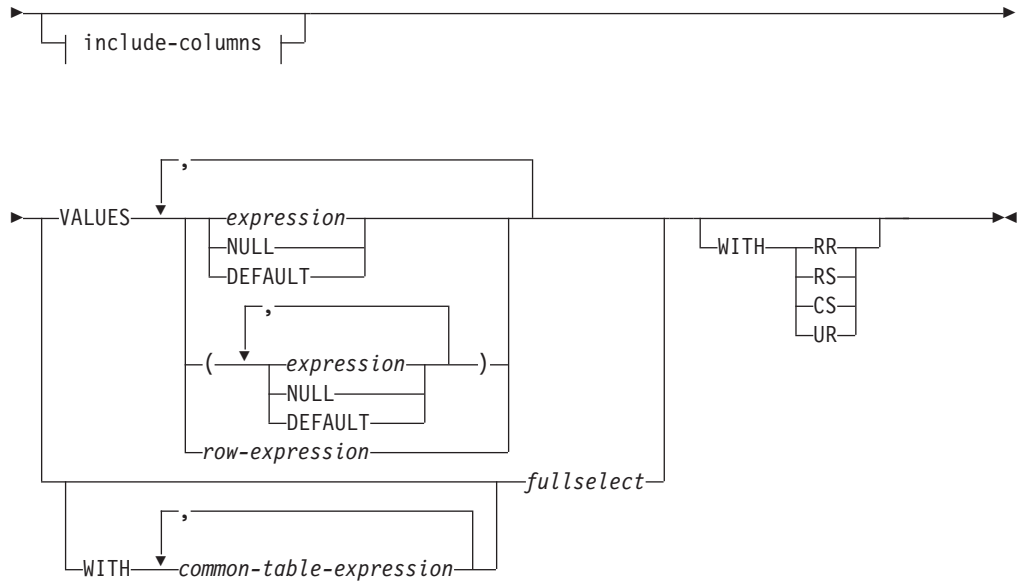
- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

GROUP privileges are not checked for static INSERT statements.

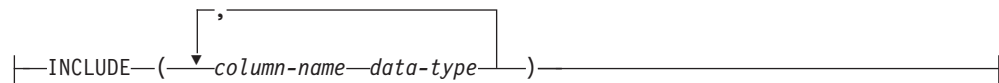
If the target of the insert operation is a nickname, the privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges required for the operation on the object at the data source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

### Syntax





**include-columns:**



**Description**

**INTO** *table-name, view-name, nickname, or (fullselect)*

Identifies the object of the insert operation. The name must identify a table, view or nickname that exists at the application server, but it must not identify a catalog table, a system-maintained materialized query table, a view of a catalog table, or a read-only view, unless an INSTEAD OF trigger is defined for the insert operation on the subject view. Rows inserted into a nickname are placed in the data source object to which the nickname refers.

If the object of the insert operation is a fullselect, the fullselect must be insertable, as defined in the “Insertable views” Notes item in the description of the CREATE VIEW statement.

If no INSTEAD OF trigger exists for the insert operation on this view, a value cannot be inserted into a view column that is derived from:

- A constant, expression, or scalar function
- The same base table column as some other column of the view

If the object of the insert operation is a view with such columns, a list of column names must be specified, and the list must not identify these columns.

A row can be inserted into a view or a fullselect that is defined using a UNION ALL if the row satisfies the check constraints of exactly one of the underlying base tables. If a row satisfies the check constraints of more than one table, or no table at all, an error is returned (SQLSTATE 23513).

*(column-name,...)*

Specifies the columns for which insert values are provided. Each name must identify a column of the specified table, view, or nickname, or a column in the

fullselect. The same column must not be identified more than once. A column that cannot accept inserted values (for example, a column based on an expression) must not be identified.

Omission of the column list is an implicit specification of a list in which every column of the table (that is not implicitly hidden) or view, or every item in the select-list of the fullselect is identified in left-to-right order. This list is established when the statement is prepared and, therefore, does not include columns that were added to a table after the statement was prepared.

#### *include-columns*

Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the INSERT statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

#### **INCLUDE**

Specifies a list of columns to be included in the intermediate result table of the INSERT statement. This clause can only be specified if the INSERT statement is nested in the FROM clause of a fullselect.

#### *column-name*

Specifies a column of the intermediate result table of the INSERT statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

#### *data-type*

Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

#### **VALUES**

Introduces one or more rows of values to be inserted.

Each row specified in the VALUES clause must be assignable to the implicit or explicit column list and the columns identified in the INCLUDE clause, unless a row variable is used. When a row value list in parentheses is specified, the first value is inserted into the first column in the list, the second value into the second column, and so on. When a row expression is specified, the number of fields in the row type must match the number of names in the implicit or explicit column list.

#### *expression*

An *expression* can be any expression defined in “Expressions”. If *expression* is a row type, it must not appear in parentheses.

#### **NULL**

Specifies the null value and should only be specified for nullable columns.

#### **DEFAULT**

Specifies that the default value is to be used. The result of specifying DEFAULT depends on how the column was defined, as follows:

- If the column was defined as a generated column based on an expression, the column value is generated by the system, based on that expression.
- If the IDENTITY clause is used, the value is generated by the database manager.
- If the ROW CHANGE TIMESTAMP clause is used, the value for each inserted row is generated by the database manager as a timestamp that is unique for the table partition within the database partition.



- If the WITH DEFAULT clause is used, the value inserted is as defined for the column (see *default-clause* in “CREATE TABLE”).
- If the NOT NULL clause is used and the GENERATED clause is not used, or the WITH DEFAULT clause is not used or DEFAULT NULL is used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).
- When inserting into a nickname, the DEFAULT keyword will be passed through the INSERT statement to the data source only if the data source supports the DEFAULT keyword in its query language syntax.

*row-expression*

Specifies any row expression of the type described in “Row expressions” that does not include a column name. The number of fields in the row must match the target of the insert and each field must be assignable to the corresponding column.

**WITH** *common-table-expression*

Defines a common table expression for use with the fullselect that follows.

*fullselect*

Specifies a set of new rows in the form of the result table of a fullselect. There may be one, more than one, or none. If the result table is empty, SQLCODE is set to +100 and SQLSTATE is set to '02000'.

When the base object of the INSERT and the base object of the fullselect or any subquery of the fullselect, are the same table, the fullselect is completely evaluated before any rows are inserted.

The number of columns in the result table must equal the number of names in the column list. The value of the first column of the result is inserted in the first column in the list, the second value in the second column, and so on.

**WITH**

Specifies the isolation level at which the fullselect is executed.

**RR**

Repeatable Read

**RS**

Read Stability

**CS**

Cursor Stability

**UR**

Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. The WITH clause has no effect on nicknames, which always use the default isolation level of the statement.

**Rules**

- **Triggers:** INSERT statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the inserted values. If an insert operation into a view causes an INSTEAD OF trigger to fire, validity, referential integrity, and constraints will be checked against the updates that are performed in the trigger, and not against the view that caused the trigger to fire, or its underlying tables.
- **Default values:** The value inserted in any column that is not in the column list is either the default value of the column or null. Columns that do not allow null

values and are not defined with NOT NULL WITH DEFAULT must be included in the column list. Similarly, if you insert into a view, the value inserted into any column of the base table that is not in the view is either the default value of the column or null. Hence, all columns of the base table that are not in the view must have either a default value or allow null values. The only value that can be inserted into a generated column defined with the GENERATED ALWAYS clause is DEFAULT (SQLSTATE 428C9).

- **Length:** If the insert value of a column is a number, the column must be a numeric column with the capacity to represent the integral part of the number. If the insert value of a column is a string, the column must either be a string column with a length attribute at least as great as the length of the string, or a datetime column if the string represents a date, time, or timestamp.
- **Assignment:** Insert values are assigned to columns in accordance with specific assignment rules.
- **Validity:** If the table named, or the base table of the view named, has one or more unique indexes, each row inserted into the table must conform to the constraints imposed by those indexes. If a view whose definition includes WITH CHECK OPTION is named, each row inserted into the view must conform to the definition of the view. For an explanation of the rules governing this situation, see "CREATE VIEW".
- **Referential integrity:** For each constraint defined on a table, each non-null insert value of the foreign key must be equal to a primary key value of the parent table.
- **Check constraint:** Insert values must satisfy the check conditions of the check constraints defined on the table. An INSERT to a table with check constraints defined has the constraint conditions evaluated once for each row that is inserted.
- **XML values:** A value that is inserted into an XML column must be a well-formed XML document (SQLSTATE 2200M).
- **Security policy:** If the identified table or the base table of the identified view is protected with a security policy, the session authorization ID must have the label-based access control (LBAC) credentials that allow:
  - Write access to all protected columns for which a data value is explicitly provided (SQLSTATE 42512)
  - Write access for any explicit value provided for a DB2SECURITYLABEL column for security policies that were created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option (SQLSTATE 23523)

The session authorization ID must also have been granted a security label for write access for the security policy if an implicit value is used for a DB2SECURITYLABEL column (SQLSTATE 23523), which can happen when:

- A value for the DB2SECURITYLABEL column is not explicitly provided
- A value for the DB2SECURITYLABEL column is explicitly provided but the session authorization ID does not have write access for that value, and the security policy is created with the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option

## Notes

- After execution of an INSERT statement, the value of the third variable of the SQLERRD(3) portion of the SQLCA indicates the number of rows that were passed to the insert operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW\_COUNT variable of the GET DIAGNOSTICS statement. SQLERRD(5) contains the count of all triggered insert, update and delete operations.

- Unless appropriate locks already exist, one or more exclusive locks are acquired at the execution of a successful INSERT statement. Until the locks are released, an inserted row can only be accessed by:
  - The application process that performed the insert.
  - Another application process using isolation level UR through a read-only cursor, SELECT INTO statement, or subselect used in a subquery.
- For further information about locking, see the description of the COMMIT, ROLLBACK, and LOCK TABLE statements.
- If an application is running against a partitioned database, and it is bound with option INSERT BUF, then INSERT with VALUES statements which are not processed using EXECUTE IMMEDIATE may be buffered. DB2 assumes that such an INSERT statement is being processed inside a loop in the application's logic. Rather than execute the statement to completion, it attempts to buffer the new row values in one or more buffers. As a result the actual insertions of the rows into the table are performed later, asynchronous with the application's INSERT logic. Be aware that this asynchronous insertion may cause an error related to an INSERT to be returned on some other SQL statement that follows the INSERT in the application.

This has the potential to dramatically improve INSERT performance, but is best used with clean data, due to the asynchronous nature of the error handling.

- When a row is inserted into a table that has an identity column, DB2 generates a value for the identity column.
  - For a GENERATED ALWAYS identity column, DB2 always generates the value.
  - For a GENERATED BY DEFAULT column, if a value is not explicitly specified (with a VALUES clause, or subselect), DB2 generates a value.

The first value generated by DB2 is the value of the START WITH specification for the identity column.

- When a value is inserted for a user-defined distinct type identity column, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column. (There is no casting of the previous value to the source type prior to the computation.)
- When inserting into a GENERATED ALWAYS identity column, DB2 will always generate a value for the column, and users must not specify a value at insertion time. If a GENERATED ALWAYS identity column is listed in the column-list of the INSERT statement, with a non-DEFAULT value in the VALUES clause, an error occurs (SQLSTATE 428C9).

For example, assuming that EMPID is defined as an identity column that is GENERATED ALWAYS, then the command:

```
INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
VALUES (:hv_valid_emp_id, :hv_name, :hv_addr)
```

will result in an error.

- When inserting into a GENERATED ALWAYS ROW CHANGE TIMESTAMP column, DB2 will always generate a value for the column, and users must not specify a value at insertion time (SQLSTATE 428C9). The value generated by DB2 is unique for each row inserted on the database partition.
- When inserting into a GENERATED BY DEFAULT column, DB2 will allow an actual value for the column to be specified within the VALUES clause, or from a subselect. However, when a value is specified in the VALUES clause, DB2 does not perform any verification of the value. To guarantee uniqueness of IDENTITY column values, a unique index on the identity column must be created.

When inserting into a table with a GENERATED BY DEFAULT identity column, without specifying a column list, the VALUES clause can specify the DEFAULT keyword to represent the value for the identity column. DB2 will generate the value for the identity column.

```
INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
VALUES (DEFAULT, :hv_name, :hv_addr)
```

In this example, EMPID is defined as an identity column, and thus the value inserted into this column is generated by DB2.

- The rules for inserting into an identity column with a subselect are similar to those for an insert with a VALUES clause. A value for an identity column may only be specified if the identity column is defined as GENERATED BY DEFAULT.

For example, assume T1 and T2 are tables with the same definition, both containing columns *intcol1* and *identcol2* (both are type INTEGER and the second column has the identity attribute). Consider the following insert:

```
INSERT INTO T2
SELECT *
FROM T1
```

This example is logically equivalent to:

```
INSERT INTO T2 (intcol1,identcol2)
SELECT intcol1, identcol2
FROM T1
```

In both cases, the INSERT statement is providing an explicit value for the identity column of T2. This explicit specification can be given a value for the identity column, but the identity column in T2 must be defined as GENERATED BY DEFAULT. Otherwise, an error will result (SQLSTATE 428C9).

If there is a table with a column defined as a GENERATED ALWAYS identity, it is still possible to propagate all other columns from a table with the same definition. For example, given the example tables T1 and T2 described above, the *intcol1* values from T1 to T2 can be propagated with the following SQL:

```
INSERT INTO T2 (intcol1)
SELECT intcol1
FROM T1
```

Note that, because *identcol2* is not specified in the column-list, it will be filled in with its default (generated) value.

- When inserting a row into a single column table where the column is defined as a GENERATED ALWAYS identity column or a ROW CHANGE TIMESTAMP column, it is possible to specify a VALUES clause with the DEFAULT keyword. In this case, the application does not provide any value for the table, and DB2 generates the value for the identity or ROW CHANGE TIMESTAMP column.

```
INSERT INTO IDTABLE
VALUES(DEFAULT)
```

Assuming the same single column table for which the column has the identity attribute, to insert multiple rows with a single INSERT statement, the following INSERT statement could be used:

```
INSERT INTO IDTABLE
VALUES (DEFAULT), (DEFAULT), (DEFAULT), (DEFAULT)
```

- When DB2 generates a value for an identity column, that generated value is consumed; the next time that a value is needed, DB2 will generate a new value. This is true even when an INSERT statement involving an identity column fails or is rolled back.

For example, assume that a unique index has been created on the identity column. If a duplicate key violation is detected in generating a value for an identity column, an error occurs (SQLSTATE 23505) and the value generated for the identity column is considered to be consumed. This can occur when the identity column is defined as GENERATED BY DEFAULT and the system tries to generate a new value, but the user has explicitly specified values for the identity column in previous INSERT statements. Reissuing the same INSERT statement in this case can lead to success. DB2 will generate the next value for the identity column, and it is possible that this next value will be unique, and that this INSERT statement will be successful.

- If the maximum value for the identity column is exceeded (or minimum value for a descending sequence) in generating a value for an identity column, an error occurs (SQLSTATE 23522). In this situation, the user would have to DROP and CREATE a new table with an identity column having a larger range (that is, change the data type or increment value for the column to allow for a larger range of values).

For example, an identity column may have been defined with a data type of SMALLINT, and eventually the column runs out of assignable values. To redefine the identity column as INTEGER, the data would need to be unloaded, the table would have to be dropped and recreated with a new definition for the column, and then the data would be reloaded. When the table is redefined, it needs to specify a START WITH value for the identity column such that the next value generated by DB2 will be the next value in the original sequence. To determine the end value, issue a query using MAX of the identity column (for an ascending sequence), or MIN of the identity column (for a descending sequence), before unloading the data.

## Examples

*Example 1:* Insert a new department with the following specifications into the DEPARTMENT table:

- Department number (DEPTNO) is 'E31'
- Department name (DEPTNAME) is 'ARCHITECTURE'
- Managed by (MGRNO) a person with number '00390'
- Reports to (ADMRDEPT) department 'E01'.

```
INSERT INTO DEPARTMENT
VALUES ('E31', 'ARCHITECTURE', '00390', 'E01')
```

*Example 2:* Insert a new department into the DEPARTMENT table as in example 1, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
VALUES ('E31', 'ARCHITECTURE', 'E01')
```

*Example 3:* Insert two new departments using one statement into the DEPARTMENT table as in example 2, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
VALUES ('B11', 'PURCHASING', 'B01'),
('E41', 'DATABASE ADMINISTRATION', 'E01')
```

*Example 4:* Create a temporary table MA\_EMP\_ACT with the same columns as the EMP\_ACT table. Load MA\_EMP\_ACT with the rows from the EMP\_ACT table with a project number (PROJNO) starting with the letters 'MA'.

```
CREATE TABLE MA_EMP_ACT
 (EMPNO CHAR(6) NOT NULL,
 PROJNO CHAR(6) NOT NULL,
 ACTNO SMALLINT NOT NULL,
 EMPTIME DEC(5,2),
 EMSTDATE DATE,
 EMENDATE DATE)
INSERT INTO MA_EMP_ACT
 SELECT * FROM EMP_ACT
 WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

*Example 5:* Use a C program statement to add a skeleton project to the PROJECT table. Obtain the project number (PROJNO), project name (PROJNAME), department number (DEPTNO), and responsible employee (RESPEMP) from host variables. Use the current date as the project start date (PRSTDATE). Assign a NULL value to the remaining columns in the table.

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTDATE)
 VALUES (:PRJNO, :PRJNM, :DPTNO, :REMP, CURRENT DATE);
```

*Example 6:* Specify an INSERT statement as the *data-change-table-reference* within a SELECT statement. Define an extra include column whose values are specified in the VALUES clause, which is then used as an ordering column for the inserted rows.

```
SELECT inorder.ordernum
 FROM (INSERT INTO orders(custno)INCLUDE (insertnum integer)
 VALUES(:cnum1, 1), (:cnum2, 2)) InsertedOrders
 ORDER BY insertnum;
```

*Example 7:* Use a C program statement to add a document to the DOCUMENTS table. Obtain values for the document ID (DOCID) column and the document data (XMLDOC) column from a host variable that binds to an SQL TYPE IS XML AS BLOB\_FILE.

```
EXEC SQL INSERT INTO DOCUMENTS
 (DOCID, XMLDOC) VALUES (:docid, :xmldoc)
```

*Example 8:* For the following INSERT statements, assume that table SALARY\_INFO is defined with three columns, and that the last column is an implicitly hidden ROW CHANGE TIMESTAMP column. In the following statement, the implicitly hidden column is explicitly referenced in the column list and a value is provided for it in the VALUES clause.

```
INSERT INTO SALARY_INFO (LEVEL, SALARY, UPDATE_TIME)
 VALUES (2, 30000, CURRENT_TIMESTAMP)
```

The following INSERT statement uses an implicit column list. An implicit column list does not include implicitly hidden columns, so the VALUES clause only contains values for the other two columns.

```
INSERT INTO SALARY_INFO VALUES (2, 30000)
```

In this case, the UPDATE\_TIME column must be defined to have a default value, and that default value is used for the row that is inserted.

---

## RENAME

The RENAME statement renames an existing table or index.



## Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

## Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the table or index
- Ownership of the table or index, as recorded in the OWNER column of the SYSCAT.TABLES catalog view for a table, and the SYSCAT.INDEXES catalog view for an index
- ALTERIN privilege on the schema
- DBADM authority

## Syntax

```
➤ RENAME TABLE source-table-name TO target-identifier ➤
 INDEX source-index-name
```

## Description

### TABLE *source-table-name*

Names the existing table that is to be renamed. The name, including the schema name, must identify a table that already exists in the database (SQLSTATE 42704). It must not be the name of a catalog table (SQLSTATE 42832), a materialized query table, a typed table (SQLSTATE 42997), a created temporary table, a declared global temporary table (SQLSTATE 42995), a nickname, or an object other than a table or an alias (SQLSTATE 42809). The TABLE keyword is optional.

### INDEX *source-index-name*

Names the existing index that is to be renamed. The name, including the schema name, must identify an index that already exists in the database (SQLSTATE 42704). It must not be the name of an index on a created temporary table or a declared global temporary table (SQLSTATE 42995). The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42832).

### *target-identifier*

Specifies the new name for the table or index without a schema name. The schema name of the source object is used to qualify the new name for the object. The qualified name must *not* identify a table, view, alias, or index that already exists in the database (SQLSTATE 42710).

## Rules

When renaming a table, the source table must not:

- Be referenced in any existing materialized query table definitions
- Be the subject table of an existing trigger
- Be a parent or dependent table in any referential integrity constraints



- Be the scope of any existing reference column
- Be referenced by an XSR object that has been enabled for decomposition

An error (SQLSTATE 42986) is returned if the source table violates one or more of these conditions.

When renaming an index:

- The source index must not be a system-generated index for an implementation table on which a typed table is based (SQLSTATE 42858).

## Notes

- Catalog entries are updated to reflect the new table or index name.
- *All* authorizations associated with the source table or index name are *transferred* to the new table or index name (the authorization catalog tables are updated appropriately).
- Indexes defined over the source table are *transferred* to the new table (the index catalog tables are updated appropriately).
- RENAME TABLE invalidates any packages that are dependent on the source table. RENAME INDEX invalidates any packages that are dependent on the source index.
- If an alias is used for the *source-table-name*, it must resolve to a table name. The table is renamed within the schema of this table. The alias is not changed by the RENAME statement and continues to refer to the old table name.
- A table with primary key or unique constraints can be renamed if none of the primary key or unique constraints are referenced by any foreign key.

## Examples

Change the name of the EMP table to EMPLOYEE.

```
RENAME TABLE EMP TO EMPLOYEE
RENAME TABLE ABC.EMP TO EMPLOYEE
```

Change the name of the index NEW-IND to IND.

```
RENAME INDEX NEW-IND TO IND
RENAME INDEX ABC.NEW-IND TO IND
```

---

## RENAME TABLESPACE

The RENAME TABLESPACE statement renames an existing table space.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include either SYSCTRL or SYSADM authority.

## Syntax

►—RENAME—TABLESPACE—*source-tablespace-name*—TO—*target-tablespace-name*—►

## Description

*source-tablespace-name*

Specifies the existing table space that is to be renamed, as a one-part name. It is an SQL identifier (either ordinary or delimited). The table space name must identify a table space that already exists in the catalog (SQLSTATE 42704).

*target-tablespace-name*

Specifies the new name for the table space, as a one-part name. It is an SQL identifier (either ordinary or delimited). The new table space name must *not* identify a table space that already exists in the catalog (SQLSTATE 42710), and it cannot start with 'SYS' (SQLSTATE 42939).

## Rules

- The SYSCATSPACE table space cannot be renamed (SQLSTATE 42832).
- Any table spaces with "rollforward pending" or "rollforward in progress" states cannot be renamed (SQLSTATE 55039)

## Notes

- Renaming a table space will update the minimum recovery time of a table space to the point in time when the rename took place. This implies that a roll forward at the table space level must be to at least this point in time.
- The new table space name must be used when restoring a table space from a backup image, where the rename was done after the backup was created.

## Example

Change the name of the table space USERSPACE1 to DATA2000:

```
RENAME TABLESPACE USERSPACE1 TO DATA2000
```

---

## REVOKE (Database Authorities)

This form of the REVOKE statement revokes authorities that apply to the entire database.

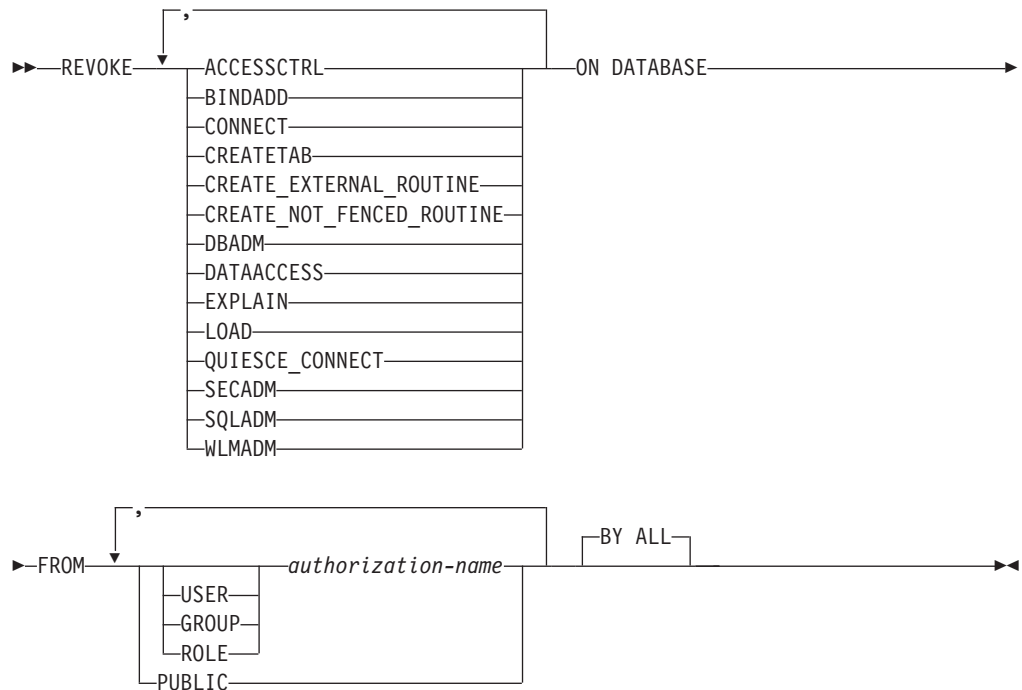
### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

To revoke ACCESSCTRL, DATAACCESS, DBADM, or SECADM authority, SECADM is required. To revoke other authorities, ACCESSCTRL or SECADM authority is required.

## Syntax



## Description

### ACCESSCTRL

Revokes the authority to grant and revoke most database authorities and object privileges.

### BINDADD

Revokes the authority to create packages. The creator of a package automatically has the CONTROL privilege on that package and retains this privilege even if his BINDADD authority is subsequently revoked.

The BINDADD authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority.

### CONNECT

Revokes the authority to access the database.

Revoking the CONNECT authority from a user does not affect any privileges that were granted to that user on objects in the database. If the user is subsequently granted the CONNECT authority again, all previously held privileges are still valid (assuming they were not explicitly revoked).

The CONNECT authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

### CREATETAB

Revokes the authority to create tables. The creator of a table automatically has the CONTROL privilege on that table, and retains this privilege even if his CREATETAB authority is subsequently revoked.

The CREATETAB authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

### CREATE\_EXTERNAL\_ROUTINE

Revokes the authority to register external routines. Once an external routine has been registered, it continues to exist, even if CREATE\_EXTERNAL\_ROUTINE is subsequently revoked from the authorization ID that registered the routine.

CREATE\_EXTERNAL\_ROUTINE authority cannot be revoked from an *authorization-name* holding DBADM or CREATE\_NOT\_FENCED\_ROUTINE authority without also revoking DBADM or CREATE\_NOT\_FENCED\_ROUTINE authority (SQLSTATE 42504).

### CREATE\_NOT\_FENCED\_ROUTINE

Revokes the authority to register routines that execute in the database manager's process. Once a routine has been registered as not fenced, it continues to run in this manner, even if CREATE\_NOT\_FENCED\_ROUTINE is subsequently revoked from the authorization ID that registered the routine.

CREATE\_NOT\_FENCED\_ROUTINE authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

### DATAACCESS

Revokes the authority to access data.

### DBADM

Revokes the DBADM authority.

DBADM authority cannot be revoked from PUBLIC (because it cannot be granted to PUBLIC).

#### CAUTION:

**Revoking DBADM authority does not automatically revoke any privileges that were held by the *authorization-name* on objects in the database.**

### EXPLAIN

Revokes the authority to explain, prepare, and describe static and dynamic statements without requiring access to data.

### LOAD

Revokes the authority to LOAD in this database.

### QUIESCE\_CONNECT

Revokes the authority to access the database while it is quiesced.

### SECADM

Revokes the authority to administer database security.

### SQLADM

Revokes the authority to monitor and tune SQL statements.

### WLMADM

Revokes the authority to manage workload manager objects.

### FROM

Indicates from whom the authorities are revoked.

### USER

Specifies that the *authorization-name* identifies a user.

### GROUP

Specifies that the *authorization-name* identifies a group name.

### ROLE

Specifies that the *authorization-name* identifies a role name.

*authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

#### **PUBLIC**

Revokes the authorities from PUBLIC.

#### **BY ALL**

Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

### **Rules**

**Security administrator mandatory:** The database must have at least one authorization ID of type USER with the SECADM authority. The SECADM authority cannot be revoked from every user authorization ID (SQLSTATE 42523).

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.DBAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

### **Notes**

- Revoking a specific privilege does not necessarily revoke the ability to perform an action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds a higher level authority, such as DBADM.
- **Compatibilities:**
  - For compatibility with previous versions of DB2:
    - CREATE\_NOT\_FENCED can be specified in place of CREATE\_NOT\_FENCED\_ROUTINE
  - For compatibility with DB2 for z/OS:
    - SYSTEM can be specified in place of DATABASE
    - NOT INCLUDING DEPENDANT PRIVILEGES may be specified as a syntax alternative

### **Examples**

*Example 1:* Given that USER6 is only a user and not a group, revoke the privilege to create tables from the user USER6.

```
REVOKE CREATETAB ON DATABASE FROM USER6
```

*Example 2:* Revoke BINDADD authority on the database from a group named D024. There are two rows in the SYSCAT.DBAUTH catalog view for this grantee; one with a GRANTEETYPE of U and one with a GRANTEETYPE of G.

```
REVOKE BINDADD ON DATABASE FROM GROUP D024
```

In this case, the GROUP keyword must be specified; otherwise an error will occur (SQLSTATE 56092).

*Example 3:* Revoke security administrator authority from user Walid.

```
REVOKE SECADM ON DATABASE FROM USER Walid
```

## REVOKE (Exemption)

This form of the REVOKE statement revokes an exemption to a label-based access control (LBAC) access rule.

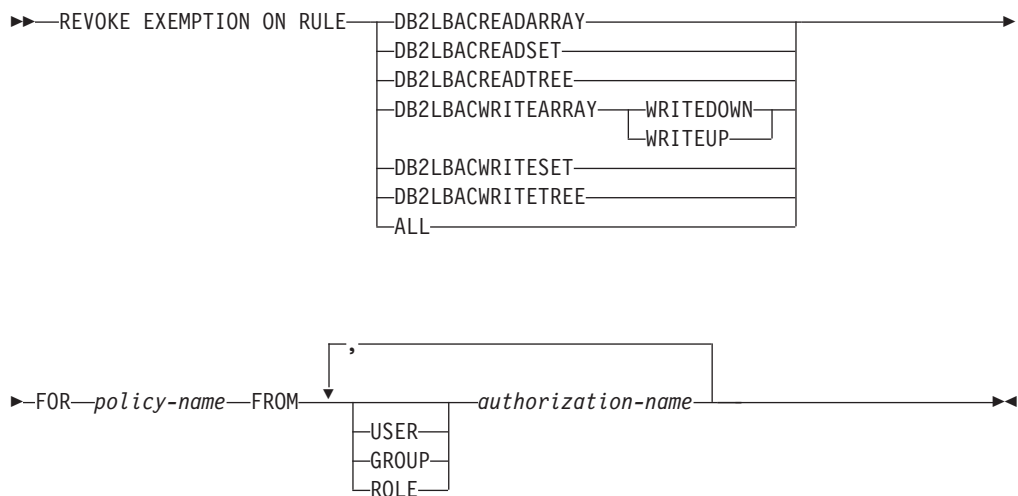
### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

### Syntax



### Description

#### EXEMPTION ON RULE

Revokes the exemption on an access rule.

#### DB2LBACREADARRAY

Revokes an exemption on the predefined DB2LBACREADARRAY rule.

#### DB2LBACREADSET

Revokes an exemption on the predefined DB2LBACREADSET rule.

#### DB2LBACREADTREE

Revokes an exemption on the predefined DB2LBACREADTREE rule.

**DB2LBACWRITEARRAY**

Revokes an exemption on the predefined DB2LBACWRITEARRAY rule.

**WRITEDOWN**

Specifies that the exemption only applies to write down.

**WRITEUP**

Specifies that the exemption only applies to write up.

**DB2LBACWRITESSET**

Revokes an exemption on the predefined DB2LBACWRITESSET rule.

**DB2LBACWRITETREE**

Revokes an exemption on the predefined DB2LBACWRITETREE rule.

**ALL**

Revokes the exemptions on all of the predefined rules.

**FOR** *policy-name*

Specifies the name of the security policy on which exemptions are to be revoked.

**FROM**

Specifies from whom the exemption is revoked.

**USER**

Specifies that the *authorization-name* identifies a user.

**GROUP**

Specifies that the *authorization-name* identifies a group name.

**ROLE**

Specifies that the *authorization-name* identifies a role name.

*authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

**Rules**

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.SECURITYPOLICYEXEMPTIONS catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

**Examples**

*Example 1:* Revoke the exemption on access rule DB2LBACREADSET for security policy DATA\_ACCESS from user WALID.

```
REVOKE EXEMPTION ON RULE DB2LBACREADSET FOR DATA_ACCESS
FROM USER WALID
```

*Example 2:* Revoke an exemption on access rule DB2LBACWRITEARRAY with the WRITEDOWN option for security policy DATA\_ACCESS from user BOBBY.

```
REVOKE EXEMPTION ON RULE DB2LBACWRITEARRAY WRITEDOWN
FOR DATA_ACCESS FROM USER BOBBY
```



*Example 3:* Revoke an exemption on access rule DB2LBACWRITEARRAY with the WRITEUP option for security policy DATA\_ACCESS from user BOBBY.

```
REVOKE EXEMPTION ON RULE DB2LBACWRITEARRAY WRITEUP
FOR DATA_ACCESS FROM USER BOBBY
```

## REVOKE (Index Privileges)

This form of the REVOKE statement revokes the CONTROL privilege on an index.

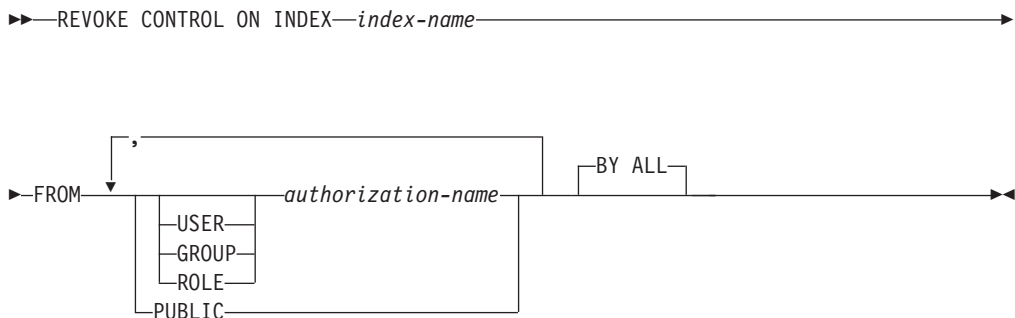
### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

### Syntax



### Description

#### CONTROL

Revokes the privilege to drop the index. This is the CONTROL privilege for indexes, which is automatically granted to creators of indexes.

#### ON INDEX *index-name*

Specifies the name of the index on which the CONTROL privilege is to be revoked.

#### FROM

Indicates from whom the privileges are revoked.

#### USER

Specifies that the *authorization-name* identifies a user.

#### GROUP

Specifies that the *authorization-name* identifies a group name.

#### ROLE

Specifies that the *authorization-name* identifies a role name.

*authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

#### **PUBLIC**

Revokes the privileges from PUBLIC.

#### **BY ALL**

Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

#### **Rules**

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.INDEXAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

#### **Notes**

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds authorities such as ALTERIN on the schema of an index.

#### **Examples**

*Example 1:* Given that USER4 is only a user and not a group, revoke the privilege to drop an index DEPTIDX from the user USER4.

```
REVOKE CONTROL ON INDEX DEPTIDX FROM KIESLER
```

*Example 2:* Revoke the privilege to drop an index LUNCHITEMS from the user CHEF and the group WAITERS.

```
REVOKE CONTROL ON INDEX LUNCHITEMS
FROM USER CHEF, GROUP WAITERS
```

---

## **REVOKE (Package Privileges)**

This form of the REVOKE statement revokes CONTROL, BIND, and EXECUTE privileges against a package.

#### **Invocation**

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

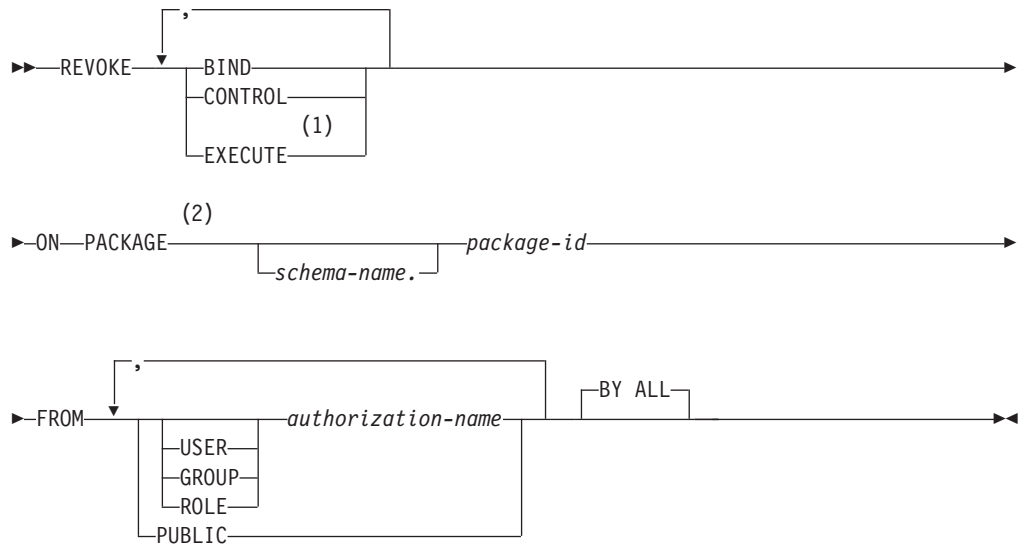
#### **Authorization**

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced package
- ACCESSCTRL or SECADM authority

ACCESSCTRL or SECADM authority is required to revoke the CONTROL privilege.

## Syntax



### Notes:

- 1 RUN can be used as a synonym for EXECUTE.
- 2 PROGRAM can be used as a synonym for PACKAGE.

## Description

### BIND

Revokes the privilege to execute BIND or REBIND on—or to add a new version of—the referenced package.

The BIND privilege cannot be revoked from an *authorization-name* that holds CONTROL privilege on the package, without also revoking the CONTROL privilege.

### CONTROL

Revokes the privilege to drop the package and to extend package privileges to other users.

Revoking CONTROL does not revoke the other package privileges.

### EXECUTE

Revokes the privilege to execute the package.

The EXECUTE privilege cannot be revoked from an *authorization-name* that holds CONTROL privilege on the package without also revoking the CONTROL privilege.

### ON PACKAGE *schema-name.package-id*

Specifies the name of the package on which privileges are to be revoked. If a

schema name is not specified, the package ID is implicitly qualified by the default schema. The revoking of a package privilege applies to all versions of the package.

#### **FROM**

Indicates from whom the privileges are revoked.

#### **USER**

Specifies that the *authorization-name* identifies a user.

#### **GROUP**

Specifies that the *authorization-name* identifies a group name.

#### **ROLE**

Specifies that the *authorization-name* identifies a role name.

*authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

#### **PUBLIC**

Revokes the privileges from PUBLIC.

#### **BY ALL**

Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

### **Rules**

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.PACKAGEAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

### **Notes**

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds privileges such as ALTERIN on the schema of a package.

### **Examples**

*Example 1:* Revoke the EXECUTE privilege on package CORPDATA.PKGA from PUBLIC.

```
REVOKE EXECUTE
ON PACKAGE CORPDATA.PKGA
FROM PUBLIC
```

*Example 2:* Revoke CONTROL authority on the RRSP\_PKG package for the user FRANK and for PUBLIC.

```

REVOKE CONTROL
ON PACKAGE RRSP_PKG
FROM USER FRANK, PUBLIC

```

## REVOKE (Role)

This form of the REVOKE statement revokes roles from users, groups, or other roles.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

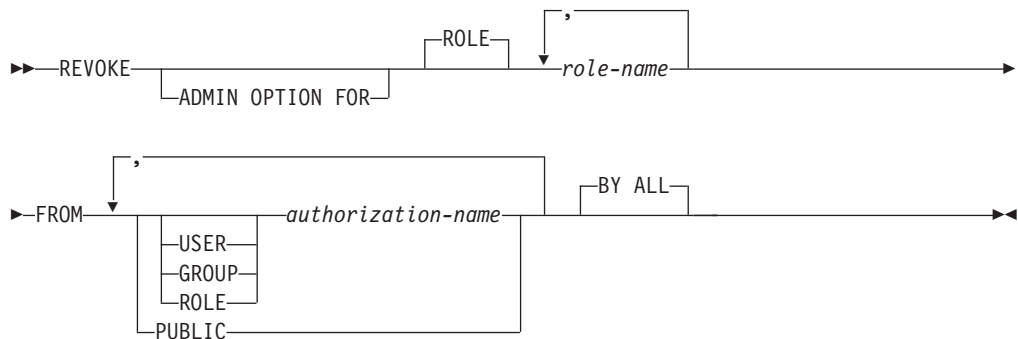
### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- The WITH ADMIN OPTION on the role
- SECADM authority

SECADM authority is required to revoke the ADMIN OPTION FOR *role-name* from an *authorization-name* or to revoke a *role-name* from an *authorization-name* that has the WITH ADMIN OPTION on that role.

### Syntax



### Description

#### ADMIN OPTION FOR

Revokes the WITH ADMIN OPTION on *role-name*. The WITH ADMIN OPTION on *role-name* must be held by *authorization-name* or by PUBLIC, if PUBLIC is specified (SQLSTATE 42504). If the ADMIN OPTION FOR clause is specified, only the WITH ADMIN OPTION on ROLE *role-name* is revoked, not the role itself.

#### ROLE *role-name*

Specifies the role that is to be revoked. The *role-name* must identify an existing role at the current server (SQLSTATE 42704) that has been granted to *authorization-name* or to PUBLIC, if PUBLIC is specified (SQLSTATE 42504).

#### FROM

Specifies from whom the role is revoked.

## USER

Specifies that the *authorization-name* identifies a user.

## GROUP

Specifies that the *authorization-name* identifies a group.

## ROLE

Specifies that the *authorization-name* identifies an existing role at the current server (SQLSTATE 42704).

*authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

## PUBLIC

Revokes the specified roles from PUBLIC.

## BY ALL

Revokes the *role-name* from each specified *authorization-name* that was explicitly granted that role, regardless of who granted it. This is the default behavior.

## Rules

- For each *authorization-name* specified, if none of the keywords USER, GROUP, or ROLE is specified, then for all rows for the specified object in the SYSCAT.ROLEAUTH catalog view where the grantee is *authorization-name*:
  - If GRANTEETYPE is 'U', USER is assumed.
  - If GRANTEETYPE is 'G', GROUP is assumed.
  - If GRANTEETYPE is 'R', ROLE is assumed.
  - If GRANTEETYPE does not have the same value, an error is returned (SQLSTATE 56092).
- The *role-name* must not identify a role, or a role that contains *role-name*, if the role has either EXECUTE privilege on a routine or USAGE privilege on a sequence, and an SQL object other than a package is dependent on the routine or sequence (SQLSTATE 42893). The owner of the SQL object is either *authorization-name* or any user that is a member of *authorization-name*, where *authorization-name* is a role.

## Notes

- If a role is revoked from an *authorization-name* or from PUBLIC, all privileges that the role held are no longer available to the *authorization-name* or to PUBLIC through that role.
- Revoking a role does not necessarily revoke the ability to perform a particular action by way of a privilege that was granted to that role. A user might still be able to proceed if other privileges are held by PUBLIC, by a group to which the user belongs, by another role granted to the user, or if the user has a higher level authority, such as DBADM.

## Examples

*Example 1:* Revoke the role INTERN from the role DOCTOR and the role DOCTOR from the role SPECIALIST.

```
REVOKE ROLE INTERN FROM ROLE DOCTOR
```

```
REVOKE ROLE DOCTOR FROM ROLE SPECIALIST
```

*Example 2:* Revoke the role INTERN from PUBLIC.

## REVOKE ROLE INTERN FROM PUBLIC

Example 3: Revoke the role SPECIALIST from user BOB and group TORONTO.

```
REVOKE ROLE SPECIALIST FROM USER BOB, GROUP TORONTO BY ALL
```

## REVOKE (routine privileges)

This form of the REVOKE statement revokes privileges on a routine (function, method, or procedure) that is not defined in a module.

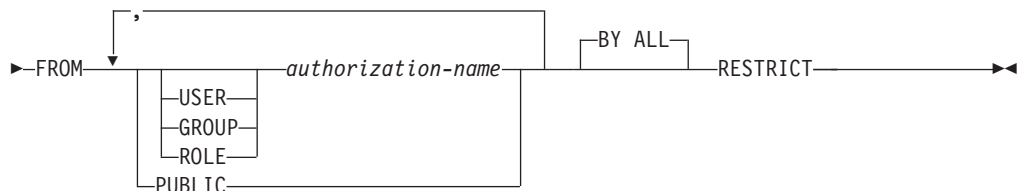
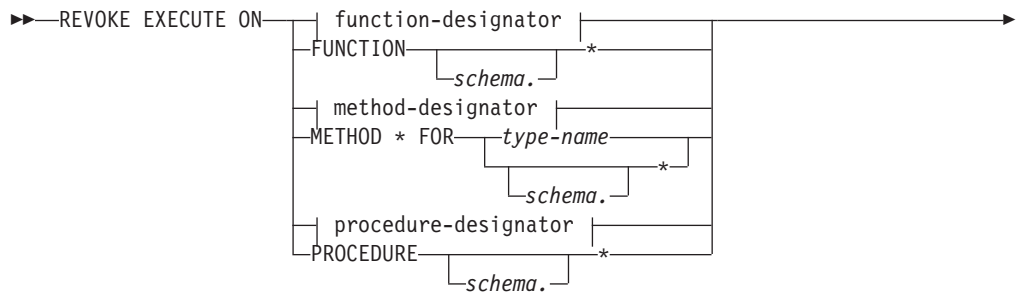
### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

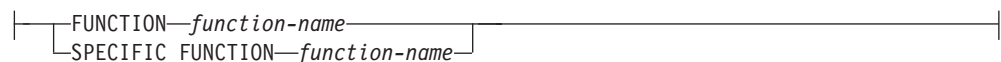
### Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

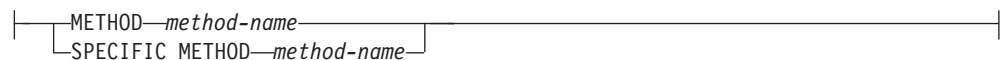
### Syntax



### function-designator:

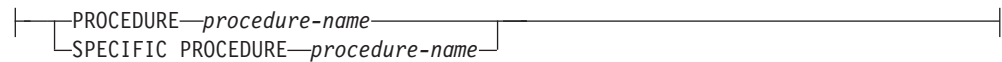


### method-designator:





## procedure-designator:



## Description

### EXECUTE

Revokes the privilege to run the identified user-defined function, method, or procedure.

#### *function-designator*

Uniquely identifies the function from which the privilege is revoked. For more information, see “Function, method, and procedure designators” on page 497.

### FUNCTION *schema.\**

Identifies the explicit grant for all the existing and future functions in the schema. Revoking the *schema.\** privilege does not revoke any privileges that were granted on a specific function. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

#### *method-designator*

Uniquely identifies the method from which the privilege is revoked. For more information, see “Function, method, and procedure designators” on page 497.

### METHOD \*

Identifies the explicit grant for all the existing and future methods for the type *type-name*. Revoking the \* privilege does not revoke any privileges that were granted on a specific method.

#### FOR *type-name*

Names the type in which the specified method is found. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the value of the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names. An asterisk (\*) can be used in place of *type-name* to identify the explicit grant on all existing and future methods for all existing and future types in the schema. Revoking the privilege using an asterisk for method and *type-name* does not revoke any privileges that were granted on a specific method or on all methods for a specific type.

#### *procedure-designator*

Uniquely identifies the procedure from which the privilege is revoked. For more information, see “Function, method, and procedure designators” on page 497.

### PROCEDURE *schema.\**

Identifies the explicit grant for all the existing and future procedures in the schema. Revoking the *schema.\** privilege does not revoke any privileges that were granted on a specific procedure. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

**FROM**

Specifies from whom the EXECUTE privilege is revoked.

**USER**

Specifies that the *authorization-name* identifies a user.

**GROUP**

Specifies that the *authorization-name* identifies a group name.

**ROLE**

Specifies that the *authorization-name* identifies a role name.

*authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

**PUBLIC**

Revokes the EXECUTE privilege from PUBLIC.

**BY ALL**

Revokes the EXECUTE privilege from all named users who were explicitly granted the privilege, regardless of who granted it. This is the default behavior.

**RESTRICT**

Specifies that the EXECUTE privilege cannot be revoked if both of the following are true (SQLSTATE 42893):

- The specified routine is used in a view, trigger, constraint, index extension, SQL function, SQL method, transform group, or is referenced as the SOURCE of a sourced function.
- The loss of the EXECUTE privilege would cause the owner of the view, trigger, constraint, index extension, SQL function, SQL method, transform group, or sourced function to no longer be able to execute the specified routine.

**Rules**

- It is not possible to revoke the EXECUTE privilege on a function or method defined with schema 'SYSIBM' or 'SYSFUN' (SQLSTATE 42832).
- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.ROUTINEAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

**Notes**

- If a package depends on a routine (function, method, or procedure), and the EXECUTE privilege on that routine is revoked from PUBLIC, a user, or a role, the package becomes inoperative if the routine is a function or a method, and the package becomes invalid if the routine is a procedure, unless the package owner still holds the EXECUTE privilege on the routine. The package owner can still hold the EXECUTE privilege if:
  - The package owner was explicitly granted the EXECUTE privilege

- The package owner is a member of a role that holds the EXECUTE privilege
- The EXECUTE privilege was granted to PUBLIC

Because group privileges are not considered for static packages, the package becomes inoperative (in the case of a function or a method) or invalid (in the case of a procedure) even if a group to which the package owner belongs holds the EXECUTE privilege.

## Examples

*Example 1:* Revoke the EXECUTE privilege on function CALC\_SALARY from user JONES. Assume that there is only one function in the schema with function name CALC\_SALARY.

```
REVOKE EXECUTE ON FUNCTION CALC_SALARY FROM JONES RESTRICT
```

*Example 2:* Revoke the EXECUTE privilege on procedure VACATION\_ACCR from all users at the current server.

```
REVOKE EXECUTE ON PROCEDURE VACATION_ACCR FROM PUBLIC RESTRICT
```

*Example 3:* Revoke the EXECUTE privilege on function NEW\_DEPT\_HIRES from HR (Human Resources). The function has two input parameters of type INTEGER and CHAR(10), respectively. Assume that the schema has more than one function named NEW\_DEPT\_HIRES.

```
REVOKE EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10))
FROM HR RESTRICT
```

*Example 4:* Revoke the EXECUTE privilege on method SET\_SALARY for type EMPLOYEE from user Jones.

```
REVOKE EXECUTE ON METHOD SET_SALARY FOR EMPLOYEE FROM JONES RESTRICT
```

---

## REVOKE (Schema Privileges)

This form of the REVOKE statement revokes the privileges on a schema.

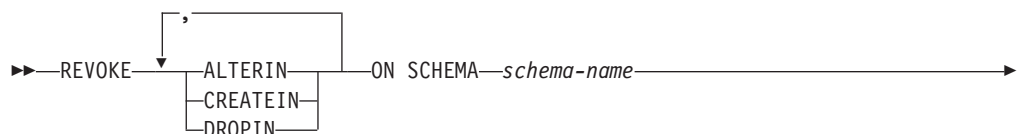
### Invocation

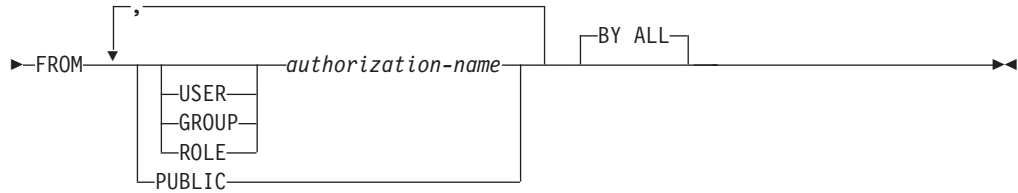
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

### Syntax





## Description

### ALTERIN

Revokes the privilege to alter or comment on objects in the schema.

### CREATEIN

Revokes the privilege to create objects in the schema.

### DROPIN

Revokes the privilege to drop objects in the schema.

### ON SCHEMA *schema-name*

Specifies the name of the schema on which privileges are to be revoked.

### FROM

Indicates from whom the privileges are revoked.

#### USER

Specifies that the *authorization-name* identifies a user.

#### GROUP

Specifies that the *authorization-name* identifies a group name.

#### ROLE

Specifies that the *authorization-name* identifies a role name.

#### *authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

#### PUBLIC

Revokes the privileges from PUBLIC.

### BY ALL

Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.SCHEMAAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

## Notes

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds a higher level authority such as DBADM.

## Examples

*Example 1:* Given that USER4 is only a user and not a group, revoke the privilege to create objects in schema DEPTIDX from the user USER4.

```
REVOKE CREATEIN ON SCHEMA DEPTIDX FROM USER4
```

*Example 2:* Revoke the privilege to drop objects in schema LUNCH from the user CHEF and the group WAITERS.

```
REVOKE DROPIN ON SCHEMA LUNCH
FROM USER CHEF, GROUP WAITERS
```

---

## REVOKE (Security Label)

This form of the REVOKE statement revokes a label-based access control (LBAC) security label.

### Invocation

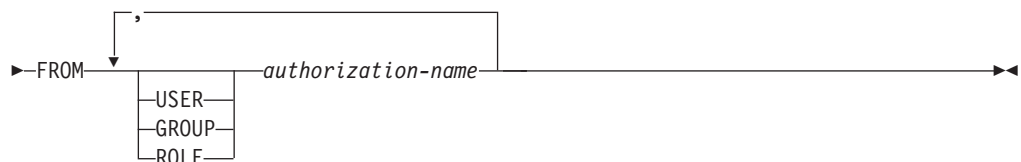
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

### Syntax

```
►►—REVOKE SECURITY LABEL—security-label-name—————►
```



### Description

**SECURITY LABEL** *security-label-name*

Revokes the security label *security-label-name*. The name must be qualified with a security policy (SQLSTATE 42704) and must identify a security label that exists at the current server (SQLSTATE 42704), and that is held by *authorization-name* (SQLSTATE 42504).

## FROM

Specifies from whom the specified security label is revoked.

## USER

Specifies that the *authorization-name* identifies a user.

## GROUP

Specifies that the *authorization-name* identifies a group name.

## ROLE

Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

*authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles.

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.SECURITYLABELACCESS catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

## Examples

*Example 1:* Revoke the security label EMPLOYEESECLABEL, which is part of the security policy DATA\_ACCESS, from user WALID.

```
REVOKE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABEL
FROM USER WALID
```

---

## REVOKE (Sequence Privileges)

This form of the REVOKE statement revokes privileges on a sequence.

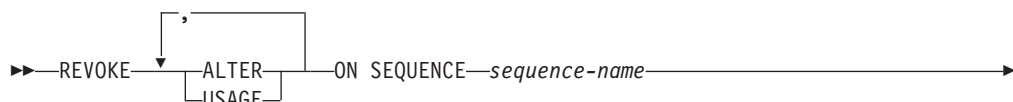
### Invocation

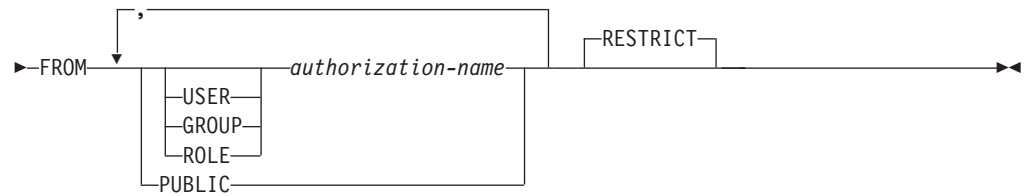
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

### Syntax





## Description

### ALTER

Revokes the privilege to change the properties of a sequence or to restart sequence number generation using the ALTER SEQUENCE statement.

### USAGE

Revokes the privilege to reference a sequence using *nextval-expression* or *prevval-expression*.

### ON SEQUENCE *sequence-name*

Identifies the sequence on which the specified privileges are to be revoked. The sequence name, including an implicit or explicit schema qualifier, must uniquely identify an existing sequence at the current server. If no sequence by this name exists, an error is returned (SQLSTATE 42704).

### FROM

Specifies from whom the privileges are revoked.

#### USER

Specifies that the *authorization-name* identifies a user.

#### GROUP

Specifies that the *authorization-name* identifies a group name.

#### ROLE

Specifies that the *authorization-name* identifies a role name.

#### *authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

#### PUBLIC

Revokes the specified privileges from PUBLIC.

### RESTRICT

This optional keyword indicates that the statement will fail if any objects depend on the privilege being revoked.

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.SEQUENCEAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).



## Notes

- Revoking a privilege on a sequence from the authorization ID under which a package was bound will cause the package to become invalid if the authorization ID does not continue to hold the privilege on the sequence through different means; for example, through membership in a role that holds the privilege.
- Revoking a specific privilege does not necessarily remove the ability to perform an action. A user can proceed if other privileges are held by PUBLIC or by a group to which the user belongs, or if the user has a higher level of authority, such as DBADM.

## Examples

*Example 1:* Revoke the USAGE privilege on a sequence called GENERATE\_ID from user ENGLES. There is one row in the SYSCAT.SEQUENCEAUTH catalog view for this sequence and grantee, and the GRANTEETYPE value is U.

```
REVOKE USAGE ON SEQUENCE GENERATE_ID FROM ENGLES
```

*Example 2:* Revoke alter privileges on sequence GENERATE\_ID that were previously granted to all local users. (Grants to specific users are not affected.)

```
REVOKE ALTER ON SEQUENCE GENERATE_ID FROM PUBLIC
```

*Example 3:* Revoke all privileges on sequence GENERATE\_ID from users PELLOW and MLI, and from group PLANNERS.

```
REVOKE ALTER, USAGE ON SEQUENCE GENERATE_ID
FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

---

## REVOKE (Server Privileges)

This form of the REVOKE statement revokes the privilege to access and use a specified data source in pass-through mode.

### Invocation

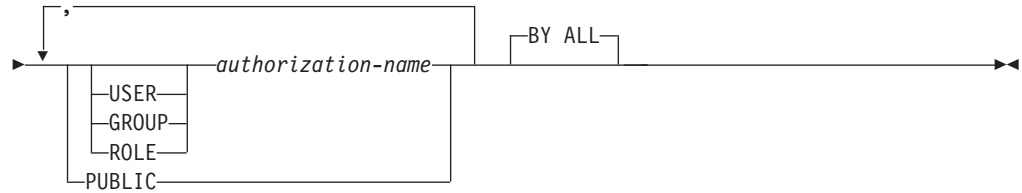
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

### Syntax

```
►►—REVOKE PASSTHRU ON SERVER—server-name—FROM—►
```



## Description

### SERVER *server-name*

Names the data source for which the privilege to use in pass-through mode is being revoked. *server-name* must identify a data source that is described in the catalog.

### FROM

Specifies from whom the privilege is revoked.

#### USER

Specifies that the *authorization-name* identifies a user.

#### GROUP

Specifies that the *authorization-name* identifies a group name.

#### ROLE

Specifies that the *authorization-name* identifies a role name.

#### *authorization-name,...*

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

#### PUBLIC

Revokes from PUBLIC the privilege to pass through to *server-name*.

### BY ALL

Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.PASSTHROUGH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

## Examples

*Example 1:* Revoke USER6's privilege to pass through to data source MOUNTAIN.

```
REVOKE PASSTHRU ON SERVER MOUNTAIN FROM USER USER6
```

*Example 2:* Revoke group D024's privilege to pass through to data source EASTWING.

## REVOKE PASSTHRU ON SERVER EASTWING FROM GROUP D024

The members of group D024 will no longer be able to use their group ID to pass through to EASTWING. But if any members have the privilege to pass through to EASTWING under their own user IDs, they will retain this privilege.

---

## REVOKE (SETSESSIONUSER Privilege)

This form of the REVOKE statement revokes one or more SETSESSIONUSER privileges from one or more authorization IDs.

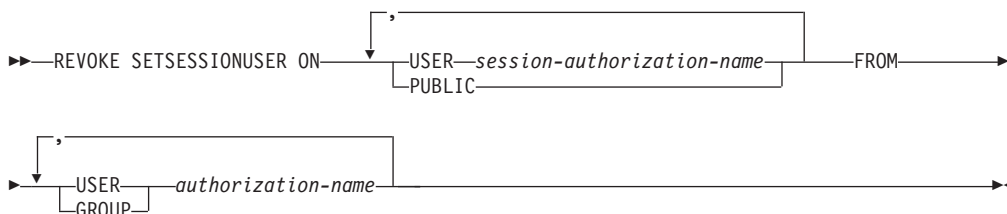
### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

### Syntax



### Description

#### SETSESSIONUSER ON

Revokes the privilege to assume the identity of a new authorization ID.

#### USER *session-authorization-name*

Specifies the authorization ID that the *authorization-name* is able to assume, using the SET SESSION AUTHORIZATION statement. The *session-authorization-name* must identify a user that the *authorization-name* can assume, not a group (SQLSTATE 42504).

#### PUBLIC

Specifies that all privileges to set the session authorization will be revoked.

#### FROM

Specifies from whom the privilege is revoked.

#### USER

Specifies that the *authorization-name* identifies a user.

#### GROUP

Specifies that the *authorization-name* identifies a group name.

*authorization-name*,...

Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

## Examples

*Example 1:* User PAUL holds the privilege to set the session authorization to WALID and therefore to execute SQL statements as user WALID. The following statement revokes that privilege.

```
REVOKE SETSESSIONUSER ON USER WALID
FROM USER PAUL
```

*Example 2:* User GUYLAINE holds the privilege to set the session authorization to BOBBY, RICK, or KEVIN and therefore to execute SQL statements as BOBBY, RICK, or KEVIN. The following statement revokes the privilege to use two of those authorization IDs. After this statement executes, GUYLAINE will only be able to set the session authorization to KEVIN.

```
REVOKE SETSESSIONUSER ON USER BOBBY, USER RICK
FROM USER GUYLAINE
```

*Example 3:* The group ACCTG and user WALID can set session authorization to any authorization ID. The following statement revokes that privilege from both ACCTG and WALID.

```
REVOKE SETSESSIONUSER ON PUBLIC
FROM USER WALID, GROUP ACCTG
```

---

## REVOKE (Table Space Privileges)

This form of the REVOKE statement revokes the USE privilege on a table space.

### Invocation

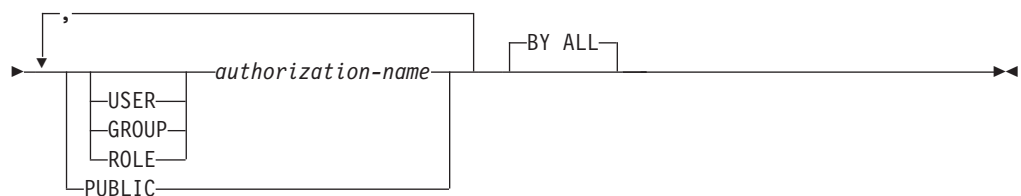
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL, SECADM, SYSCTRL, or SYSADM authority.

### Syntax

```
►►—REVOKE USE OF TABLESPACE—tablespace-name—FROM—►►
```



## Description

### USE

Revokes the privilege to specify or default to the table space when creating a table.

### OF TABLESPACE *tablespace-name*

Specifies the table space on which the USE privilege is to be revoked. The table space cannot be SYSCATSPACE (SQLSTATE 42838) or a SYSTEM TEMPORARY table space (SQLSTATE 42809).

### FROM

Indicates from whom the USE privilege is revoked.

### USER

Specifies that the *authorization-name* identifies a user.

### GROUP

Specifies that the *authorization-name* identifies a group name.

### ROLE

Specifies that the *authorization-name* identifies a role name.

### *authorization-name*

Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

### PUBLIC

Revokes the USE privilege from PUBLIC.

### BY ALL

Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

## Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.TBSPACEAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

## Notes

- Revoking the USE privilege does not necessarily revoke the ability to create tables in that table space. A user may still be able to create tables in that table space if the USE privilege is held by PUBLIC or a group, or if the user has a higher level authority, such as DBADM.

## Examples

*Example 1:* Revoke the privilege to create tables in table space PLANS from the user BOBBY.

```
REVOKE USE OF TABLESPACE PLANS FROM USER BOBBY
```

## REVOKE (Table, View, or Nickname Privileges)

This form of the REVOKE statement revokes privileges on a table, view, or nickname.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

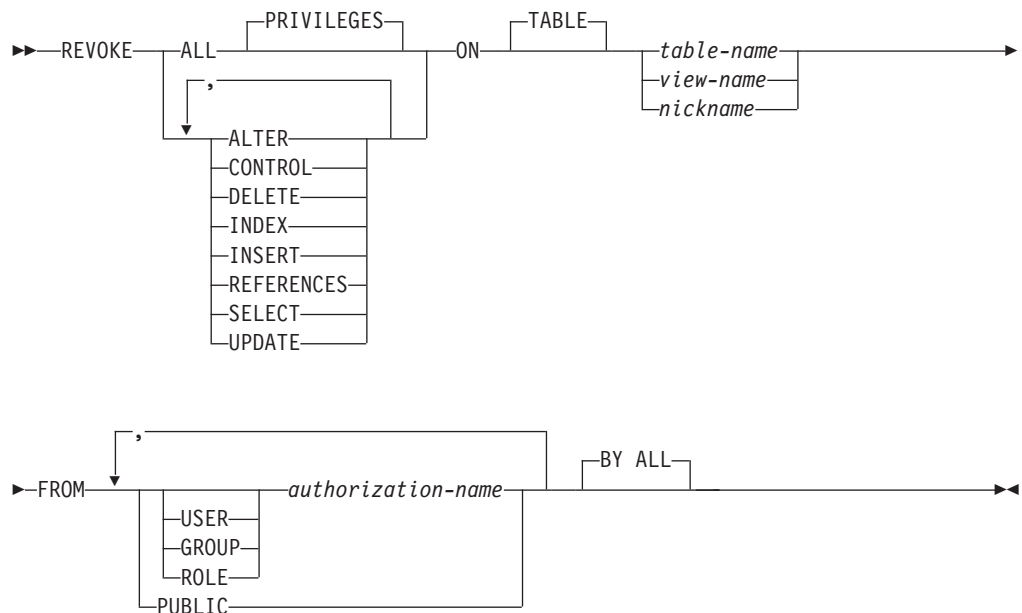
### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced table, view, or nickname
- ACCESSCTRL or SECADM authority

ACCESSCTRL or SECADM authority is required to revoke the CONTROL privilege, or to revoke privileges on catalog tables and views.

### Syntax



### Description

#### ALL or ALL PRIVILEGES

Revokes all privileges (except CONTROL) held by an authorization-name for the specified tables, views, or nicknames.

If ALL is not used, one or more of the keywords listed below must be used. Each keyword revokes the privilege described, but only as it applies to the tables, views, or nicknames named in the ON clause. The same keyword must not be specified more than once.

**ALTER**

Revokes the privilege to add columns to the base table definition; create or drop a primary key or unique constraint on the table; create or drop a foreign key on the table; add/change a comment on the table, view, or nickname; create or drop a check constraint; create a trigger; add, reset, or drop a column option for a nickname; or, change nickname column names or data types.

**CONTROL**

Revokes the ability to drop the table, view, or nickname, and the ability to execute the RUNSTATS utility on the table and indexes.

Revoking CONTROL privilege from an *authorization-name* does not revoke other privileges granted to the user on that object.

**DELETE**

Revokes the privilege to delete rows from the table, updatable view, or nickname.

**INDEX**

Revokes the privilege to create an index on the table or an index specification on the nickname. The creator of an index or index specification automatically has the CONTROL privilege over the index or index specification (authorizing the creator to drop the index or index specification). In addition, the creator retains this privilege even if the INDEX privilege is revoked.

**INSERT**

Revokes the privileges to insert rows into the table, updatable view, or nickname, and to run the IMPORT utility.

**REFERENCES**

Revokes the privilege to create or drop a foreign key referencing the table as the parent. Any column level REFERENCES privileges are also revoked.

**SELECT**

Revokes the privilege to retrieve rows from the table or view, to create a view on a table, and to run the EXPORT utility against the table or view.

Revoking SELECT privilege may cause some views to be marked inoperative. (For information on inoperative views, see "CREATE VIEW".)

**UPDATE**

Revokes the privilege to update rows in the table, updatable view, or nickname. Any column level UPDATE privileges are also revoked.

**ON TABLE** *table-name* **or** *view-name* **or** *nickname*

Specifies the table, view, or nickname on which privileges are to be revoked. The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

**FROM**

Indicates from whom the privileges are revoked.

**USER**

Specifies that the *authorization-name* identifies a user.

**GROUP**

Specifies that the *authorization-name* identifies a group name.

**ROLE**

Specifies that the *authorization-name* identifies a role name.

*authorization-name*,...

Lists the authorization IDs of one or more users, groups, or roles.



The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

### **PUBLIC**

Revokes the privileges from PUBLIC.

### **BY ALL**

Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

### **Rules**

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.TABAUTH and SYSCAT.COLAUTH catalog views where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

### **Notes**

- If a privilege is revoked from the *authorization-name* that is the owner of the view (as recorded in the OWNER column in SYSCAT.VIEWS), that privilege is also revoked from any dependent views.
- If the owner of the view loses a SELECT privilege on some object on which the view definition depends (or an object upon which the view definition depends is dropped, or made inoperative in the case of another view), the view will be made inoperative.

However, if a user who holds ACCESSCTRL or SECADM authority explicitly revokes all privileges on the view from the owner, then the record of the OWNER will not appear in SYSCAT.TABAUTH but nothing will happen to the view - it remains operative.

- Privileges on inoperative views cannot be revoked.
- A package might become invalid when the authorization ID under which the package was bound loses a privilege on an object on which the package depends. The privilege can be lost in one of the following ways:
  - The privilege is revoked from the authorization ID
  - The privilege is revoked from a role of which the authorization ID is a member
  - The privilege is revoked from PUBLIC

A package remains invalid until a bind or rebind operation on the application is successfully executed, or the application is executed and the database manager successfully rebinds the application (using information stored in the catalogs). Packages marked invalid due to a revoke may be successfully rebound without any additional grants.

For example, if a package owned by USER1 contains a SELECT from table T1, and the SELECT privilege on table T1 is revoked from USER1, the package will be marked invalid. If SELECT authority is granted again, or if the user holds DBADM authority, the package is successfully rebound when executed.

Another example is a package owned by USER1, who is a member of role R1. The package contains a SELECT from table T1, and the SELECT privilege on table T1 is revoked from role R1. The package will be marked invalid, assuming USER1 does not hold the SELECT privilege on table T1 by other means.

- Packages, triggers or views that include the use of OUTER(Z) in the FROM clause, are dependent on having SELECT privilege on every subtable or subview of Z. Similarly, packages, triggers, or views that include the use of Deref(Y) where Y is a reference type with a target table or view Z, are dependent on having SELECT privilege on every subtable or subview of Z. Such packages might become invalid, and such triggers or views made inoperative when the authorization ID under which the packages were bound, or the owner of the triggers or views loses the SELECT privilege. The SELECT privilege can be lost in one of the following ways:
  - SELECT privilege is revoked from the authorization ID
  - SELECT privilege is revoked from a role of which the authorization ID is a member
  - SELECT privilege is revoked from PUBLIC
- Table, view, or nickname privileges cannot be revoked from an *authorization-name* with CONTROL on the object without also revoking the CONTROL privilege (SQLSTATE 42504).
- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds privileges such as ALTERIN on the schema of a table or a view.
- If the owner of the materialized query table loses a SELECT privilege on a table on which the materialized query table definition depends (or a table upon which the materialized query table definition depends is dropped), the materialized query table will be dropped.

However, if a user who holds SECADM or ACCESSCTRL authority explicitly revokes all privileges on the materialized query table from the owner, then the record in SYSTABAUTH for the OWNER will be deleted, but nothing will happen to the materialized query table - it remains operative.

- Revoking nickname privileges has no affect on data source object (table or view) privileges.
- Revoking the SELECT privilege for a table or view that is directly or indirectly referenced in an SQL function or method body may fail if the SQL function or method body cannot be dropped because some other object is dependent on it (SQLSTATE 42893).
- Revoking the SELECT privilege causes an SQL function or method body to be dropped when:
  - The owner of the SQL function or method body loses the SELECT privilege on some object on which the SQL function or method body definition depends; note that the privilege can be lost because of a revoke from PUBLIC or from a role of which the owner is a member
  - An object on which the SQL function or method body definition depends is dropped

However, the revoke fails if another object depends on the function or method (SQLSTATE 42893).

## Examples

*Example 1:* Revoke SELECT privilege on table EMPLOYEE from user ENGLES. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is U.

```
REVOKE SELECT
ON TABLE EMPLOYEE
FROM ENGLES
```

*Example 2:* Revoke update privileges on table EMPLOYEE previously granted to all local users. Note that grants to specific users are not affected.

```
REVOKE UPDATE
ON EMPLOYEE
FROM PUBLIC
```

*Example 3:* Revoke all privileges on table EMPLOYEE from users PELLOW and MLI and from group PLANNERS.

```
REVOKE ALL
ON EMPLOYEE
FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

*Example 4:* Revoke SELECT privilege on table CORPDATA.EMPLOYEE from a user named JOHN. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is U.

```
REVOKE SELECT
ON CORPDATA.EMPLOYEE FROM JOHN
```

or

```
REVOKE SELECT
ON CORPDATA.EMPLOYEE FROM USER JOHN
```

Note that an attempt to revoke the privilege from GROUP JOHN would result in an error, since the privilege was not previously granted to GROUP JOHN.

*Example 5:* Revoke SELECT privilege on table CORPDATA.EMPLOYEE from a group named JOHN. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is G.

```
REVOKE SELECT
ON CORPDATA.EMPLOYEE FROM JOHN
```

or

```
REVOKE SELECT
ON CORPDATA.EMPLOYEE FROM GROUP JOHN
```

*Example 6:* Revoke user SHAWN's privilege to create an index specification on nickname ORAREM1.

```
REVOKE INDEX
ON ORAREM1 FROM USER SHAWN
```

---

## SET ROLE

The SET ROLE statement verifies that the authorization ID of the session is a member of a specific role. An authorization ID acquires membership in a role when the role is granted to the authorization ID, or to a group or role in which the authorization ID is a member.

## Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

## Authorization

None required.

## Syntax

```
▶▶ SET ROLE role-name ▶▶
```

## Description

*role-name*

Specifies a role in whose membership the authorization ID of the session is to be verified. The *role-name* must identify an existing role at the current server (SQLSTATE 42704). If the authorization ID of the session is not a member of *role-name*, an error is returned (SQLSTATE 42501).

## Notes

- All roles that have been granted to an authorization ID are used for authorization checking. The SET ROLE statement does not affect which roles are used for this authorization checking. Use the GRANT ROLE and REVOKE ROLE statements to change the roles in which an authorization ID has membership.

## Examples

*Example 1:* User WALID has been granted the role EDITOR, but not the role AUTHOR. Verify that WALID is a member of the EDITOR role.

```
SET ROLE EDITOR
```

*Example 2:* Verify that WALID is not a member of the AUTHOR role. The following statement returns an error (SQLSTATE 42501).

```
SET ROLE AUTHOR
```

---

## UPDATE

The UPDATE statement updates the values of specified columns in rows of a table, view or nickname, or the underlying tables, nicknames, or views of the specified fullselect. Updating a row of a view updates a row of its base table, if no INSTEAD OF trigger is defined for the update operation on this view. If such a trigger is defined, the trigger will be executed instead. Updating a row using a nickname updates a row in the data source object to which the nickname refers.

The forms of this statement are:

- The *Searched* UPDATE form is used to update one or more rows (optionally determined by a search condition).

- The *Positioned* UPDATE form is used to update exactly one row (as determined by the current position of a cursor).

## Invocation

An UPDATE statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

## Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- UPDATE privilege on the target table, view, or nickname
- UPDATE privilege on each of the columns that are to be updated
- CONTROL privilege on the target table, view, or nickname
- DATAACCESS authority

If a *row-fullselect* is included in the assignment, the privileges held by the authorization ID of the statement must include at least one of the following for each referenced table, view, or nickname:

- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

For each table, view, or nickname referenced by a subquery, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

If the package used to process the statement is precompiled with SQL92 rules (option LANGLEVEL with a value of SQL92E or MIA), and the searched form of an UPDATE statement includes a reference to a column of the table, view, or nickname in the right hand side of the *assignment-clause*, or anywhere in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

If the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

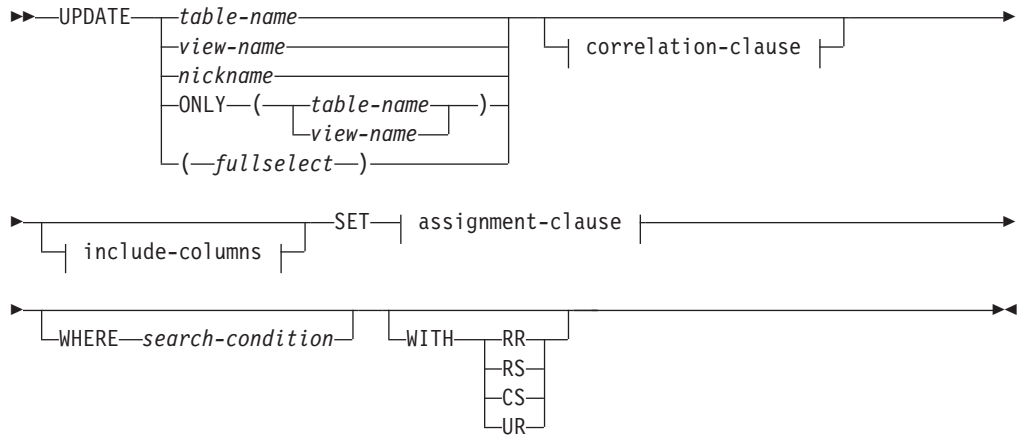
GROUP privileges are not checked for static UPDATE statements.

If the target of the update operation is a nickname, privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges that are required for the operation on the object at the data

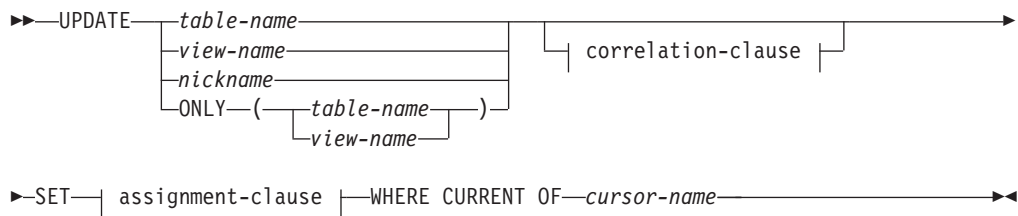
source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

## Syntax

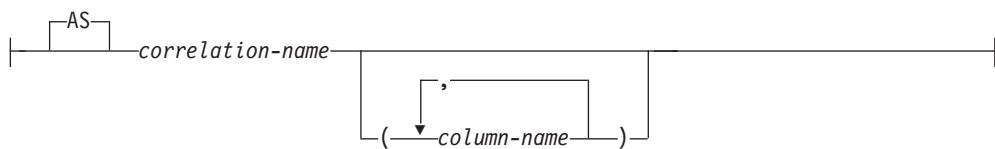
### searched-update:



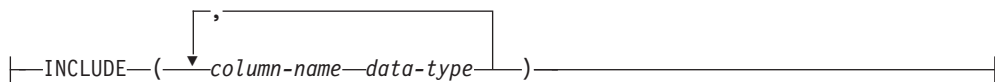
### positioned-update:



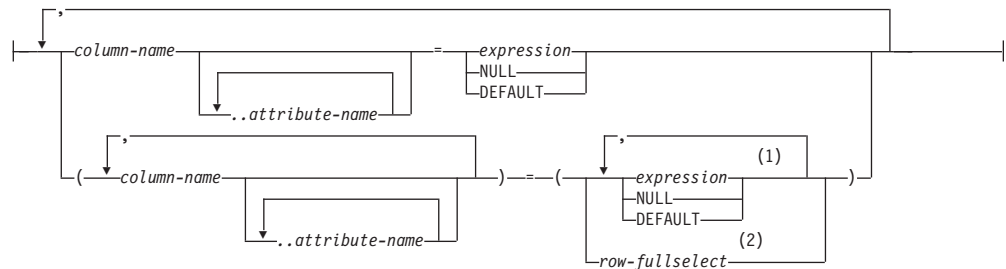
### correlation-clause:



### include-columns:



### assignment-clause:



**Notes:**

- 1 The number of expressions, NULLs and DEFAULTs must match the number of column names.
- 2 The number of columns in the select list must match the number of column names.

**Description**

*table-name, view-name, nickname, or (fullselect)*

Identifies the object of the update operation. The name must identify a table, view, or nickname described in the catalog, but not a catalog table, a view of a catalog table (unless it is one of the updatable SYSSTAT views), a system-maintained materialized query table, or a read-only view that has no INSTEAD OF trigger defined for its update operations.

If *table-name* is a typed table, rows of the table or any of its proper subtables may get updated by the statement. Only the columns of the specified table may be set or referenced in the WHERE clause. For a positioned UPDATE, the associated cursor must also have specified the same table, view or nickname in the FROM clause without using ONLY.

If the object of the update operation is a fullselect, the fullselect must be updatable, as defined in the “Updatable views” Notes item in the description of the CREATE VIEW statement.

**ONLY (*table-name*)**

Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be updated by the statement. For a positioned UPDATE, the associated cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

**ONLY (*view-name*)**

Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews cannot be updated by the statement. For a positioned UPDATE, the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

**correlation-clause**

Can be used within *search-condition* or *assignment-clause* to designate a table, view, nickname, or fullselect. For a description of *correlation-clause*, see “table-reference” in the description of “Subselect”.

*include-columns*

Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the UPDATE



statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

### INCLUDE

Specifies a list of columns to be included in the intermediate result table of the UPDATE statement.

#### *column-name*

Specifies a column of the intermediate result table of the UPDATE statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

#### *data-type*

Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

### SET

Introduces the assignment of values to column names.

#### *assignment-clause*

##### *column-name*

Identifies a column to be updated. The *column-name* must identify an updatable column of the specified table, view, or nickname, or identify an INCLUDE column. The object ID column of a typed table is not updatable (SQLSTATE 428DZ). A column must not be specified more than once, unless it is followed by *..attribute-name* (SQLSTATE 42701).

If it specifies an INCLUDE column, the column name cannot be qualified.

For a Positioned UPDATE:

- If the *update-clause* was specified in the *select-statement* of the cursor, each column name in the *assignment-clause* must also appear in the *update-clause*.
- If the *update-clause* was not specified in the *select-statement* of the cursor and LANGLEVEL MIA or SQL92E was specified when the application was precompiled, the name of any updatable column may be specified.
- If the *update-clause* was not specified in the *select-statement* of the cursor and LANGLEVEL SAA1 was specified either explicitly or by default when the application was precompiled, no columns may be updated.

##### *..attribute-name*

Specifies the attribute of a structured type that is set (referred to as an *attribute assignment*). The *column-name* specified must be defined with a user-defined structured type (SQLSTATE 428DP). The attribute-name must be an attribute of the structured type of *column-name* (SQLSTATE 42703). An assignment that does not involve the *..attribute-name* clause is referred to as a *conventional assignment*.

##### *expression*

Indicates the new value of the column. The expression is any expression of the type described in "Expressions". The expression cannot include an aggregate function except when it occurs within a scalar fullselect (SQLSTATE 42903).

An *expression* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated.

An expression cannot contain references to an INCLUDE column.

## NULL

Specifies the null value and can only be specified for nullable columns (SQLSTATE 23502). NULL cannot be the value in an attribute assignment (SQLSTATE 429B9) unless it is specifically cast to the data type of the attribute.

## DEFAULT

Specifies that the default value should be used based on how the corresponding column is defined in the table. The value that is inserted depends on how the column was defined.

- If the column was defined as a generated column based on an expression, the column value will be generated by the system, based on the expression.
- If the column was defined using the IDENTITY clause, the value is generated by the database manager.
- If the column was defined using the WITH DEFAULT clause, the value is set to the default defined for the column (see *default-clause* in “ALTER TABLE”).
- If the column was defined using the NOT NULL clause and the GENERATED clause was not used, or the WITH DEFAULT clause was not used, or DEFAULT NULL was used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).
- If the column was defined using the ROW CHANGE TIMESTAMP clause, the value is generated by the database manager.

The only value that a generated column defined with the GENERATED ALWAYS clause can be set to is DEFAULT (SQLSTATE 428C9).

The DEFAULT keyword cannot be used as the value in an attribute assignment (SQLSTATE 429B9).

The DEFAULT keyword cannot be used as the value in an assignment for update on a nickname where the data source does not support DEFAULT syntax.

### *row-fullselect*

A fullselect that returns a single row with the number of columns corresponding to the number of *column-names* specified for assignment. The values are assigned to each corresponding *column-name*. If the result of the *row-fullselect* is no rows, then null values are assigned.

A *row-fullselect* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated. An error is returned if there is more than one row in the result (SQLSTATE 21000).

## WHERE

Introduces a condition that indicates what rows are updated. You can omit the clause, give a search condition, or name a cursor. If the clause is omitted, all rows of the table, view or nickname are updated.

### *search-condition*

Each *column-name* in the search condition, other than in a subquery, must name a column of the table, view or nickname. When the search condition

includes a subquery in which the same table is the base object of both the UPDATE and the subquery, the subquery is completely evaluated before any rows are updated.

The search-condition is applied to each row of the table, view or nickname and the updated rows are those for which the result of the search-condition is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a row, and the results used in applying the search condition. In actuality, a subquery with no correlated references is executed only once, whereas a subquery with a correlated reference may have to be executed once for each row.

#### **CURRENT OF** *cursor-name*

Identifies the cursor to be used in the update operation. The *cursor-name* must identify a declared cursor, explained in "DECLARE CURSOR". The DECLARE CURSOR statement must precede the UPDATE statement in the program.

The specified table, view, or nickname must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see "DECLARE CURSOR".)

When the UPDATE statement is executed, the cursor must be positioned on a row; that row is updated.

This form of UPDATE cannot be used (SQLSTATE 42828) if the cursor references:

- A view on which an INSTEAD OF UPDATE trigger is defined
- A view that includes an OLAP function in the select list of the fullselect that defines the view
- A view that is defined, either directly or indirectly, using the WITH ROW MOVEMENT clause

#### **WITH**

Specifies the isolation level at which the UPDATE statement is executed.

##### **RR**

Repeatable Read

##### **RS**

Read Stability

##### **CS**

Cursor Stability

##### **UR**

Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. The WITH clause has no effect on nicknames, which always use the default isolation level of the statement.

#### **Rules**

- **Triggers:** UPDATE statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the update values. If an update operation on a view causes an INSTEAD OF trigger to fire, validity, referential integrity, and constraints will be checked

against the updates that are performed in the trigger, and not against the view that caused the trigger to fire, or its underlying tables.

- **Assignment:** Update values are assigned to columns according to specific assignment rules.
- **Validity:** The updated row must conform to any constraints imposed on the table (or on the base table of the view) by any unique index on an updated column.

If a view is used that is not defined using WITH CHECK OPTION, rows can be changed so that they no longer conform to the definition of the view. Such rows are updated in the base table of the view and no longer appear in the view.

If a view is used that is defined using WITH CHECK OPTION, an updated row must conform to the definition of the view. For an explanation of the rules governing this situation, see “CREATE VIEW”.

- **Check constraint:** Update value must satisfy the check-conditions of the check constraints defined on the table.

An UPDATE to a table with check constraints defined has the constraint conditions for each column updated evaluated once for each row that is updated. When processing an UPDATE statement, only the check constraints referring to the updated columns are checked.

- **Referential integrity:** The value of the parent unique keys cannot be changed if the update rule is RESTRICT and there are one or more dependent rows. However, if the update rule is NO ACTION, parent unique keys can be updated as long as every child has a parent key by the time the update statement completes. A non-null update value of a foreign key must be equal to a value of the primary key of the parent table of the relationship.
- **XML values:** When an XML column value is updated, the new value must be a well-formed XML document (SQLSTATE 2200M).
- **Security policy:** If the identified table or the base table of the identified view is protected with a security policy, the session authorization ID must have the label-based access control (LBAC) credentials that allow:
  - Write access to all protected columns that are being updated (SQLSTATE 42512)
  - Write access for any explicit value provided for a DB2SECURITYLABEL column for security policies that were created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option (SQLSTATE 23523)
  - Read and write access to all rows that are being updated (SQLSTATE 42519)The session authorization ID must also have been granted a security label for write access for the security policy if an implicit value is used for a DB2SECURITYLABEL column (SQLSTATE 23523), which can happen when:
  - The DB2SECURITYLABEL column is not included in the list of columns that are to be updated (and so it will be implicitly updated to the security label for write access of the session authorization ID)
  - A value for the DB2SECURITYLABEL column is explicitly provided but the session authorization ID does not have write access for that value, and the security policy is created with the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option

## Notes

- If an update value violates any constraints, or if any other error occurs during the execution of the UPDATE statement, no rows are updated. The order in which multiple rows are updated is undefined.

- An update to a view defined using the WITH ROW MOVEMENT clause could cause a delete operation and an insert operation against the underlying tables of the view. For details, see the description of the CREATE VIEW statement.
- When an UPDATE statement completes execution, the value of SQLERRD(3) in the SQLCA is the number of rows that qualified for the update operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW\_COUNT variable of the GET DIAGNOSTICS statement. The SQLERRD(5) field contains the number of rows inserted, deleted, or updated by all activated triggers.
- Unless appropriate locks already exist, one or more exclusive locks are acquired by the execution of a successful UPDATE statement. Until the locks are released, the updated row can only be accessed by the application process that performed the update (except for applications using the Uncommitted Read isolation level). For further information on locking, see the descriptions of the COMMIT, ROLLBACK, and LOCK TABLE statements.
- When updating the column distribution statistics for a typed table, the subtable that first introduced the column must be specified.
- Multiple attribute assignments on the same structured type column occur in the order specified in the SET clause and, within a parenthesized set clause, in left-to-right order.
- An attribute assignment invokes the mutator method for the attribute of the user-defined structured type. For example, the assignment `st..a1=x` has the same effect as using the mutator method in the assignment `st = st..a1(x)`.
- While a given column may be a target column in only one conventional assignment, a column may be a target column in multiple attribute assignments (but only if it is not also a target column in a conventional assignment).
- When an identity column defined as a distinct type is updated, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column. (There is no casting of the previous value to the source type prior to the computation.)
- To have DB2 generate a value on a SET statement for an identity column, use the DEFAULT keyword:

```
SET NEW.EMPNO = DEFAULT
```

In this example, NEW.EMPNO is defined as an identity column, and the value used to update this column is generated by DB2.

- For more information about consuming values of a generated sequence for an identity column, or about exceeding the maximum value for an identity column, see "INSERT".
- With partitioned tables, an UPDATE WHERE CURRENT OF *cursor-name* operation can move a row from one data partition to another. After this occurs, the cursor is no longer positioned on the row, and no further UPDATE WHERE CURRENT OF *cursor-name* modifications to that row are possible. The next row in the cursor can be fetched, however.
- For a column defined using the ROW CHANGE TIMESTAMP clause, the value is always changed on update of the row. If the column is not specified in the SET list explicitly, the database manager still generates a value for that row. The value is unique for each table partition within the database partition and is set to the approximate timestamp corresponding to the row update.

## Examples

- *Example 1:* Change the job (JOB) of employee number (EMPNO) '000290' in the EMPLOYEE table to 'LABORER'.

```

UPDATE EMPLOYEE
 SET JOB = 'LABORER'
 WHERE EMPNO = '000290'

```

- *Example 2:* Increase the project staffing (PRSTAFF) by 1.5 for all projects that department (DEPTNO) 'D21' is responsible for in the PROJECT table.

```

UPDATE PROJECT
 SET PRSTAFF = PRSTAFF + 1.5
 WHERE DEPTNO = 'D21'

```

- *Example 3:* All the employees except the manager of department (WORKDEPT) 'E21' have been temporarily reassigned. Indicate this by changing their job (JOB) to NULL and their pay (SALARY, BONUS, COMM) values to zero in the EMPLOYEE table.

```

UPDATE EMPLOYEE
 SET JOB=NULL, SALARY=0, BONUS=0, COMM=0
 WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'

```

This statement could also be written as follows.

```

UPDATE EMPLOYEE
 SET (JOB, SALARY, BONUS, COMM) = (NULL, 0, 0, 0)
 WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'

```

- *Example 4:* Update the salary and the commission column of the employee with employee number 000120 to the average of the salary and of the commission of the employees of the updated row's department, respectively.

```

UPDATE (SELECT EMPNO, SALARY, COMM,
 AVG(SALARY) OVER (PARTITION BY WORKDEPT),
 AVG(COMM) OVER (PARTITION BY WORKDEPT)
 FROM EMPLOYEE E) AS E(EMPNO, SALARY, COMM, AVGSAL, AVGCOMM)
 SET (SALARY, COMM) = (AVGSAL, AVGCOMM)
 WHERE EMPNO = '000120'

```

The previous statement is semantically equivalent to the following statement, but requires only one access to the EMPLOYEE table, whereas the following statement specifies the EMPLOYEE table twice.

```

UPDATE EMPLOYEE EU
 SET (EU.SALARY, EU.COMM)
 =
 (SELECT AVG(ES.SALARY), AVG(ES.COMM)
 FROM EMPLOYEE ES
 WHERE ES.WORKDEPT = EU.WORKDEPT)
 WHERE EU.EMPNO = '000120'

```

- *Example 5:* In a C program display the rows from the EMPLOYEE table and then, if requested to do so, change the job (JOB) of certain employees to the new job keyed in.

```

EXEC SQL DECLARE C1 CURSOR FOR
 SELECT *
 FROM EMPLOYEE
 FOR UPDATE OF JOB;

EXEC SQL OPEN C1;

EXEC SQL FETCH C1 INTO ... ;
if (strcmp (change, "YES") == 0)
 EXEC SQL UPDATE EMPLOYEE
 SET JOB = :newjob
 WHERE CURRENT OF C1;

EXEC SQL CLOSE C1;

```

- *Example 6:* These examples mutate attributes of column objects. Assume that the following types and tables exist:

```

CREATE TYPE POINT AS (X INTEGER, Y INTEGER)
 NOT FINAL WITHOUT COMPARISONS
 MODE DB2SQL

CREATE TYPE CIRCLE AS (RADIUS INTEGER, CENTER POINT)
 NOT FINAL WITHOUT COMPARISONS
 MODE DB2SQL

CREATE TABLE CIRCLES (ID INTEGER, OWNER VARCHAR(50), C CIRCLE)

```

The following example updates the CIRCLES table by changing the OWNER column and the RADIUS attribute of the CIRCLE column where the ID is 999:

```

UPDATE CIRCLES
 SET OWNER = 'Bruce'
 C..RADIUS = 5
 WHERE ID = 999

```

The following example transposes the X and Y coordinates of the center of the circle identified by 999:

```

UPDATE CIRCLES
 SET C..CENTER..X = C..CENTER..Y,
 C..CENTER..Y = C..CENTER..X
 WHERE ID = 999

```

The following example is another way of writing both of the above statements. This example combines the effects of both of the above examples:

```

UPDATE CIRCLES
 SET (OWNER,C..RADIUS,C..CENTER..X,C..CENTER..Y) =
 ('Bruce',5,C..CENTER..Y,C..CENTER..X)
 WHERE ID = 999

```

- *Example 7:* Update the XMLDOC column of the DOCUMENTS table with DOCID '001' to the character string that is selected and parsed from the XMLTEXT table.

```

UPDATE DOCUMENTS SET XMLDOC =
 (SELECT XMLPARSE(DOCUMENT C1 STRIP WHITESPACE)
 FROM XMLTEXT WHERE TEXTID = '001')
WHERE DOCID = '001'

```



---

## Chapter 2. SQL Statements for Users

---

### COMMIT

The COMMIT statement terminates a unit of work and commits the database changes that were made by that unit of work.

#### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

#### Authorization

None required.

#### Syntax

▶—COMMIT—WORK—▶

#### Description

The unit of work in which the COMMIT statement is executed is terminated and a new unit of work is initiated. All changes made by the following statements executed during the unit of work are committed: ALTER, COMMENT, CREATE, DROP, GRANT, LOCK TABLE, REVOKE, SET INTEGRITY, SET Variable, and the data change statements (INSERT, DELETE, MERGE, UPDATE), including those nested in a query.

The following statements, however, are not under transaction control and changes made by them are independent of the COMMIT statement:

- SET CONNECTION
- SET PASSTHRU

**Note:** Although the SET PASSTHRU statement is not under transaction control, the passthru session initiated by the statement is under transaction control.

- SET SERVER OPTION
- Assignments to updatable special registers

All locks acquired by the unit of work subsequent to its initiation are released, except necessary locks for open cursors that are declared WITH HOLD. All open cursors not defined WITH HOLD are closed. Open cursors defined WITH HOLD remain open, and the cursor is positioned before the next logical row of the result table. (A FETCH must be performed before a positioned UPDATE or DELETE statement is issued.) All LOB locators are freed. Note that this is true even when the locators are associated with LOB values retrieved via a cursor that has the WITH HOLD property.

All savepoints set within the transaction are released.

The following statements behave differently than other data definition language (DDL) and data control language (DCL) statements. Changes made by these statements do not take effect until the statement is committed, even for the current connection that issues the statement. Only one of these statements can be issued by any application at a time, and only one of these statements is allowed within any one unit of work. Each statement must be followed by a COMMIT or a ROLLBACK statement before another one of these statements can be issued.

- CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
- CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)
- CREATE WORK ACTION, ALTER WORK ACTION, or DROP (WORK ACTION)
- CREATE WORK CLASS, ALTER WORK CLASS, or DROP (WORK CLASS)
- CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
- GRANT (Workload Privileges) or REVOKE (Workload Privileges)

### Notes

- It is strongly recommended that each application process explicitly ends its unit of work before terminating. If the application program ends normally without a COMMIT or ROLLBACK statement then the database manager attempts a commit or rollback depending on the application environment.
- For information on the impact of COMMIT on cached dynamic SQL statements, see "EXECUTE".
- For information on potential impacts of COMMIT on created temporary tables, see "CREATE GLOBAL TEMPORARY TABLE".
- For information on potential impacts of COMMIT on declared temporary tables, see "DECLARE GLOBAL TEMPORARY TABLE".

### Example

Commit alterations to the database made since the last commit point.

```
COMMIT WORK
```

---

## CONNECT (Type 1)

The CONNECT (Type 1) statement connects an application process to the identified application server according to the rules for remote unit of work.

An application process can only be connected to one application server at a time. This is called the *current server*. A default application server may be established when the application requester is initialized. If implicit connect is available and an application process is started, it is implicitly connected to the default application server. The application process can explicitly connect to a different application server by issuing a CONNECT statement. A connection lasts until a CONNECT RESET statement or a DISCONNECT statement is issued or until another CONNECT statement changes the application server.

### Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically

prepared. When invoked using the command line processor, additional options can be specified. For more information, refer to “Using command line SQL statements and XQuery statements”.

## Authorization

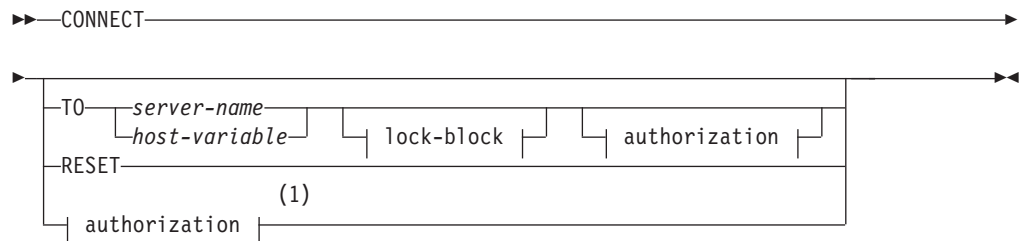
CONNECT processing goes through two levels of access control. Both levels must be satisfied for the connection to be successful.

The first level of access control is authentication, where the user ID associated with the connection must be successfully authenticated according to the authentication method set up for the server. At successful authentication, a DB2 authorization ID is derived from the connection user ID according to the authentication plug-in in effect for the server. This DB2 authorization ID must then pass the second level of access control for the connection, that is, authorization. To do so, this authorization ID must hold at least one of the following authorities:

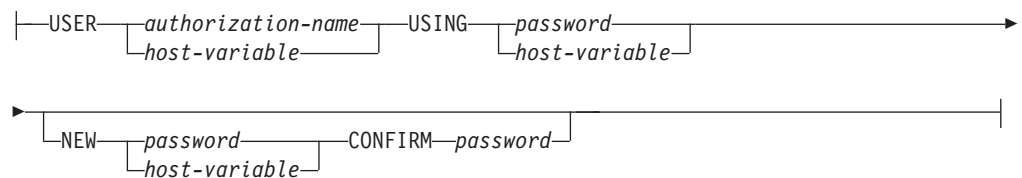
- CONNECT authority
- SECADM authority
- DBADM authority
- SYSADM authority
- SYSCTRL authority
- SYSMANT authority
- SYSMON authority

**Note:** For a partitioned database, the user and group definitions must be identical across all database partitions.

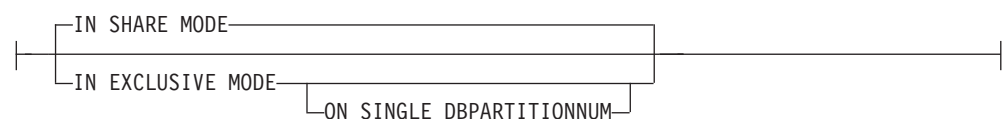
## Syntax



### authorization:



### lock-block:



## Notes:

- 1 This form is only valid if implicit connect is enabled.

## Description

### CONNECT (with no operand)

Returns information about the current server. The information is returned in the SQLERRP field of the SQLCA as described in “Successful Connection”.

If a connection state exists, the authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If the authorization ID is longer than 8 bytes, it will be truncated to 8 bytes, and the truncation will be flagged in the SQLWARN0 and SQLWARN1 fields of the SQLCA, with 'W' and 'A', respectively. If the database configuration parameter **dyn\_query\_mgmt** is enabled, then the SQLWARN0 and SQLWARN7 fields of the SQLCA will be flagged with 'W' and 'E', respectively.

If no connection exists and implicit connect is possible, then an attempt to make an implicit connection is made. If implicit connect is not available, this attempt results in an error (no existing connection). If no connection, then the SQLERRMC field is blank.

The territory code and code page of the application server are placed in the SQLERRMC field (as they are with a successful CONNECT statement).

This form of CONNECT:

- Does not require the application process to be in the connectable state.
- If connected, does not change the connection state.
- If unconnected and implicit connect is available, a connection to the default application server is made. In this case, the country or region code and code page of the application server are placed in the SQLERRMC field, like a successful CONNECT statement.
- If unconnected and implicit connect is not available, the application process remains unconnected.
- Does not close cursors.

### TO *server-name* or *host-variable*

Identifies the application server by the specified *server-name* or a *host-variable* which contains the server-name.

If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-justified and must not be delimited by quotation marks.

Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

When the CONNECT statement is executed, the application process must be in the connectable state.

### *Successful Connection:*

If the CONNECT statement is successful:

- All open cursors are closed, all prepared statements are destroyed, and all locks are released from the previous application server.
- The application process is disconnected from its previous application server, if any, and connected to the identified application server.

- The actual name of the application server (not an alias) is placed in the CURRENT SERVER special register.
- Information about the application server is placed in the SQLERRP field of the SQLCA. If the application server is an IBM product, the information has the form *pppvrrm*, where:
  - *ppp* identifies the product as follows:
    - DSN for DB2 for z/OS
    - ARI for DB2 Server for VSE & VM
    - QSQ for DB2 for i5/OS
    - SQL for DB2 Database for Linux, UNIX, and Windows
  - *vv* is a two-digit version identifier, such as '08'
  - *rr* is a two-digit release identifier, such as '01'
  - *m* is a one-character modification level identifier, such as '0'.

This release (Version 9.5) of DB2 Database for Linux, UNIX, and Windows is identified as 'SQL09050'.

- The SQLERRMC field of the SQLCA is set to contain the following values (separated by X'FF')
1. the country or region code of the application server (or blanks if using DB2 Connect),
  2. the code page of the application server (or CCSID if using DB2 Connect),
  3. the authorization ID (up to first 8 bytes only),
  4. the database alias,
  5. the platform type of the application server. Currently identified values are:

**Token Server**

**QAS** DB2 for System i®

**QDB2** DB2 for z/OS

**QDB2/6000**  
DB2 Database for AIX

**QDB2/HPUX**  
DB2 Database for HP-UX

**QDB2/LINUX**  
DB2 Database for Linux

**QDB2/NT**  
DB2 Database for Windows

**QDB2/SUN**  
DB2 Database for Solaris Operating System

**QSQLDS/VM**  
DB2 Server for VM

**QSQLDS/VSE**  
DB2 Server for VSE

6. The agent ID. It identifies the agent executing within the database manager on behalf of the application. This field is the same as the **agent\_id** element returned by the database monitor.

7. The agent index. It identifies the index of the agent and is used for service.
  8. Database partition number. For a non-partitioned database, this is always 0, if present.
  9. The code page of the application client.
  10. Number of database partitions in a partitioned database. If the database cannot be distributed, the value is 0 (zero). Token is present only with Version 5 or later.
- The SQLERRD(1) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the database code page from the application code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction.
  - The SQLERRD(2) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the application code page from the database code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction.
  - The SQLERRD(3) field of the SQLCA indicates whether or not the database on the connection is updatable. A database is initially updatable, but is changed to read-only if a unit of work determines the authorization ID cannot perform updates. The value is one of:
    - 1 - updatable
    - 2 - read-only
  - The SQLERRD(4) field of the SQLCA returns certain characteristics of the connection. The value is one of:
    - 0 N/A (only possible if running from a down-level client that is one-phase commit and is an updater).
    - 1 one-phase commit.
    - 2 one-phase commit; read-only (only applicable to connections to DRDA1 databases in a TP Monitor environment).
    - 3 two-phase commit.
  - The SQLERRD(5) field of the SQLCA returns the authentication type for the connection. The value is one of:
    - 0 Authenticated on the server.
    - 1 Authenticated on the client.
    - 2 Authenticated using DB2 Connect.
    - 4 Authenticated on the server with encryption.
    - 5 Authenticated using DB2 Connect with encryption.
    - 7 Authenticated using an external Kerberos security mechanism.
    - 9 Authenticated using an external GSS API plug-in security mechanism.
    - 11 Authenticated on the server, which accepts encrypted data.
    - 255 Authentication not specified.
  - The SQLERRD(6) field of the SQLCA returns the database partition number of the database partition to which the connection was made if the database is distributed. Otherwise, a value of 0 is returned.

- The SQLWARN1 field in the SQLCA will be set to 'A' if the authorization ID of the successful connection is longer than 8 bytes. This indicates that truncation has occurred. The SQLWARN0 field in the SQLCA will be set to 'W' to indicate this warning.
- The SQLWARN7 field in the SQLCA will be set to 'E' if the database configuration parameter **dyn\_query\_mgmt** for the database is enabled. The SQLWARN0 field in the SQLCA will be set to 'W' to indicate this warning.

**Unsuccessful Connection:**

If the CONNECT statement is unsuccessful:

- The SQLERRP field of the SQLCA is set to the name of the module at the application requester that detected the error. The first three characters of the module name identify the product.
- If the CONNECT statement is unsuccessful because the application process is not in the connectable state, the connection state of the application process is unchanged.
- If the CONNECT statement is unsuccessful because the *server-name* is not listed in the local directory, an error message (SQLSTATE 08001) is issued and the connection state of the application process remains unchanged:
  - If the application requester was not connected to an application server then the application process remains unconnected.
  - If the application requester was already connected to an application server, the application process remains connected to that application server. Any further statements are executed at that application server.
- If the CONNECT statement is unsuccessful for any other reason, the application process is placed into the unconnected state.

**IN SHARE MODE**

Allows other concurrent connections to the database and prevents other users from connecting to the database in exclusive mode.

**IN EXCLUSIVE MODE**

Prevents concurrent application processes from executing any operations at the application server, unless they have the same authorization ID as the user holding the exclusive lock. This option is not supported by DB2 Connect.

**ON SINGLE DBPARTITIONNUM**

Specifies that the coordinator database partition is connected in exclusive mode and all other database partitions are connected in share mode. This option is only effective in a partitioned database.

**RESET**

Disconnects the application process from the current server. A commit operation is performed. If implicit connect is available, the application process remains unconnected until an SQL statement is issued.

**USER** *authorization-name/host-variable*

Identifies the user ID trying to connect to the application server. If a host-variable is specified, it must be a character string variable that does not include an indicator variable. The user ID that is contained within the *host-variable* must be left justified and must not be delimited by quotation marks.

**USING** *password/host-variable*

Identifies the password of the user ID trying to connect to the application server. The *password* or *host-variable* can be up to 14 bytes long. If a host



variable is specified, it must be a character string variable with a length attribute not greater than 14, and it must not include an indicator variable.

**NEW** *password*/*host-variable* **CONFIRM** *password*

Identifies the new password that should be assigned to the user ID identified by the USER option. The *password* or *host-variable* can be up to 14 bytes long. If a host variable is specified, it must be a character string variable with a length attribute not greater than 14, and it must not include an indicator variable. The system on which the password will be changed depends on how the user authentication has been set up. New passwords can be assigned using this clause on the following servers for the indicated (and later) releases: DB2 Universal Database Version 8 on AIX and Windows operating systems, DB2 Version 9.1 Fix Pack 3 or later on Linux operating systems, DB2 for z/OS Version 7, DB2 for i5/OS V6R1. To support the changing passwords for DB2 database products on Linux, the DB2 instance must be configured to use the security plug-ins IBMOSchgpwdclient and IBMOSchgpwdserver.

## Notes

- It is good practice for the first SQL statement executed by an application process to be the CONNECT statement.
- If a CONNECT statement is issued to the current application server with a different user ID and password then the conversation is deallocated and reallocated. All cursors are closed by the database manager (with the loss of the cursor position if the WITH HOLD option was used).
- If a CONNECT statement is issued to the current application server with the same user ID and password then the conversation is not deallocated and reallocated. Cursors, in this case, are not closed.
- To use a multiple-partition partitioned database environment, the user or application must connect to one of the database partitions listed in the db2nodes.cfg file. You should try to ensure that not all users use the same database partition as the coordinator partition.
- The *authorization-name* SYSTEM cannot be explicitly specified in the CONNECT statement. However, on Windows operating systems, local applications running under the Local System Account can implicitly connect to the database, such that the user ID is SYSTEM.
- When connecting to Windows Server explicitly, the *authorization-name* or user *host-variable* can be specified using the Microsoft Windows Security Account Manager (SAM)-compatible name. The qualifier must be a NetBIOS style name, which has a maximum length of 15 bytes. For example, 'Domain\User'.
- **Compatibilities:** For compatibility with previous versions of DB2 products:
  - NODE can be specified in place of DBPARTITIONNUM

## Examples

*Example 1:* In a C program, connect to the application server TOROLAB, using database alias TOROLAB, user ID FERMAT, and password THEOREM.

```
EXEC SQL CONNECT TO TOROLAB USER FERMAT USING THEOREM;
```

*Example 2:* In a C program, connect to an application server whose database alias is stored in the host variable APP\_SERVER (varchar(8)). Following a successful connection, copy the 3-character product identifier of the application server to the variable PRODUCT (char(3)).

```
EXEC SQL CONNECT TO :APP_SERVER;
if (strncmp(SQLSTATE,'00000',5))
 strncpy(PRODUCT,sqlca.sqlerrp,3);
```

---

## CONNECT (Type 2)

The CONNECT (Type 2) statement connects an application process to the identified application server and establishes the rules for application-directed distributed unit of work. This server is then the current server for the process.

Most aspects of a CONNECT (Type 1) statement also apply to a CONNECT (Type 2) statement. Rather than repeating that material here, this section describes only those elements of Type 2 that differ from Type 1.

### Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared. When invoked using the command line processor, additional options can be specified. For more information, refer to “Using command line SQL statements and XQuery statements”.

### Authorization

CONNECT processing goes through two levels of access control. Both levels must be satisfied for the connection to be successful.

The first level of access control is authentication, where the user ID associated with the connection must be successfully authenticated according to the authentication method set up for the server. At successful authentication, a DB2 authorization ID is derived from the connection user ID according to the authentication plug-in in effect for the server. This DB2 authorization ID must then pass the second level of access control for the connection, that is, authorization. To do so, this authorization ID must hold at least one of the following authorities:

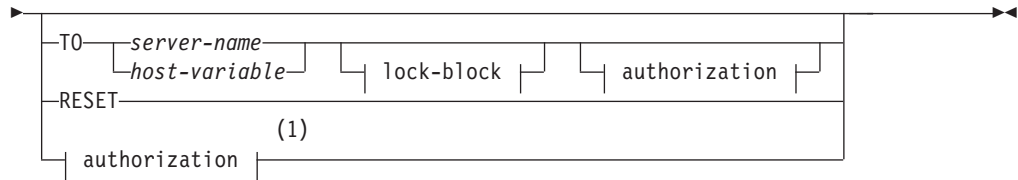
- CONNECT authority
- SECADM authority
- DBADM authority
- SYSADM authority
- SYSCTRL authority
- SYSMANT authority
- SYSMON authority

**Note:** For a partitioned database, the user and group definitions must be identical across all database partitions.

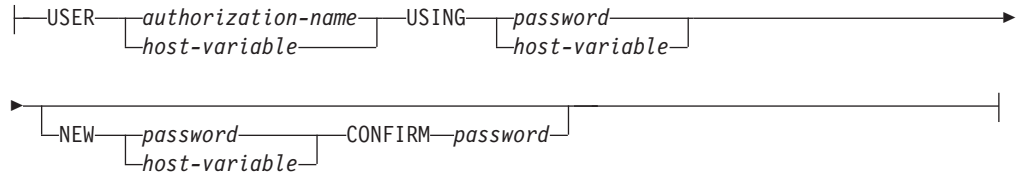
### Syntax

The selection between Type 1 and Type 2 is determined by precompiler options. For an overview of these options, see “Connecting to distributed relational databases”.

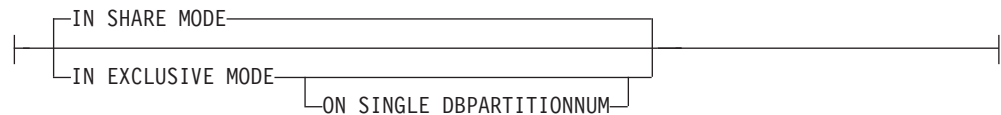
▶▶—CONNECT—————▶▶



**authorization:**



**lock-block:**



**Notes:**

- 1 This form is only valid if implicit connect is enabled.

**Description**

**TO** *server-name/host-variable*

The rules for coding the name of the server are the same as for Type 1.

If the SQLRULES(STD) option is in effect, the *server-name* must not identify an existing connection of the application process, otherwise an error (SQLSTATE 08002) is raised.

If the SQLRULES(DB2) option is in effect and the *server-name* identifies an existing connection of the application process, that connection is made current and the old connection is placed into the dormant state. That is, the effect of the CONNECT statement in this situation is the same as that of a SET CONNECTION statement.

For information about the specification of SQLRULES, see “Options that Govern Distributed Unit of Work Semantics”.

**Successful Connection**

If the CONNECT statement is successful:

- A connection to the application server is either created (or made non-dormant) and placed into the current and held states.
- If the CONNECT TO is directed to a different server than the current server, then the current connection is placed into the dormant state.
- The CURRENT SERVER special register and the SQLCA are updated in the same way as for CONNECT (Type 1).

**Unsuccessful Connection**

If the CONNECT statement is unsuccessful:

- No matter what the reason for failure, the connection state of the application process and the states of its connections are unchanged.
- As with an unsuccessful Type 1 CONNECT, the SQLERRP field of the SQLCA is set to the name of the module at the application requester or server that detected the error.

### **CONNECT (with no operand), IN SHARE/EXCLUSIVE MODE, USER, and USING**

If a connection exists, Type 2 behaves like a Type 1. The authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If a connection does not exist, no attempt to make an implicit connection is made and the SQLERRP and SQLERRMC fields return a blank. (Applications can check if a current connection exists by checking these fields.)

A CONNECT with no operand that includes USER and USING can still connect an application process to a database using the DB2DBDFT environment variable. This method is equivalent to a Type 2 CONNECT RESET, but permits the use of a user ID and password.

### **RESET**

Equivalent to an explicit connect to the default database if it is available. If a default database is not available, the connection state of the application process and the states of its connections are unchanged.

Availability of a default database is determined by installation options, environment variables, and authentication settings.

### **Rules**

- As outlined in “Options that Govern Distributed Unit of Work Semantics”, a set of connection options governs the semantics of connection management. Default values are assigned to every preprocessed source file. An application can consist of multiple source files precompiled with different connection options.

Unless a SET CLIENT command or API has been executed first, the connection options used when preprocessing the source file containing the first SQL statement executed at run time become the effective connection options.

If a CONNECT statement from a source file preprocessed with different connection options is subsequently executed without the execution of any intervening SET CLIENT command or API, an error (SQLSTATE 08001) is returned. Note that once a SET CLIENT command or API has been executed, the connection options used when preprocessing all source files in the application are ignored.

Example 1 in the “Examples” section of this statement illustrates these rules.

- Although the CONNECT statement can be used to establish or switch connections, CONNECT with the USER/USING clause will only be accepted when there is no current or dormant connection to the named server. The connection must be released before issuing a connection to the same server with the USER/USING clause, otherwise it will be rejected (SQLSTATE 51022). Release the connection by issuing a DISCONNECT statement or a RELEASE statement followed by a COMMIT statement.

### **Notes**

- Implicit connect is supported for the first SQL statement in an application with Type 2 connections. In order to execute SQL statements on the default database, first the CONNECT RESET or the CONNECT USER/USING statement must be used to establish the connection. The CONNECT statement with no operands

will display information about the current connection if there is one, but will not connect to the default database if there is no current connection.

- The *authorization-name* SYSTEM cannot be explicitly specified in the CONNECT statement. However, on Windows operating systems, local applications running under the Local System Account can implicitly connect to the database, such that the user ID is SYSTEM.
- When connecting to Windows Server explicitly, the *authorization-name* or user *host-variable* can be specified using the Microsoft Windows Security Account Manager (SAM)-compatible name. The qualifier must be a NetBIOS style name, which has a maximum length of 15 bytes. For example, 'Domain\User'.

### Comparing Type 1 and Type 2 CONNECT Statements:

The semantics of the CONNECT statement are determined by the CONNECT precompiler option or the SET CLIENT API (see "Options that Govern Distributed Unit of Work Semantics"). CONNECT Type 1 or CONNECT Type 2 can be specified and the CONNECT statements in those programs are known as Type 1 and Type 2 CONNECT statements, respectively. Their semantics are described below:

#### Use of CONNECT:

| Type 1                                                                                                                                                                               | Type 2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Each unit of work can only establish connection to one application server.                                                                                                           | Each unit of work can establish connection to multiple application servers.                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| The current unit of work must be committed or rolled back before allowing a connection to another application server.                                                                | The current unit of work need not be committed or rolled back before connecting to another application server.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| The CONNECT statement establishes the current connection. Subsequent SQL requests are forwarded to this connection until changed by another CONNECT.                                 | Same as Type 1 CONNECT if establishing the first connection. If switching to a dormant connection and SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.                                                                                                                                                                                                                                                                                                                                            |
| Connecting to the current connection is valid and does not change the current connection.                                                                                            | Same as Type 1 CONNECT if the SQLRULES precompiler option is set to DB2. If SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.                                                                                                                                                                                                                                                                                                                                                                      |
| Connecting to another application server disconnects the current connection. The new connection becomes the current connection. Only one connection is maintained in a unit of work. | Connecting to another application server puts the current connection into the <i>dormant state</i> . The new connection becomes the current connection. Multiple connections can be maintained in a unit of work.<br><br>If the CONNECT is for an application server on a dormant connection, it becomes the current connection.<br><br>Connecting to a dormant connection using CONNECT is only allowed if SQLRULES(DB2) was specified. If SQLRULES(STD) was specified, then the SET CONNECTION statement must be used instead. |

---

**Type 1**

SET CONNECTION statement is supported for Type 1 connections, but the only valid target is the current connection.

---

**Type 2**

SET CONNECTION statement is supported for Type 2 connections to change the state of a connection from dormant to current.

**Use of CONNECT...USER...USING:**

---

**Type 1**

Connecting with the USER...USING clauses disconnects the current connection and establishes a new connection with the given authorization name and password.

---

**Type 2**

Connecting with the USER/USING clause will only be accepted when there is no current or dormant connection to the same named server.

**Use of Implicit CONNECT, CONNECT RESET, and Disconnecting:**

---

**Type 1**

CONNECT RESET can be used to disconnect the current connection.

---

**Type 2**

CONNECT RESET is equivalent to connecting to the default application server explicitly if one has been defined in the system.

Connections can be disconnected by the application at a successful COMMIT. Prior to the commit, use the RELEASE statement to mark a connection as release-pending. All such connections will be disconnected at the next COMMIT.

An alternative is to use the precompiler options DISCONNECT(EXPLICIT), DISCONNECT(CONDITIONAL), DISCONNECT(AUTOMATIC), or the DISCONNECT statement instead of the RELEASE statement.

---

After using CONNECT RESET to disconnect the current connection, if the next SQL statement is not a CONNECT statement, then it will perform an implicit connect to the default application server if one has been defined in the system.

---

CONNECT RESET is equivalent to an explicit connect to the default application server if one has been defined in the system.

---

It is an error to issue consecutive CONNECT RESETs.

---

It is an error to issue consecutive CONNECT RESETs ONLY if SQLRULES(STD) was specified because this option disallows the use of CONNECT to existing connection.

---

CONNECT RESET also implicitly commits the current unit of work.

---

CONNECT RESET does not commit the current unit of work.

---

If an existing connection is disconnected by the system for whatever reasons, then subsequent non-CONNECT SQL statements to this database will receive an SQLSTATE of 08003.

---

If an existing connection is disconnected by the system, COMMIT, ROLLBACK, and SET CONNECTION statements are still permitted.

---

The unit of work will be implicitly committed when the application process terminates successfully.

---

Same as Type 1.

**Type 1**

All connections (only one) are disconnected when the application process terminates.

**Type 2**

All connections (current, dormant, and those marked for release pending) are disconnected when the application process terminates.

**CONNECT Failures:****Type 1**

Regardless of whether there is a current connection when a CONNECT fails (with an error other than server-name not defined in the local directory), the application process is placed in the unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.

**Type 2**

If there is a current connection when a CONNECT fails, the current connection is unaffected.  
  
If there was no current connection when the CONNECT fails, then the program is then in an unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.

**Examples***Example 1:*

This example illustrates the use of multiple source programs (shown in the boxes), some preprocessed with different connection options (shown above the code), and one of which contains a SET CLIENT API call.

PGM1: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO OTTAWA;
exec sql SELECT col1 INTO :hv1
FROM tb11;
...
```

PGM2: CONNECT(2) SQLRULES(STD) DISCONNECT(AUTOMATIC)

```
...
exec sql CONNECT TO QUEBEC;
exec sql SELECT col1 INTO :hv1
FROM tb12;
...
```

PGM3: CONNECT(2) SQLRULES(STD) DISCONNECT(EXPLICIT)

```
...
SET CLIENT CONNECT 2 SQLRULES DB2 DISCONNECT EXPLICIT 1
exec sql CONNECT TO LONDON;
exec sql SELECT col1 INTO :hv1
FROM tb13;
...
```

<sup>1</sup> Note: not the actual syntax of the SET CLIENT API

PGM4: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO REGINA;
exec sql SELECT col1 INTO :hv1
FROM tb14;
...
```



If the application executes PGM1 then PGM2:

- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to QUEBEC fails with SQLSTATE 08001 because both SQLRULES and DISCONNECT are different.

If the application executes PGM1 then PGM3:

- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to LONDON runs: connect=2, sqlrules=DB2, disconnect=EXPLICIT

This is OK because the SET CLIENT API is run before the second CONNECT statement.

If the application executes PGM1 then PGM4:

- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to REGINA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL

This is OK because the preprocessor options for PGM1 are the same as those for PGM4.

*Example 2:*

This example shows the interrelationships of the CONNECT (Type 2), SET CONNECTION, RELEASE, and DISCONNECT statements. S0, S1, S2, and S3 represent four servers.

| Sequence | Statement                                | Current Server | Dormant Connections          | Release Pending  |
|----------|------------------------------------------|----------------|------------------------------|------------------|
| 0        | • No statement                           | • None         | • None                       | • None           |
| 1        | • SELECT * FROM TBLA                     | • S0 (default) | • None                       | • None           |
| 2        | • CONNECT TO S1<br>• SELECT * FROM TBLB  | • S1<br>• S1   | • S0<br>• S0                 | • None<br>• None |
| 3        | • CONNECT TO S2<br>• UPDATE TBLC SET ... | • S2<br>• S2   | • S0, S1<br>• S0, S1         | • None<br>• None |
| 4        | • CONNECT TO S3<br>• SELECT * FROM TBLD  | • S3<br>• S3   | • S0, S1, S2<br>• S0, S1, S2 | • None<br>• None |
| 5        | • SET CONNECTION S2                      | • S2           | • S0, S1, S3                 | • None           |
| 6        | • RELEASE S3                             | • S2           | • S0, S1                     | • S3             |
| 7        | • COMMIT                                 | • S2           | • S0, S1                     | • None           |
| 8        | • SELECT * FROM TBLE                     | • S2           | • S0, S1                     | • None           |
| 9        | • DISCONNECT S1<br>• SELECT * FROM TBLF  | • S2<br>• S2   | • S0<br>• S0                 | • None<br>• None |

## DISCONNECT

The DISCONNECT statement destroys one or more connections when there is no active unit of work (that is, after a commit or rollback operation). If a single connection is the target of the DISCONNECT statement, the connection is destroyed only if the database has participated in an existing unit of work,

regardless of whether there is an active unit of work. For example, if several other databases have done work, but the target in question has not, it can still be disconnected without destroying the connection.

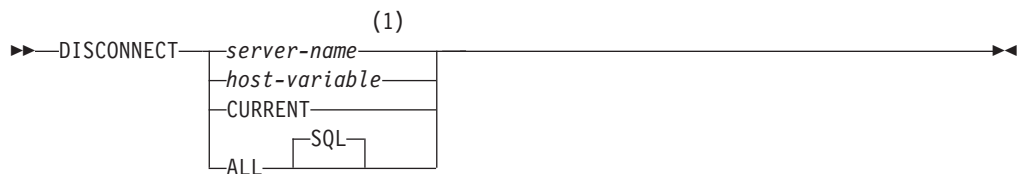
## Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

## Authorization

None required.

## Syntax



### Notes:

- 1 Note that an application server named CURRENT or ALL can only be identified by a host variable.

## Description

*server-name* or *host-variable*

Identifies the application server by the specified *server-name* or a *host-variable* which contains the *server-name*.

If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-justified and must not be delimited by quotation marks.

Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

The specified database-alias or the database-alias contained in the host variable must identify an existing connection of the application process. If the database-alias does not identify an existing connection, an error (SQLSTATE 08003) is raised.

### CURRENT

Identifies the current connection of the application process. The application process must be in the connected state. If not, an error (SQLSTATE 08003) is raised.

### ALL

Indicates that all existing connections of the application process are to be destroyed. An error or warning does not occur if no connections exist when the statement is executed. The optional keyword SQL is included to be consistent with the syntax of the RELEASE statement.

## Rules

- Generally, the DISCONNECT statement cannot be executed while within a unit of work. If attempted, an error (SQLSTATE 25000) is raised. The exception to this rule is if a single connection is specified to be disconnected and the database has not participated in an existing unit of work. In this case, it does not matter if there is an active unit of work when the DISCONNECT statement is issued.
- The DISCONNECT statement cannot be executed at all in the Transaction Processing (TP) Monitor environment (SQLSTATE 25000). It is used when the SYNCPOINT precompiler option is set to TWOPHASE.

## Notes

- If the DISCONNECT statement is successful, each identified connection is destroyed.  
If the DISCONNECT statement is unsuccessful, the connection state of the application process and the states of its connections are unchanged.
- If DISCONNECT is used to destroy the current connection, the next executed SQL statement should be CONNECT or SET CONNECTION.
- Type 1 CONNECT semantics do not preclude the use of DISCONNECT. However, though DISCONNECT CURRENT and DISCONNECT ALL can be used, they will not result in a commit operation like a CONNECT RESET statement would do.  
If *server-name* or *host-variable* is specified in the DISCONNECT statement, it must identify the current connection because Type 1 CONNECT only supports one connection at a time. Generally, DISCONNECT will fail if within a unit of work with the exception noted in “Rules”.
- Resources are required to create and maintain remote connections. Thus, a remote connection that is not going to be reused should be destroyed as soon as possible.
- Connections can also be destroyed during a commit operation because the connection option is in effect. The connection option could be AUTOMATIC, CONDITIONAL, or EXPLICIT, which can be set as a precompiler option or through the SET CLIENT API at run time. For information about the specification of the DISCONNECT option, see “Distributed relational databases”.

## Examples

*Example 1:* The SQL connection to IBMSTHDB is no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy the connection.

```
EXEC SQL DISCONNECT IBMSTHDB;
```

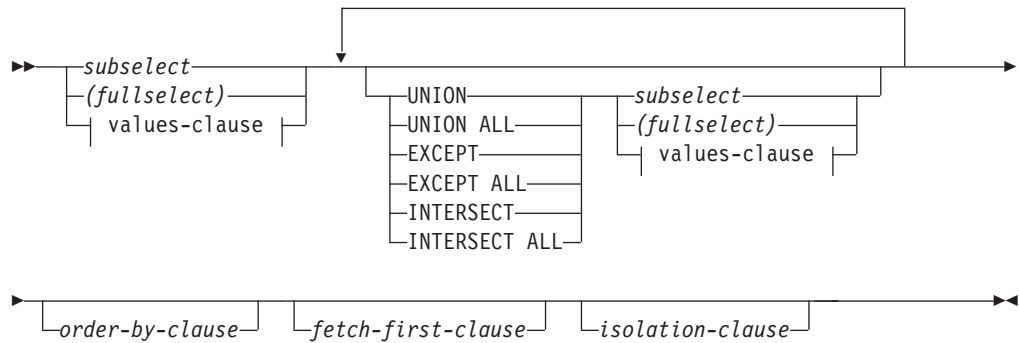
*Example 2:* The current connection is no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy the connection.

```
EXEC SQL DISCONNECT CURRENT;
```

*Example 3:* The existing connections are no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy all the connections.

```
EXEC SQL DISCONNECT ALL;
```

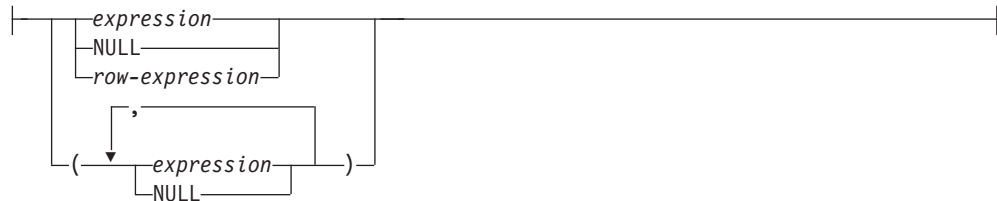
## fullselect



### values-clause:



### values-row:



The *fullselect* is a component of the select-statement, the INSERT statement, and the CREATE VIEW statement. It is also a component of certain predicates which, in turn, are components of a statement. A fullselect that is a component of a predicate is called a *subquery*, and a fullselect that is enclosed in parentheses is sometimes called a subquery.

The set operators UNION, EXCEPT, and INTERSECT correspond to the relational operators union, difference, and intersection.

A 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 values-clause.

The authorization for a *fullselect* is described in the Authorization section in "SQL queries".

### *values-clause*

Derives a result table by specifying the actual values, using expressions or row expressions, for each column of a row in the result table. Multiple rows may be specified. The result type of any expression in the *values-clause* cannot be a row type (SQLSTATE 428H2).

NULL can only be used with multiple specifications of *values-row*, either as the column value of a single column result table or within a *row-expression*, and at least one row in the same column must not be NULL (SQLSTATE 42608).

A *values-row* is specified by:

- A single expression for a single column result table
- $n$  expressions (or NULL) separated by commas and enclosed in parentheses, where  $n$  is the number of columns in the result table or, a row expression for a multiple column result table.

A multiple row VALUES clause must have the same number of columns in each *values-row* (SQLSTATE 42826).

The following are examples of *values-clause* and their meaning.

|                             |                       |
|-----------------------------|-----------------------|
| VALUES (1),(2),(3)          | - 3 rows of 1 column  |
| VALUES 1, 2, 3              | - 3 rows of 1 column  |
| VALUES (1, 2, 3)            | - 1 row of 3 columns  |
| VALUES (1,21),(2,22),(3,23) | - 3 rows of 2 columns |

A *values-clause* that is composed of  $n$  specifications of *values-row*,  $RE_1$  to  $RE_n$ , where  $n$  is greater than 1, is equivalent to:

$RE_1$  UNION ALL  $RE_2$  ... UNION ALL  $RE_n$

This means that the corresponding columns of each *values-row* must be comparable (SQLSTATE 42825).

#### UNION or UNION ALL

Derives a result table by combining two other result tables (R1 and R2). If UNION ALL is specified, the result consists of all rows in R1 and R2. If UNION is specified without the ALL option, the result is the set of all rows in either R1 or R2, with the duplicate rows eliminated. In either case, however, each row of the UNION table is either a row from R1 or a row from R2.

#### EXCEPT or EXCEPT ALL

Derives a result table by combining two other result tables (R1 and R2). If EXCEPT ALL is specified, the result consists of all rows that do not have a corresponding row in R2, where duplicate rows are significant. If EXCEPT is specified without the ALL option, the result consists of all rows that are only in R1, with duplicate rows in the result of this operation eliminated.

For compatibility with other SQL implementations, MINUS can be specified as a synonym for EXCEPT.

#### INTERSECT or INTERSECT ALL

Derives a result table by combining two other result tables (R1 and R2). If INTERSECT ALL is specified, the result consists of all rows that are in both R1 and R2. If INTERSECT is specified without the ALL option, the result consists of all rows that are in both R1 and R2, with the duplicate rows eliminated.

#### *order-by-clause*

See “subselect” for details of the *order-by-clause*. A fullselect that contains an ORDER BY clause cannot be specified in (SQLSTATE 428FJ):

- A materialized query table
- The outermost fullselect of a view

**Note:** An ORDER BY clause in a fullselect does not affect the order of the rows returned by a query. An ORDER BY clause only affects the order of the rows returned if it is specified in the outermost fullselect.

| *fetch-first-clause*

| See “subselect” for details of the *fetch-first-clause*. A fullselect that contains a  
| FETCH FIRST clause cannot be specified in (SQLSTATE 428FJ):

- | • A materialized query table
- | • The outermost fullselect of a view

| **Note:** A FETCH FIRST clause in a fullselect does not affect the number of rows  
| returned by a query. A FETCH FIRST clause only affects the number of rows  
| returned if it is specified in the outermost fullselect.

| *isolation-clause*

| See “subselect” for details of the *isolation-clause*. If *isolation-clause* is specified for  
| a fullselect and it could apply equally to a subselect of the fullselect,  
| *isolation-clause* is applied to the fullselect. For example, consider the following  
| query.

```
| SELECT NAME FROM PRODUCT
| UNION
| SELECT NAME FROM CATALOG
| WITH UR
```

| Even though the isolation clause WITH UR could apply only to the subselect  
| SELECT NAME FROM CATALOG, it is applied to the whole fullselect.

| The number of columns in the result tables R1 and R2 must be the same  
(SQLSTATE 42826). If the ALL keyword is not specified, R1 and R2 must not  
| include any columns having a data type of CLOB, DBCLOB, BLOB, distinct type  
| on any of these types, or structured type (SQLSTATE 42907).

| The columns of the result are named as follows:

- | • If the *n*th column of R1 and the *n*th column of R2 have the same result column  
| name, then the *n*th column of R has the result column name.
- | • If the *n*th column of R1 and the *n*th column of R2 have different result column  
| names, a name is generated. This name cannot be used as the column name in  
| an ORDER BY or UPDATE clause.

| The generated name can be determined by performing a DESCRIBE of the SQL  
| statement and consulting the SQLNAME field.

| **Duplicate rows:** Two rows are duplicates if each value in the first is equal to the  
| corresponding value of the second. For determining duplicates, two null values are  
| considered equal, and two decimal floating-point representations of the same  
| number are considered equal. For example, 2.00 and 2.0 have the same value (2.00  
| and 2.0 compare as equal) but have different exponents, which allows you to  
| represent both 2.00 and 2.0. So, for example, if the result table of a UNION  
| operation contains a decimal floating-point column and multiple representations of  
| the same number exist, the one that is returned (for example, 2.00 or 2.0) is  
| unpredictable. For more information, see “Numeric comparisons” on page 733.

| When multiple operations are combined in an expression, operations within  
| parentheses are performed first. If there are no parentheses, the operations are  
| performed from left to right with the exception that all INTERSECT operations are  
| performed before UNION or EXCEPT operations.

| In the following example, the values of tables R1 and R2 are shown on the left.  
| The other headings listed show the values as a result of various set operations on  
| R1 and R2.

| R1 | R2 | UNION |       | EXCEPT |        | INTER- | INTER- |
|----|----|-------|-------|--------|--------|--------|--------|
|    |    | ALL   | UNION | ALL    | EXCEPT | SECT   | SECT   |
| 1  | 1  | 1     | 1     | 1      | 2      | 1      | 1      |
| 1  | 1  | 1     | 2     | 2      | 5      | 1      | 3      |
| 1  | 3  | 1     | 3     | 2      |        | 3      | 4      |
| 2  | 3  | 1     | 4     | 2      |        | 4      |        |
| 2  | 3  | 1     | 5     | 4      |        |        |        |
| 2  | 3  | 2     |       | 5      |        |        |        |
| 3  | 4  | 2     |       |        |        |        |        |
| 4  |    | 2     |       |        |        |        |        |
| 4  |    | 3     |       |        |        |        |        |
| 5  |    | 3     |       |        |        |        |        |
|    |    | 3     |       |        |        |        |        |
|    |    | 3     |       |        |        |        |        |
|    |    | 3     |       |        |        |        |        |
|    |    | 4     |       |        |        |        |        |
|    |    | 4     |       |        |        |        |        |
|    |    | 4     |       |        |        |        |        |
|    |    | 4     |       |        |        |        |        |
|    |    | 5     |       |        |        |        |        |

## Examples of a fullselect

*Example 1:* Select all columns and rows from the EMPLOYEE table.

```
SELECT * FROM EMPLOYEE
```

*Example 2:* List the employee numbers (EMPNO) of all employees in the EMPLOYEE table whose department number (WORKDEPT) either begins with 'E' or who are assigned to projects in the EMP\_ACT table whose project number (PROJNO) equals 'MA2100', 'MA2110', or 'MA2112'.

```
SELECT EMPNO
FROM EMPLOYEE
WHERE WORKDEPT LIKE 'E%'
UNION
SELECT EMPNO
FROM EMP_ACT
WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
```

*Example 3:* Make the same query as in example 2, and, in addition, "tag" the rows from the EMPLOYEE table with 'emp' and the rows from the EMP\_ACT table with 'emp\_act'. Unlike the result from example 2, this query may return the same EMPNO more than once, identifying which table it came from by the associated "tag".

```
SELECT EMPNO, 'emp'
FROM EMPLOYEE
WHERE WORKDEPT LIKE 'E%'
UNION
SELECT EMPNO, 'emp_act' FROM EMP_ACT
WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
```



*Example 4:* Make the same query as in example 2, only use UNION ALL so that no duplicate rows are eliminated.

```
SELECT EMPNO
 FROM EMPLOYEE
 WHERE WORKDEPT LIKE 'E%'
UNION ALL
SELECT EMPNO
 FROM EMP_ACT
 WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
```

*Example 5:* Make the same query as in Example 3, only include an additional two employees currently not in any table and tag these rows as "new".

```
SELECT EMPNO, 'emp'
 FROM EMPLOYEE
 WHERE WORKDEPT LIKE 'E%'
UNION
SELECT EMPNO, 'emp_act'
 FROM EMP_ACT
 WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
UNION
VALUES ('NEWAAA', 'new'), ('NEWBBB', 'new')
```

*Example 6:* This example of EXCEPT produces all rows that are in T1 but not in T2.

```
(SELECT * FROM T1)
EXCEPT ALL
(SELECT * FROM T2)
```

If no NULL values are involved, this example returns the same results as

```
SELECT ALL *
 FROM T1
 WHERE NOT EXISTS (SELECT * FROM T2
 WHERE T1.C1 = T2.C1 AND T1.C2 = T2.C2 AND...)
```

*Example 7:* This example of INTERSECT produces all rows that are in both tables T1 and T2, removing duplicates.

```
(SELECT * FROM T1)
INTERSECT
(SELECT * FROM T2)
```

If no NULL values are involved, this example returns the same result as

```
SELECT DISTINCT * FROM T1
 WHERE EXISTS (SELECT * FROM T2
 WHERE T1.C1 = T2.C1 AND T1.C2 = T2.C2 AND...)
```

where C1, C2, and so on represent the columns of T1 and T2.

---

## LOCK TABLE

The LOCK TABLE statement prevents concurrent application processes from using or changing a table.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

## Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- SELECT privilege on the table
- CONTROL privilege on the table
- DATAACCESS authority

## Syntax

```
▶▶ LOCK TABLE table-name | nickname IN SHARE | EXCLUSIVE MODE ▶▶
```

## Description

*table-name* or *nickname*

Identifies the table or nickname. The *table-name* must identify a table that exists at the application server, but it must not identify a catalog table, a created temporary table, or a declared temporary table (SQLSTATE 42995). If the *table-name* is a typed table, it must be the root table of the table hierarchy (SQLSTATE 428DR). When a nickname is specified, DB2 will lock the underlying object (that is, a table or view) of the data source to which the nickname refers.

### IN SHARE MODE

Prevents concurrent application processes from executing any but read-only operations on the table.

### IN EXCLUSIVE MODE

Prevents concurrent application processes from executing any operations on the table. Note that EXCLUSIVE MODE does not prevent concurrent application processes that are running at isolation level Uncommitted Read (UR) from executing read-only operations on the table.

## Notes

- Locking is used to prevent concurrent operations. A lock is not necessarily acquired during execution of the LOCK TABLE statement if a suitable lock already exists. The lock that prevents concurrent operations is held at least until termination of the unit of work.
- In a partitioned database, a table lock is first acquired at the first database partition in the database partition group (the database partition with the lowest number) and then at other database partitions. If the LOCK TABLE statement is interrupted, the table may be locked on some database partitions but not on others. If this occurs, either issue another LOCK TABLE statement to complete the locking on all database partitions, or issue a COMMIT or ROLLBACK statement to release the current locks.
- This statement affects all database partitions in the database partition group.
- For partitioned tables, the only lock acquired for the LOCK TABLE statement is at the table level; no data partition locks are acquired.

## Example

Obtain a lock on the table EMP. Do not allow other programs to read or update the table.

```
LOCK TABLE EMP IN EXCLUSIVE MODE
```

---

# ROLLBACK

The ROLLBACK statement is used to back out of the database changes that were made within a unit of work or a savepoint.

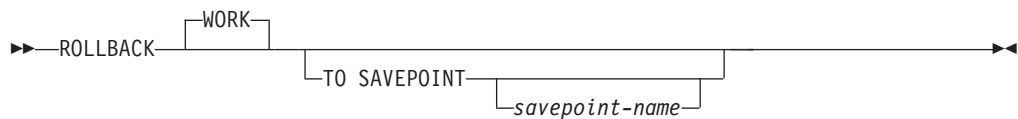
## Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

## Authorization

None required.

## Syntax



## Description

The unit of work in which the ROLLBACK statement is executed is terminated and a new unit of work is initiated. All changes made to the database during the unit of work are backed out.

The following statements, however, are not under transaction control, and changes made by them are independent of the ROLLBACK statement:

- SET CONNECTION
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT LOCK TIMEOUT
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT PACKAGESET
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET ENCRYPTION PASSWORD
- SET EVENT MONITOR STATE
- SET PASSTHRU

**Note:** Although the SET PASSTHRU statement is not under transaction control, the passthru session initiated by the statement is under transaction control.

- SET PATH
- SET SCHEMA
- SET SERVER OPTION

The generation of sequence and identity values is not under transaction control. Values generated and consumed by the *nextval-expression* or by inserting rows into a table that has an identity column are independent of issuing the ROLLBACK statement. Also, issuing the ROLLBACK statement does not affect the value returned by the *prevval-expression*, nor the IDENTITY\_VAL\_LOCAL function.

Modification of the values of global variables is not under transaction control. ROLLBACK statements do not affect the values assigned to global variables.

## TO SAVEPOINT

Specifies that a partial rollback (ROLLBACK TO SAVEPOINT) is to be performed. If no savepoint is active in the current savepoint level (see the “Rules” section in the description of the SAVEPOINT statement), an error is returned (SQLSTATE 3B502). After a successful rollback, the savepoint continues to exist, but any nested savepoints are released and no longer exist. The nested savepoints, if any, are considered to have been rolled back and then released as part of the rollback to the current savepoint. If a *savepoint-name* is not provided, rollback occurs to the most recently set savepoint within the current savepoint level.

If this clause is omitted, the ROLLBACK statement rolls back the entire transaction. Furthermore, savepoints within the transaction are released.

### *savepoint-name*

Specifies the savepoint that is to be used in the rollback operation. The specified *savepoint-name* cannot begin with ‘SYS’ (SQLSTATE 42939). After a successful rollback operation, the named savepoint continues to exist. If the savepoint name does not exist, an error (SQLSTATE 3B001) is returned. Data and schema changes made since the savepoint was set are undone.

## Notes

- All locks held are released on a ROLLBACK of the unit of work. All open cursors are closed. All LOB locators are freed.
- Executing a ROLLBACK statement does not affect either the SET statements that change special register values or the RELEASE statement.
- If the program terminates abnormally, the unit of work is implicitly rolled back.
- Statement caching is affected by the rollback operation.
- The impact on cursors resulting from a ROLLBACK TO SAVEPOINT depends on the statements within the savepoint
  - If the savepoint contains DDL on which a cursor is dependent, the cursor is marked invalid. Attempts to use such a cursor results in an error (SQLSTATE 57007).
  - Otherwise:
    - If the cursor is referenced in the savepoint, the cursor remains open and is positioned before the next logical row of the result table. (A FETCH must be performed before a positioned UPDATE or DELETE statement is issued.)
    - Otherwise, the cursor is not affected by the ROLLBACK TO SAVEPOINT (it remains open and positioned).
- Dynamically prepared statement names are still valid, although the statement may be implicitly prepared again, as a result of DDL operations that are rolled back within the savepoint.
- A ROLLBACK TO SAVEPOINT operation will drop any created temporary tables created within the savepoint. If a created temporary table is modified within the savepoint and that table has been defined as not logged, then all rows in the table are deleted.

- A ROLLBACK TO SAVEPOINT operation will drop any declared temporary tables declared within the savepoint. If a declared temporary table is modified within the savepoint and that table has been defined as not logged, then all rows in the table are deleted.
- All locks are retained after a ROLLBACK TO SAVEPOINT statement.
- All LOB locators are preserved following a ROLLBACK TO SAVEPOINT operation.

### Example

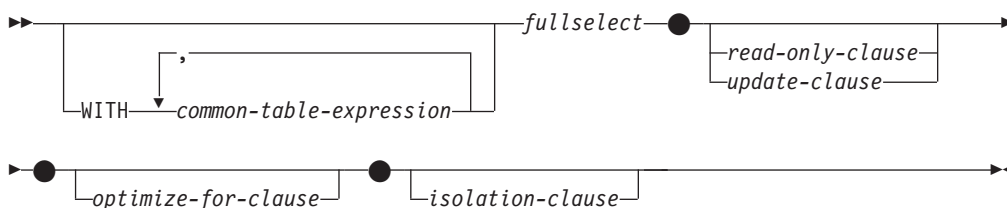
Delete the alterations made since the last commit point or rollback.

**ROLLBACK WORK**

## SELECT

The SELECT statement is a form of query. It can be embedded in an application program or issued interactively.

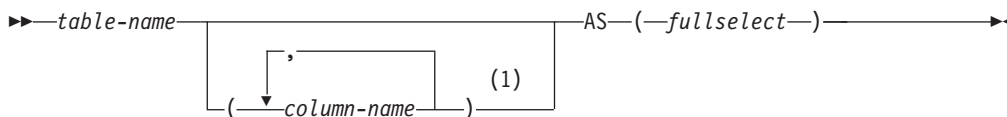
### select-statement



The *select-statement* is the form of a query that can be directly specified in a DECLARE CURSOR statement, or prepared and then referenced in a DECLARE CURSOR statement. It can also be issued through the use of dynamic SQL statements using the command line processor (or similar tools), causing a result table to be displayed on the user's screen. In either case, the table specified by a *select-statement* is the result of the fullselect.

The authorization for a *select-statement* is described in the Authorization section in "SQL queries".

### common-table-expression



#### Notes:

- 1 If a common table expression is recursive, or if the fullselect results in duplicate column names, column names must be specified.

A *common table expression* permits defining a result table with a *table-name* that can be specified as a table name in any FROM clause of the fullselect that follows. Multiple common table expressions can be specified following the single WITH

keyword. Each common table expression specified can also be referenced by name in the FROM clause of subsequent common table expressions.

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

The *table-name* of a common table expression must be different from any other common table expression *table-name* in the same statement (SQLSTATE 42726). If the common table expression is specified in an INSERT statement the *table-name* cannot be the same as the table or view name that is the object of the insert (SQLSTATE 42726). A common table expression *table-name* can be specified as a table name in any FROM clause throughout the fullselect. A *table-name* of a common table expression overrides any existing table, view or alias (in the catalog) with the same qualified name.

If more than one common table expression is defined in the same statement, cyclic references between the common table expressions are not permitted (SQLSTATE 42835). A *cyclic reference* occurs when two common table expressions *dt1* and *dt2* are created such that *dt1* refers to *dt2* and *dt2* refers to *dt1*.

If the fullselect of a common table expression contains a *data-change-table-reference* in the FROM clause, the common table expression is said to modify data. A common table expression that modifies data is always evaluated when the statement is processed, regardless of whether the common table expression is used anywhere else in the statement. If there is at least one common table expression that reads or modifies data, all common table expressions are processed in the order in which they occur, and each common table expression that reads or modifies data is completely executed, including all constraints and triggers, before any subsequent common table expressions are executed.

The common table expression is also optional prior to the fullselect in the CREATE VIEW and INSERT statements.

A common table expression can be used:

- In place of a view to avoid creating the view (when general use of the view is not required and positioned updates or deletes are not used)
- To enable grouping by a column that is derived from a scalar subselect or function that is not deterministic or has external action
- When the desired result table is based on host variables
- When the same result table needs to be shared in a fullselect
- When the result needs to be derived using recursion
- When multiple SQL data change statements need to be processed within the query

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

The following must be true of a recursive common table expression:

- Each fullselect that is part of the recursion cycle must start with SELECT or SELECT ALL. Use of SELECT DISTINCT is not allowed (SQLSTATE 42925). Furthermore, the unions must use UNION ALL (SQLSTATE 42925).
- The column names must be specified following the *table-name* of the common table expression (SQLSTATE 42908).
- The first fullselect of the first union (the initialization fullselect) must not include a reference to any column of the common table expression in any FROM clause (SQLSTATE 42836).
- If a column name of the common table expression is referred to in the iterative fullselect, the data type, length, and code page for the column are determined based on the initialization fullselect. The corresponding column in the iterative fullselect must have the same data type and length as the data type and length determined based on the initialization fullselect and the code page must match (SQLSTATE 42825). However, for character string types, the length of the two data types may differ. In this case, the column in the iterative fullselect must have a length that would always be assignable to the length determined from the initialization fullselect.
- Each fullselect that is part of the recursion cycle must not include any aggregate functions, group-by-clauses, or having-clauses (SQLSTATE 42836).  
The FROM clauses of these fullselects can include at most one reference to a common table expression that is part of a recursion cycle (SQLSTATE 42836).
- The iterative fullselect and the overall recursive fullselect must not include an order-by-clause (SQLSTATE 42836).
- Subqueries (scalar or quantified) must not be part of any recursion cycles (SQLSTATE 42836).

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

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

A warning is issued if this syntax is not found in the recursive common table expression (SQLSTATE 01605).

## Recursion example: bill of materials

Bill of materials (BOM) applications are a common requirement in many business environments. To illustrate the capability of a recursive common table expression for BOM applications, consider a table of parts with associated subparts and the quantity of subparts required by the part. For this example, create the table as follows:

```
CREATE TABLE PARTLIST
(PART VARCHAR(8),
SUBPART VARCHAR(8),
QUANTITY INTEGER);
```

To give query results for this example, assume that the PARTLIST table is populated with the following values:



| PART | SUBPART | QUANTITY |
|------|---------|----------|
| 00   | 01      | 5        |
| 00   | 05      | 3        |
| 01   | 02      | 2        |
| 01   | 03      | 3        |
| 01   | 04      | 4        |
| 01   | 06      | 3        |
| 02   | 05      | 7        |
| 02   | 06      | 6        |
| 03   | 07      | 6        |
| 04   | 08      | 10       |
| 04   | 09      | 11       |
| 05   | 10      | 10       |
| 05   | 11      | 10       |
| 06   | 12      | 10       |
| 06   | 13      | 10       |
| 07   | 14      | 8        |
| 07   | 12      | 8        |

### Example 1: Single level explosion

The first example is called single level explosion. It answers the question, “What parts are needed to build the part identified by ‘01?’”. The list will include the direct subparts, subparts of the subparts and so on. However, if a part is used multiple times, its subparts are only listed once.

```

WITH RPL (PART, SUBPART, QUANTITY) AS
(SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
 FROM PARTLIST ROOT
 WHERE ROOT.PART = '01'
 UNION ALL
 SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
 FROM RPL PARENT, PARTLIST CHILD
 WHERE PARENT.SUBPART = CHILD.PART
)
SELECT DISTINCT PART, SUBPART, QUANTITY
FROM RPL
ORDER BY PART, SUBPART, QUANTITY;

```

The above query includes a common table expression, identified by the name *RPL*, that expresses the recursive part of this query. It illustrates the basic elements of a recursive common table expression.

The first operand (fullselect) of the UNION, referred to as the *initialization fullselect*, gets the direct children of part ‘01’. The FROM clause of this fullselect refers to the source table and will never refer to itself (*RPL* in this case). The result of this first fullselect goes into the common table expression *RPL* (Recursive PARTLIST). As in this example, the UNION must always be a UNION ALL.

The second operand (fullselect) of the UNION uses *RPL* to compute subparts of subparts by having the FROM clause refer to the common table expression *RPL* and the source table with a join of a part from the source table (child) to a subpart of the current result contained in *RPL* (parent). The result goes back to *RPL* again. The second operand of UNION is then used repeatedly until no more children exist.

The SELECT DISTINCT in the main fullselect of this query ensures the same part/subpart is not listed more than once.

The result of the query is as follows:

| PART | SUBPART | QUANTITY |
|------|---------|----------|
| 01   | 02      | 2        |
| 01   | 03      | 3        |
| 01   | 04      | 4        |
| 01   | 06      | 3        |
| 02   | 05      | 7        |
| 02   | 06      | 6        |
| 03   | 07      | 6        |
| 04   | 08      | 10       |
| 04   | 09      | 11       |
| 05   | 10      | 10       |
| 05   | 11      | 10       |
| 06   | 12      | 10       |
| 06   | 13      | 10       |
| 07   | 12      | 8        |
| 07   | 14      | 8        |

Observe in the result that from part '01' we go to '02' which goes to '06' and so on. Further, notice that part '06' is reached twice, once through '01' directly and another time through '02'. In the output, however, its subcomponents are listed only once (this is the result of using a SELECT DISTINCT) as required.

It is important to remember that with recursive common table expressions it is possible to introduce an *infinite loop*. In this example, an infinite loop would be created if the search condition of the second operand that joins the parent and child tables was coded as:

```
PARENT.SUBPART = CHILD.SUBPART
```

This example of causing an infinite loop is obviously a case of not coding what is intended. However, care should also be exercised in determining what to code so that there is a definite end of the recursion cycle.

The result produced by this example query could be produced in an application program without using a recursive common table expression. However, this approach would require starting of a new query for every level of recursion. Furthermore, the application needs to put all the results back in the database to order the result. This approach complicates the application logic and does not perform well. The application logic becomes even harder and more inefficient for other bill of material queries, such as summarized and indented explosion queries.

### *Example 2: Summarized explosion*

The second example is a summarized explosion. The question posed here is, what is the total quantity of each part required to build part '01'. The main difference from the single level explosion is the need to aggregate the quantities. The first example indicates the quantity of subparts required for the part whenever it is required. It does not indicate how many of the subparts are needed to build part '01'.

```
WITH RPL (PART, SUBPART, QUANTITY) AS
(
 SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
 FROM PARTLIST ROOT
 WHERE ROOT.PART = '01'
 UNION ALL
 SELECT PARENT.PART, CHILD.SUBPART, PARENT.QUANTITY*CHILD.QUANTITY
 FROM RPL PARENT, PARTLIST CHILD
 WHERE PARENT.SUBPART = CHILD.PART
)
```

```

SELECT PART, SUBPART, SUM(QUANTITY) AS "Total QTY Used"
FROM RPL
GROUP BY PART, SUBPART
ORDER BY PART, SUBPART;

```

In the above query, the select list of the second operand of the UNION in the recursive common table expression, identified by the name *RPL*, shows the aggregation of the quantity. To find out how much of a subpart is used, the quantity of the parent is multiplied by the quantity per parent of a child. If a part is used multiple times in different places, it requires another final aggregation. This is done by the grouping over the common table expression *RPL* and using the SUM aggregate function in the select list of the main fullselect.

The result of the query is as follows:

| PART | SUBPART | Total Qty Used |
|------|---------|----------------|
| 01   | 02      | 2              |
| 01   | 03      | 3              |
| 01   | 04      | 4              |
| 01   | 05      | 14             |
| 01   | 06      | 15             |
| 01   | 07      | 18             |
| 01   | 08      | 40             |
| 01   | 09      | 44             |
| 01   | 10      | 140            |
| 01   | 11      | 140            |
| 01   | 12      | 294            |
| 01   | 13      | 150            |
| 01   | 14      | 144            |

Looking at the output, consider the line for subpart '06'. The total quantity used value of 15 is derived from a quantity of 3 directly for part '01' and a quantity of 6 for part '02' which is needed 2 times by part '01'.

### Example 3: Controlling depth

The question may come to mind, what happens when there are more levels of parts in the table than you are interested in for your query? That is, how is a query written to answer the question, "What are the first two levels of parts needed to build the part identified by '01'?" For the sake of clarity in the example, the level is included in the result.

```

WITH RPL (LEVEL, PART, SUBPART, QUANTITY) AS
(
 SELECT 1, ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
 FROM PARTLIST ROOT
 WHERE ROOT.PART = '01'
 UNION ALL
 SELECT PARENT.LEVEL+1, CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
 FROM RPL PARENT, PARTLIST CHILD
 WHERE PARENT.SUBPART = CHILD.PART
 AND PARENT.LEVEL < 2
)
SELECT PART, LEVEL, SUBPART, QUANTITY
FROM RPL;

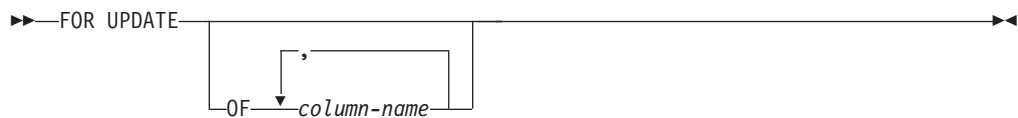
```

This query is similar to example 1. The column *LEVEL* was introduced to count the levels from the original part. In the initialization fullselect, the value for the *LEVEL* column is initialized to 1. In the subsequent fullselect, the level from the parent is incremented by 1. Then to control the number of levels in the result, the second fullselect includes the condition that the parent level must be less than 2. This ensures that the second fullselect only processes children to the second level.

The result of the query is:

| PART | LEVEL | SUBPART | QUANTITY |
|------|-------|---------|----------|
| 01   |       | 1 02    | 2        |
| 01   |       | 1 03    | 3        |
| 01   |       | 1 04    | 4        |
| 01   |       | 1 06    | 3        |
| 02   |       | 2 05    | 7        |
| 02   |       | 2 06    | 6        |
| 03   |       | 2 07    | 6        |
| 04   |       | 2 08    | 10       |
| 04   |       | 2 09    | 11       |
| 06   |       | 2 12    | 10       |
| 06   |       | 2 13    | 10       |

### update-clause



The FOR UPDATE clause identifies the columns that can be updated in a subsequent Positioned UPDATE statement. Each *column-name* must be unqualified and must identify a column of the table or view identified in the first FROM clause of the fullselect. If the FOR UPDATE clause is specified without column names, all updatable columns of the table or view identified in the first FROM clause of the fullselect are included.

The FOR UPDATE clause cannot be used if one of the following is true:

- The cursor associated with the select-statement is not deletable .
- One of the selected columns is a non-updatable column of a catalog table and the FOR UPDATE clause has not been used to exclude that column.

### read-only-clause



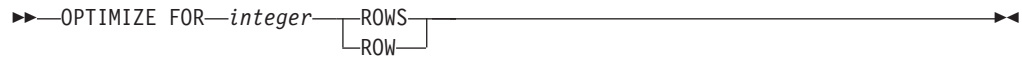
The FOR READ ONLY clause indicates that the result table is read-only and therefore the cursor cannot be referred to in Positioned UPDATE and DELETE statements. FOR FETCH ONLY has the same meaning.

Some result tables are read-only by nature. (For example, a table based on a read-only view.) FOR READ ONLY can still be specified for such tables, but the specification has no effect.

For result tables in which updates and deletes are allowed, specifying FOR READ ONLY (or FOR FETCH ONLY) can possibly improve the performance of FETCH operations by allowing the database manager to do blocking. For example, in programs that contain dynamic SQL statements without the FOR READ ONLY or ORDER BY clause, the database manager might open cursors as if the FOR UPDATE clause were specified. It is recommended, therefore, that the FOR READ ONLY clause be used to improve performance, except in cases where queries will be used in positioned UPDATE or DELETE statements.

A read-only result table must not be referred to in a Positioned UPDATE or DELETE statement, whether it is read-only by nature or specified as FOR READ ONLY (FOR FETCH ONLY).

### optimize-for-clause

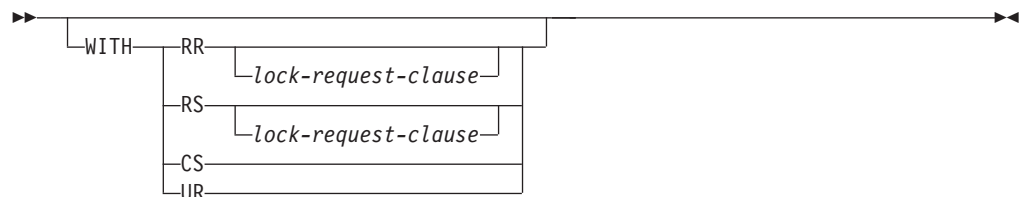


The OPTIMIZE FOR clause requests special processing of the *select statement*. If the clause is omitted, it is assumed that all rows of the result table will be retrieved; if it is specified, it is assumed that the number of rows retrieved will probably not exceed *n*, where *n* is the value of *integer*. The value of *n* must be a positive integer. Use of the OPTIMIZE FOR clause influences query optimization, based on the assumption that *n* rows will be retrieved. In addition, for cursors that are blocked, this clause will influence the number of rows that will be returned in each block (that is, no more than *n* rows will be returned in each block). If both the *fetch-first-clause* and the *optimize-for-clause* are specified, the lower of the integer values from these clauses will be used to influence the communications buffer size. The values are considered independently for optimization purposes.

This clause does not limit the number of rows that can be fetched, or affect the result in any other way than performance. Using OPTIMIZE FOR *n* ROWS can improve performance if no more than *n* rows are retrieved, but may degrade performance if more than *n* rows are retrieved.

If the value of *n* multiplied by the size of the row exceeds the size of the communication buffer, the OPTIMIZE FOR clause will have no impact on the data buffers. The size of the communication buffer is defined by the **rqrioblk** or the **aslheapsz** configuration parameter.

### isolation-clause



The optional *isolation-clause* specifies the isolation level at which the statement is executed, and whether a specific type of lock is to be acquired.

- RR - Repeatable Read
- RS - Read Stability
- CS - Cursor Stability
- UR - Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. When a nickname is used in a *select-statement* to access data in DB2 family and Microsoft SQL Server data sources, the *isolation-clause* can be included in the statement to specify the statement isolation level. If the *isolation-clause* is included in statements that access other data sources, the specified isolation level is ignored. The current isolation level on the federated

server is mapped to a corresponding isolation level at the data source on each connection to the data source. After a connection is made to a data source, the isolation level cannot be changed for the duration of the connection.

## lock-request-clause



The optional *lock-request-clause* specifies the type of lock that the database manager is to acquire and hold:

### SHARE

Concurrent processes can acquire SHARE or UPDATE locks on the data.

### UPDATE

Concurrent processes can acquire SHARE locks on the data, but no concurrent process can acquire an UPDATE or EXCLUSIVE lock.

### EXCLUSIVE

Concurrent processes cannot acquire a lock on the data.

The *lock-request-clause* applies to all base table and index scans required by the query, including those within subqueries, SQL functions and SQL methods. It has no affect on locks placed by procedures, external functions, or external methods. Any SQL function or SQL method invoked (directly or indirectly) by the statement must be created with INHERIT ISOLATION LEVEL WITH LOCK REQUEST (SQLSTATE 42601). The *lock-request-clause* cannot be used with a modifying query that might invoke triggers or that requires referential integrity checks (SQLSTATE 42601).

## Examples of a select-statement

*Example 1:* Select all columns and rows from the EMPLOYEE table.

```
SELECT * FROM EMPLOYEE
```

*Example 2:* Select the project name (PROJNAME), start date (PRSTDATE), and end date (PRENDATE) from the PROJECT table. Order the result table by the end date with the most recent dates appearing first.

```
SELECT PROJNAME, PRSTDATE, PRENDATE
FROM PROJECT
ORDER BY PRENDATE DESC
```

*Example 3:* Select the department number (WORKDEPT) and average departmental salary (SALARY) for all departments in the EMPLOYEE table. Arrange the result table in ascending order by average departmental salary.

```
SELECT WORKDEPT, AVG(SALARY)
FROM EMPLOYEE
GROUP BY WORKDEPT
ORDER BY 2
```

*Example 4:* Declare a cursor named UP\_CUR to be used in a C program to update the start date (PRSTDATE) and the end date (PRENDATE) columns in the PROJECT table. The program must receive both of these values together with the project number (PROJNO) value for each row.

```
EXEC SQL DECLARE UP_CUR CURSOR FOR
 SELECT PROJNO, PRSTDATE, PRENDATE
 FROM PROJECT
 FOR UPDATE OF PRSTDATE, PRENDATE;
```

*Example 5:* This example names the expression SAL+BONUS+COMM as TOTAL\_PAY

```
SELECT SALARY+BONUS+COMM AS TOTAL_PAY
FROM EMPLOYEE
ORDER BY TOTAL_PAY
```

*Example 6:* Determine the employee number and salary of sales representatives along with the average salary and head count of their departments. Also, list the average salary of the department with the highest average salary.

Using a common table expression for this case saves the overhead of creating the DINFO view as a regular view. During statement preparation, accessing the catalog for the view is avoided and, because of the context of the rest of the fullselect, only the rows for the department of the sales representatives need to be considered by the view.

```
WITH
 DINFO (DEPTNO, AVGSALARY, EMPCOUNT) AS
 (SELECT OTHERS.WORKDEPT, AVG(OTHERS.SALARY), COUNT(*)
 FROM EMPLOYEE OTHERS
 GROUP BY OTHERS.WORKDEPT
),
 DINFOMAX AS
 (SELECT MAX(AVGSALARY) AS AVGMAX FROM DINFO)
SELECT THIS_EMP.EMPNO, THIS_EMP.SALARY,
 DINFO.AVGSALARY, DINFO.EMPCOUNT, DINFOMAX.AVGMAX
FROM EMPLOYEE THIS_EMP, DINFO, DINFOMAX
WHERE THIS_EMP.JOB = 'SALESREP'
AND THIS_EMP.WORKDEPT = DINFO.DEPTNO
```

*Example 7:* Given two tables, EMPLOYEE and PROJECT, replace employee SALLY with a new employee GEORGE, assign all projects lead by SALLY to GEORGE, and return the names of the updated projects.

```
WITH
 NEWEMP AS (SELECT EMPNO FROM NEW TABLE
 (INSERT INTO EMPLOYEE(EMPNO, FIRSTNAME)
 VALUES(NEXT VALUE FOR EMPNO_SEQ, 'GEORGE'))),
 OLDEMP AS (SELECT EMPNO FROM EMPLOYEE WHERE FIRSTNAME = 'SALLY'),
 UP PROJ AS (SELECT PROJNAME FROM NEW TABLE
 (UPDATE PROJECT
 SET RESPEMP = (SELECT EMPNO FROM NEWEMP)
 WHERE RESPEMP = (SELECT EMPNO FROM OLDEMP))),
 DELEMP AS (SELECT EMPNO FROM OLD TABLE
 (DELETE FROM EMPLOYEE
 WHERE EMPNO = (SELECT EMPNO FROM OLDEMP)))
SELECT PROJNAME FROM UP PROJ;
```

*Example 8:* Retrieve data from the DEPT table. That data will later be updated with a searched update, and should be locked when the query executes.

```
SELECT DEPTNO, DEPTNAME, MGRNO
FROM DEPT
WHERE ADMRDEPT = 'A00'
FOR READ ONLY WITH RS USE AND KEEP EXCLUSIVE LOCKS
```



---

## SET INTEGRITY

The SET INTEGRITY statement is used to:

- Bring one or more tables out of set integrity pending state (previously known as "check pending state") by performing required integrity processing on those tables.
- Bring one or more tables out of set integrity pending state without performing required integrity processing on those tables.
- Place one or more tables in set integrity pending state.
- Place one or more tables into full access state.
- Prune the contents of one or more staging tables.

When the statement is used to perform integrity processing for a table after it has been loaded or attached, the system can incrementally process the table by checking only the appended portion for constraints violations. If the subject table is a materialized query table or a staging table, and load, attach, or detach operations are performed on its underlying tables, the system can incrementally refresh the materialized query table or incrementally propagate to the staging table with only the delta portions of its underlying tables. However, there are some situations in which the system will not be able to perform such optimizations and will instead perform full integrity processing to ensure data integrity. Full integrity processing is done by checking the entire table for constraints violations, recomputing a materialized query table's definition, or marking a staging table as inconsistent. The latter implies that a full refresh of its associated materialized query table is required. There is also a situation in which you might want to explicitly request incremental processing by specifying the INCREMENTAL option.

The SET INTEGRITY statement is under transaction control.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges required to execute the SET INTEGRITY statement depend on the purpose, as outlined below.

- Bringing tables out of set integrity pending state and performing the required integrity processing.

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on:
  - The tables on which integrity processing is performed and, if exception tables are provided for one or more of those tables, INSERT privilege on the exception tables
  - All descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables that will implicitly be placed in set integrity pending state by the statement
- LOAD authority (with conditions). The following conditions must all be met before LOAD authority can be considered as providing valid privileges:

- The required integrity processing does not involve the following actions:
  - Refreshing a materialized query table
  - Propagating to a staging table
  - Updating a generated or identity column
- If exception tables are provided for one or more tables, the required access is granted for the duration of the integrity processing to the tables on which integrity processing is performed, and to the associated exception tables. That is:
  - SELECT and DELETE privilege on each table on which integrity processing is performed, and
  - INSERT privilege on the exception tables

- DATAACCESS authority

- Bringing tables out of set integrity pending state without performing the required integrity processing.

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the tables that are being processed; CONTROL privilege on each descendent foreign key table, descendent immediate materialized query table, and descendent immediate staging table that will implicitly be placed in set integrity pending state by the statement

- LOAD authority

- DATAACCESS authority

- DBADM authority

- Placing tables in set integrity pending state.

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on:

- The specified tables, and

- The descendent foreign key tables that will be placed in set integrity pending state by the statement, and

- The descendent immediate materialized query tables that will be placed in set integrity pending state by the statement, and

- The descendent immediate staging tables that will be placed in set integrity pending state by the statement

- LOAD authority

- DATAACCESS authority

- DBADM authority

- Place a table into the full access state.

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the tables that are placed into the full access state

- LOAD authority

- DATAACCESS authority

- DBADM authority

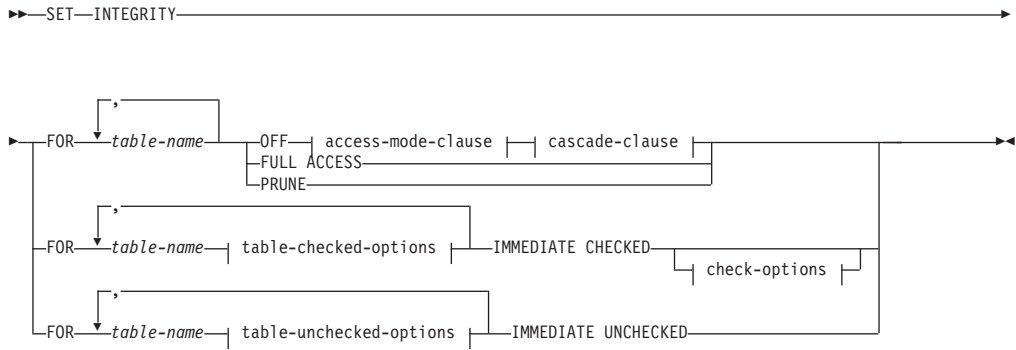
- Prune a staging table.

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the table being pruned

– DATAACCESS authority

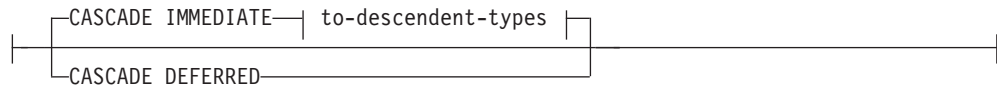
## Syntax



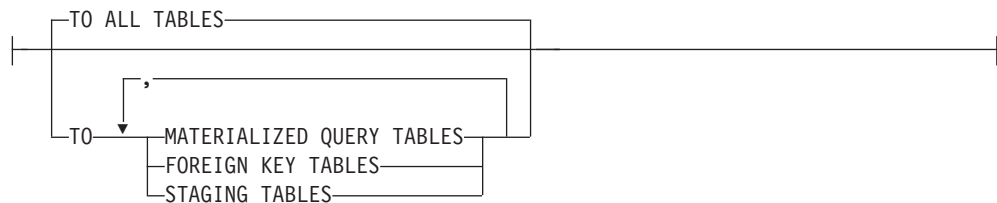
### access-mode-clause:



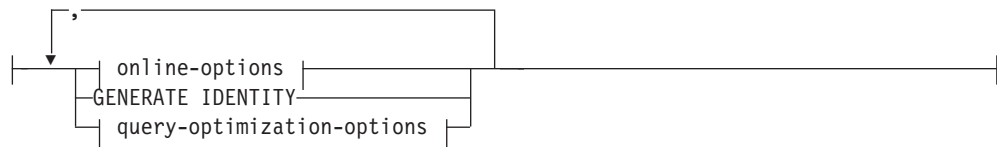
### cascade-clause:



### to-descendent-types:



### table-checked-options:



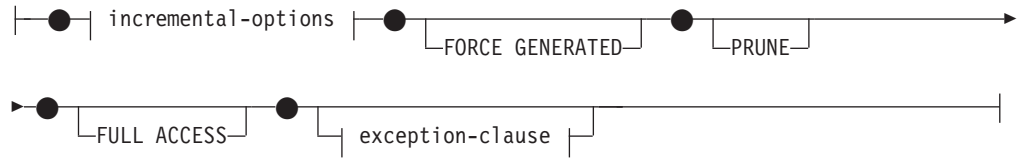
### online-options:



### query-optimization-options:



### check-options:



### incremental-options:



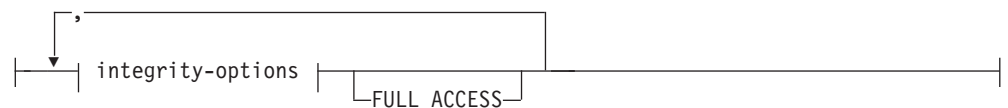
### exception-clause:



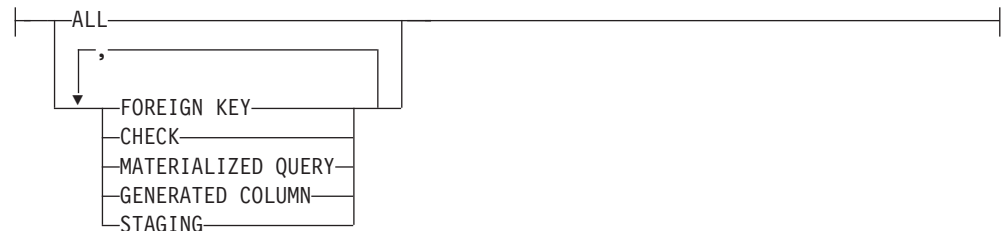
### in-table-use-clause:



### table-unchecked-options:



### integrity-options:



## Description

**FOR** *table-name*

Identifies one or more tables for integrity processing. It must be a table described in the catalog and must not be a view, catalog table, or typed table.

**OFF**

Specifies that the tables are placed in set integrity pending state. Only very limited activity is allowed on a table that is in set integrity pending state.

*access-mode-clause*

Specifies the readability of the table while it is in set integrity pending state.

**NO ACCESS**

Specifies that the table is to be put in set integrity pending no access state, which does not allow read or write access to the table.

**READ ACCESS**

Specifies that the table is to be put in set integrity pending read access state, which allows read access to the non-appended portion of the table. This option is not allowed on a table that is in set integrity pending no access state (SQLSTATE 428FH).

*cascade-clause*

Specifies whether the set integrity pending state of the table referenced in the SET INTEGRITY statement is to be immediately cascaded to descendent tables.

**CASCADE IMMEDIATE**

Specifies that the set integrity pending state is to be immediately extended to descendent tables.

*to-descendent-types*

Specifies the type of descendent tables to which the set integrity pending state is immediately cascaded.

**TO ALL TABLES**

Specifies that the set integrity pending state is to be immediately cascaded to all descendent tables of the tables in the invocation list. Descendent tables include all descendent foreign key tables, immediate staging tables, and immediate materialized query tables that are descendants of the tables in the invocation list, or descendants of descendent foreign key tables.

Specifying TO ALL TABLES is equivalent to specifying TO FOREIGN KEY TABLES, TO MATERIALIZED QUERY TABLES, and TO STAGING TABLES, all in the same statement.

**TO MATERIALIZED QUERY TABLES**

If only TO MATERIALIZED QUERY TABLES is specified, the set integrity pending state is to be immediately cascaded only to descendent immediate materialized query tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state. If both TO FOREIGN KEY TABLES and TO MATERIALIZED QUERY TABLES are specified, the set integrity pending state will be immediately cascaded to all descendent foreign key tables, all descendent immediate materialized query tables of the tables in the invocation list, and to all immediate materialized query tables that are descendants of the descendent foreign key tables.

**TO FOREIGN KEY TABLES**

Specifies that the set integrity pending state is to be immediately cascaded to descendent foreign key tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state.

## TO STAGING TABLES

Specifies that the set integrity pending state is to be immediately cascaded to descendent staging tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state. If both TO FOREIGN KEY TABLES and TO STAGING TABLES are specified, the set integrity pending state will be immediately cascaded to all descendent foreign key tables, all descendent immediate staging tables of the tables in the invocation list, and to all immediate staging tables that are descendants of the descendent foreign key tables.

## CASCADE DEFERRED

Specifies that only the tables in the invocation list are to be put in set integrity pending state. The states of the descendent tables will remain unchanged. Descendent foreign key tables might later be implicitly put in set integrity pending state when their parent tables are checked for constraints violations. Descendent immediate materialized query tables and descendent immediate staging tables might be implicitly put in set integrity pending state when one of their underlying tables is checked for integrity violations.

If *cascade-clause* is not specified, the set integrity pending state is immediately cascaded to all descendent tables.

## IMMEDIATE CHECKED

Specifies that the table is to be taken out of set integrity pending state by performing required integrity processing on the table. This is done in accordance with the information set in the STATUS and CONST\_CHECKED columns of the SYSCAT.TABLES catalog view. That is:

- The value in the STATUS column must be 'C' (the table is in set integrity pending state), or an error is returned (SQLSTATE 51027), unless the table is a descendent foreign key table, descendent materialized query table, or descendent staging table of a table that is specified in the list, is in set integrity pending state, and whose intermediate ancestors are also in the list.
- If the table being checked is in set integrity pending state, the value in CONST\_CHECKED indicates which integrity options are to be checked.

When the table is taken out of set integrity pending state, its descendent tables are, if necessary, put in set integrity pending state. A warning to indicate that descendent tables have been put in set integrity pending state is returned (SQLSTATE 01586).

If the table is a system-maintained materialized query table, the data is checked against the query and refreshed as necessary. (IMMEDIATE CHECKED cannot be used for user-maintained materialized query tables.) If the table is a staging table, the data is checked against its query definition and propagated as necessary.

When the integrity of a child table is checked:

- None of its parents can be in set integrity pending state, or
- Each of its parents must be checked for constraints violations in the same SET INTEGRITY statement

When an immediate materialized query table is refreshed, or deltas are propagated to a staging table:

- None of its underlying tables can be in set integrity pending state, or

- Each of its underlying tables must be checked in the same SET INTEGRITY statement

Otherwise, an error is returned (SQLSTATE 428A8).

*table-checked-options*

*online-options*

Specifies the accessibility of the table while it is being processed.

**ALLOW NO ACCESS**

Specifies that no other users can access the table while it is being processed, except if they are using the Uncommitted Read isolation level.

**ALLOW READ ACCESS**

Specifies that other users have read-only access to the table while it is being processed.

**ALLOW WRITE ACCESS**

Specifies that other users have read and write access to the table while it is being processed.

**GENERATE IDENTITY**

Specifies that if the table includes an identity column, the values are generated by the SET INTEGRITY statement. By default, when the GENERATE IDENTITY option is specified, only attached rows will have their identity column values generated by the SET INTEGRITY statement. The NOT INCREMENTAL option must be specified in conjunction with the GENERATE IDENTITY option to have the SET INTEGRITY statement generate identity column values for all rows in the table, including attached rows, loaded rows, and existing rows. If the GENERATE IDENTITY option is not specified, the current identity column values for all rows in the table are left unchanged.

*query-optimization-options*

Specifies the query optimization options for the maintenance of REFRESH DEFERRED materialized query tables.

**ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY**

Specifies that when the CURRENT REFRESH AGE special register is set to 'ANY', the maintenance of *table-name* will allow REFRESH DEFERRED materialized query tables to be used to optimize the query that maintains *table-name*. If *table-name* is not a REFRESH DEFERRED materialized query table, an error is returned (SQLSTATE 428FH). REFRESH IMMEDIATE materialized query tables are always considered during query optimization.

*check-options*

*incremental-options*

**INCREMENTAL**

Specifies the application of integrity processing on the appended portion (if any) of the table. If such a request cannot be satisfied (that is, the system detects that the whole table needs to be checked for data integrity), an error is returned (SQLSTATE 55019).

**NOT INCREMENTAL**

Specifies the application of integrity processing on the whole table. If the table is a materialized query table, the materialized query



table definition is recomputed. If the table has at least one constraint defined on it, this option causes full processing of descendent foreign key tables and descendent immediate materialized query tables. If the table is a staging table, it is set to an inconsistent state.

If the *incremental-options* clause is not specified, the system determines whether incremental processing is possible; if not, the whole table is checked.

#### **FORCE GENERATED**

If the table includes generated by expression columns, the values are computed on the basis of the expression and stored in the column. If this option is not specified, the current values are compared to the computed value of the expression, as though an equality check constraint were in effect. If the table is processed for integrity incrementally, generated columns are computed only for the appended portion.

#### **PRUNE**

This option can be specified for staging tables only. Specifies that the content of the staging table is to be pruned, and that the staging table is to be set to an inconsistent state. If any table in the *table-name* list is not a staging table, an error is returned (SQLSTATE 428FH). If the INCREMENTAL check option is also specified, an error is returned (SQLSTATE 428FH).

#### **FULL ACCESS**

Specifies that the table is to become fully accessible after the SET INTEGRITY statement executes.

When an underlying table (that has dependent immediate materialized query tables or dependent immediate staging tables) in the invocation list is incrementally processed, the underlying table is put in no data movement state, as required, after the SET INTEGRITY statement executes. When all incrementally refreshable dependent immediate materialized query tables and staging tables are taken out of set integrity pending state, the underlying table is automatically brought out of the no data movement state into the full access state. If the FULL ACCESS option is specified with the IMMEDIATE CHECKED option, the underlying table is put directly in full access state (bypassing the no data movement state). Dependent immediate materialized query tables that have not been refreshed might undergo a full recomputation in the subsequent REFRESH TABLE statement, and dependent immediate staging tables that have not had the appended portions of the table propagated to them might be flagged as inconsistent.

When an underlying table in the invocation list requires full processing, or does not have dependent immediate materialized query tables, or dependent immediate staging tables, the underlying table is put directly into full access state after the SET INTEGRITY statement executes, regardless of whether the FULL ACCESS option was specified.

*exception-clause*

#### **FOR EXCEPTION**

Specifies that any row that is in violation of a constraint being

checked is to be moved to an exception table. Even if errors are detected, the table is taken out of set integrity pending state. A warning to indicate that one or more rows have been moved to the exception tables is returned (SQLSTATE 01603).

If the FOR EXCEPTION option is not specified and any constraints are violated, only the first detected violation is returned (SQLSTATE 23514). If there is a violation in any table, all of the tables are left in set integrity pending state.

It is recommended to always use the FOR EXCEPTION option when checking for constraints violations to prevent a rollback of the SET INTEGRITY statement if a violation is found.

**IN** *table-name*

Specifies the table from which rows that violate constraints are to be moved. There must be one exception table specified for each table being checked. This clause cannot be specified for a materialized query table or a staging table (SQLSTATE 428A7).

**USE** *table-name*

Specifies the exception table into which error rows are to be moved.

**FULL ACCESS**

If the FULL ACCESS option is specified as the only operation of the statement, the table is placed into the full access state without being rechecked for integrity violations. However, dependent immediate materialized query tables that have not been refreshed might require a full recomputation in subsequent REFRESH TABLE statements, and dependent immediate staging tables that have not had the delta portions of the table propagated to them might be changed to incomplete state. This option can only be specified for a table that is in the no data movement state or the no access state, but not in the set integrity pending state (SQLSTATE 428FH).

**PRUNE**

This option can be specified for staging tables only. Specifies that the content of the staging table is to be pruned, and that the staging table is to be set to an inconsistent state. If any table in the *table-name* list is not a staging table, an error is returned (SQLSTATE 428FH).

*table-unchecked-options*

*integrity-options*

Used to define the types of required integrity processing that are to be bypassed when the table is taken out of the set integrity pending state.

**ALL**

The table will be immediately taken out of set integrity pending state without any of its required integrity processing being performed.

**FOREIGN KEY**

Required foreign key constraints checking will not be performed when the table is brought out of set integrity pending state.

**CHECK**

Required check constraints checking will not be performed when the table is brought out of set integrity pending state.

**MATERIALIZED QUERY**

Required refreshing of a materialized query table will not be performed when the table is brought out of set integrity pending state.

### **GENERATED COLUMN**

Required generated column constraints checking will not be performed when the table is brought out of set integrity pending state.

### **STAGING**

Required propagation of data to a staging table will not be performed when the table is brought out of set integrity pending state.

If no other types of integrity processing are required on the table after a specific type of integrity processing has been marked as bypassed, the table is immediately taken out of set integrity pending state.

### **FULL ACCESS**

Specifies that the tables are to become fully accessible after the SET INTEGRITY statement executes.

When an underlying table in the invocation list is incrementally processed, and it has dependent immediate materialized query tables or dependent immediate staging tables, the underlying table is placed, as required, in the no data movement state after the SET INTEGRITY statement executes. When all incrementally refreshable dependent immediate materialized query tables and staging tables have been taken out of set integrity pending state, the underlying table is automatically brought out of the no data movement state into the full access state. If the FULL ACCESS option is specified with the IMMEDIATE UNCHECKED option, the underlying table is placed directly in full access state (it bypasses the no data movement state). Dependent immediate materialized query tables that have not been refreshed might undergo a full recomputation in the subsequent REFRESH TABLE statement, and dependent immediate staging tables that have not had the appended portions of the table propagated to them might be flagged as inconsistent.

When an underlying table in the invocation list requires full processing, or does not have dependent immediate materialized query tables, or dependent immediate staging tables, the underlying table is placed directly in full access state after the SET INTEGRITY statement executes, regardless of whether the FULL ACCESS option has been specified.

If the FULL ACCESS option has been specified with the IMMEDIATE UNCHECKED option, and the statement does not bring the table out of set integrity pending state, an error is returned (SQLSTATE 428FH).

### **IMMEDIATE UNCHECKED**

Specifies one of the following:

- The table is to be brought out of set integrity pending state immediately without any required integrity processing.
- The table is to have one or more types of required integrity processing bypassed when the table is brought out of set integrity pending state by a subsequent SET INTEGRITY statement using the IMMEDIATE CHECKED option.

Consider the data integrity implications of this option before using it. See the “Notes” section below.

### **Notes**

- Effects on tables in one of the restricted set integrity-related states:
  - Use of INSERT, UPDATE, or DELETE is disallowed on a table that is in read access state or in no access state. Furthermore, any statement that requires

this type of modification to a table that is in such a state will be rejected. For example, deletion of a row in a parent table that cascades to a dependent table that is in the no access state is not allowed.

- Use of SELECT is disallowed on a table that is in the no access state. Furthermore, any statement that requires read access to a table that is in the no access state will be rejected.
- New constraints added to a table are normally enforced immediately. However, if the table is in set integrity pending state, the checking of any new constraints is deferred until the table is taken out of set integrity pending state. If the table is in set integrity pending state, addition of a new constraint places the table into set integrity pending no access state, because validity of data is at risk.
- The CREATE INDEX statement cannot reference any table that is in read access state or in no access state. Similarly, an ALTER TABLE statement to add a primary key or a unique constraint cannot reference any table that is in read access state or in no access state.
- The import utility is not allowed to operate on a table that is in read access state or in no access state.
- The export utility is not allowed to operate on a table that is in no access state, but is allowed to operate on a table that is in read access state. If a table is in read access state, the export utility will only export the data that is in the non-appended portion.
- Operations (like REORG, REDISTRIBUTE, update distribution key, update multidimensional clustering key, update range clustering key, update table partitioning key, and so on) that might involve data movement within a table are not allowed on a table that is in any of the following states: read access, no access, or no data movement.
- The load, backup, restore, update statistics, runstats, reorgchk, list history, and rollforward utilities are allowed on a table that is in any of the following states: full access, read access, no access, or no data movement.
- The ALTER TABLE, COMMENT, DROP TABLE, CREATE ALIAS, CREATE TRIGGER, CREATE VIEW, GRANT, REVOKE, and SET INTEGRITY statements can reference a table that is in any of the following states: full access, read access, no access, or no data movement. However, they might cause the table to be put into no access state.
- Packages, views, and any other objects that depend on a table that is in no access state will return an error when the table is accessed at run time. Packages that depend on a table that is in read access state will return an error when an insert, update, or delete operation is attempted on the table at run time.

The removal of violating rows by the SET INTEGRITY statement is not a delete event. Therefore, triggers are never activated by a SET INTEGRITY statement. Similarly, updating generated columns using the FORCE GENERATED option does not activate triggers.

- Incremental processing will be used whenever the situation allows it, because it is more efficient. The INCREMENTAL option is not needed in most cases. It is needed, however, to ensure that integrity checks are indeed processed incrementally. If the system detects that full processing is needed to ensure data integrity, an error is returned (SQLSTATE 55019).
- Warning about the use of the IMMEDIATE UNCHECKED clause:
  - This clause is intended to be used by utility programs, and its use by application programs is not recommended. If there is data in the table that

does not meet the integrity specifications that were defined for the table, and the IMMEDIATE UNCHECKED option is used, incorrect query results might be returned.

The fact that the table was taken out of the set integrity pending state without performing the required integrity processing will be recorded in the catalog (the respective byte in the CONST\_CHECKED column in the SYSCAT.TABLES view will be set to 'U'). This indicates that the user has assumed responsibility for data integrity with respect to the specific constraints. This value remains unchanged until either:

- The table is put back into set integrity pending state (by referencing the table in a SET INTEGRITY statement with the OFF option), at which time 'U' values in the CONST\_CHECKED column are changed to 'W' values, indicating that the user had previously assumed responsibility for data integrity, and the system needs to verify the data.
- All unchecked constraints for the table are dropped.

The 'W' state differs from the 'N' state in that it records the fact that integrity was previously checked by the user, but not yet by the system. If the user issues the SET INTEGRITY ... IMMEDIATE CHECKED statement with the NOT INCREMENTAL option, the system rechecks the whole table for data integrity (or performs a full refresh on a materialized query table), and then changes the 'W' state to the 'Y' state. If IMMEDIATE UNCHECKED is specified, or if NOT INCREMENTAL is not specified, the 'W' state is changed back to the 'U' state to record the fact that some data has still not been verified by the system. In the latter case (when the NOT INCREMENTAL is not specified), a warning is returned (SQLSTATE 01636).

If an underlying table's integrity has been checked using the IMMEDIATE UNCHECKED clause, the 'U' values in the CONST\_CHECKED column of the underlying table will be propagated to the corresponding CONST\_CHECKED column of:

- Dependent immediate materialized query tables
- Dependent deferred materialized query tables
- Dependent staging tables

For a dependent immediate materialized query table, this propagation is done whenever the underlying table is brought out of set integrity pending state, and whenever the materialized query table is refreshed. For a dependent deferred materialized query table, this propagation is done whenever the materialized query table is refreshed. For dependent staging tables, this propagation is done whenever the underlying table is brought out of set integrity pending state. These propagated 'U' values in the CONST\_CHECKED columns of dependent materialized query tables and staging tables record the fact that these materialized query tables and staging tables depend on some underlying table whose required integrity processing has been bypassed using the IMMEDIATE UNCHECKED option.

For a materialized query table, the 'U' value in the CONST\_CHECKED column that was propagated by the underlying table will remain until the materialized query table is fully refreshed and none of its underlying tables have a 'U' value in their corresponding CONST\_CHECKED column. After such a refresh, the 'U' value in the CONST\_CHECKED column for the materialized query table will be changed to 'Y'.

For a staging table, the 'U' value in the CONST\_CHECKED column that was propagated by the underlying table will remain until the corresponding

deferred materialized query table of the staging table is refreshed. After such a refresh, the 'U' value in the CONST\_CHECKED column for the staging table will be changed to 'Y'.

- If a child table and its parent table are checked in the same SET INTEGRITY statement with the IMMEDIATE CHECKED option, and the parent table requires full checking of its constraints, the child table will have its foreign key constraints checked, independently of whether or not the child table has a 'U' value in the CONST\_CHECKED column for foreign key constraints.
- After appending data using LOAD INSERT or ALTER TABLE ATTACH, the SET INTEGRITY statement with the IMMEDIATE CHECKED option checks the table for constraints violations. The system determines whether incremental processing on the table is possible. If so, only the appended portion is checked for integrity violations. If not, the system checks the whole table for integrity violations.
- Consider the statement:

**SET INTEGRITY FOR T IMMEDIATE CHECKED**

Situations in which the system will require a full refresh, or will check the whole table for integrity (the INCREMENTAL option cannot be specified) are:

- When new constraints have been added to T itself while it is in the set integrity pending state
- When a LOAD REPLACE operation against T, its parents, or its underlying tables has taken place
- When the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated after the last integrity check on T, its parents, or its underlying tables
- The cascading effect of full processing, when any parent of T (or underlying table, if T is a materialized query table or a staging table) has been checked for integrity non-incrementally
- If the table space containing the table or its parent (or underlying table of a materialized query table or a staging table) has been rolled forward to a point in time, and the table and its parent (or underlying table if the table is a materialized query table or a staging table) reside in different table spaces
- When T is a materialized query table, and a LOAD REPLACE or LOAD INSERT operation directly into T has taken place after the last refresh
- If the conditions for full processing described in the previous bullet are not satisfied, the system will attempt to check only the appended portion for integrity, or perform an incremental refresh (if it is a materialized query table) when the user does not specify the NOT INCREMENTAL option for the statement SET INTEGRITY FOR T IMMEDIATE CHECKED.
- If an error occurs during integrity processing, all the effects of the processing (including deleting from the original and inserting into the exception tables) will be rolled back.
- If a SET INTEGRITY statement issued with the FORCE GENERATED option fails because of a lack of log space, increase available active log space and reissue the SET INTEGRITY statement. Alternatively, use the SET INTEGRITY statement with the GENERATED COLUMN and IMMEDIATE UNCHECKED options to bypass generated column checking for the table. Then, issue a SET INTEGRITY statement with the IMMEDIATE CHECKED option and without the FORCE GENERATED option to check the table for other integrity violations (if applicable) and to bring it out of set integrity pending state. After the table is out of the set integrity pending state, the generated columns can be updated to their default (generated) values by assigning them to the keyword DEFAULT in



an UPDATE statement. This is accomplished by using either multiple searched update statements based on ranges (each followed by a commit), or a cursor-based approach using intermittent commits. A “with hold” cursor should be used if locks are to be retained after intermittent commits using the cursor-based approach.

- A table that was put into set integrity pending state using the CASCADE DEFERRED option of the SET INTEGRITY statement or the LOAD command, or through the ALTER TABLE statement with the ATTACH clause, and that is checked for integrity violations using the IMMEDIATE CHECKED option of the SET INTEGRITY statement, will have its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables put in set integrity pending state, as required:
  - If the entire table is checked for integrity violations, its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables will be put in set integrity pending state.
  - If the table is checked for integrity violations incrementally, its descendent immediate materialized query tables and staging tables will be put in set integrity pending state, and its descendent foreign key tables will remain in their original states.
  - If the table requires no checking at all, its descendent immediate materialized query tables, descendent staging tables, and descendent foreign key tables will remain in their original states.
- A table that was put in set integrity pending state using the CASCADE DEFERRED option (of the SET INTEGRITY statement or the LOAD command), and that is brought out of set integrity pending state using the IMMEDIATE UNCHECKED option of the SET INTEGRITY statement, will have its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables put in set integrity pending state, as required:
  - If the table has been loaded using the REPLACE mode, its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables will be put in set integrity pending state.
  - If the table has been loaded using the INSERT mode, its descendent immediate materialized query tables and staging tables will be put in set integrity pending state, and its descendent foreign key tables will remain in their original states.
  - If the table has not been loaded, its descendent immediate materialized query tables, descendent staging tables, and its descendent foreign key tables will remain in their original states.
- SET INTEGRITY is usually a long running statement. In light of this, to reduce the risk of a rollback of the entire statement because of a lock timeout, you can issue the SET CURRENT LOCK TIMEOUT statement with the WAIT option before executing the SET INTEGRITY statement, and then reset the special register to its previous value after the transaction commits. Note, however, that the CURRENT LOCK TIMEOUT special register only impacts a specific set of lock types.
- If you use the ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY option, ensure that the maintenance order is correct for REFRESH DEFERRED materialized query tables. For example, consider two materialized query tables, MQT1 and MQT2, whose materialized queries share the same underlying tables. The materialized query for MQT2 can be calculated using MQT1, instead of the underlying tables. If separate statements are used to maintain these two materialized query tables, and MQT2



is maintained first, the system might choose to use the contents of MQT1, which has not yet been maintained, to maintain MQT2. In this case, MQT1 would contain current data, but MQT2 could still contain stale data, even though both were maintained at almost the same time. The correct maintenance order, if two SET INTEGRITY statements are used instead of one, is to maintain MQT1 first.

- When using the SET INTEGRITY statement to perform integrity processing on a base table that has been loaded or attached, it is recommended that you process its dependent REFRESH IMMEDIATE materialized query tables and its PROPAGATE IMMEDIATE staging tables in the same SET INTEGRITY statement to avoid putting these dependent tables in set integrity pending no access state at the end of SET INTEGRITY processing. Note that for base tables that have a large number of dependent REFRESH IMMEDIATE materialized query tables and PROPAGATE IMMEDIATE staging tables, memory constraints might make it impossible to process all of the dependents in the same statement as the base table.
- If the FORCE GENERATED or the GENERATE IDENTITY option is specified, and the column that is generated is part of a unique index, the SET INTEGRITY statement returns an error (SQLSTATE 23505) and rolls back if it detects duplicate keys in the unique index. This error is returned even if there is an exception table for the table being processed.

This scenario can occur under the following circumstances:

- The SET INTEGRITY statement runs after a LOAD command against the table, and the GENERATEDOVERRIDE or the IDENTITYOVERRIDE file type modifier is specified during the load operation. To prevent this scenario, it is recommended that you use the GENERATEDIGNORE or the GENERATEDMISSING file type modifier instead of GENERATEDOVERRIDE, and that you use the IDENTITYIGNORE or the IDENTITYMISSING modifier instead of IDENTITYOVERRIDE. Using the recommended modifiers will prevent the need for any generated by expression column or identity column processing during SET INTEGRITY statement execution.
- The SET INTEGRITY statement is run after an ALTER TABLE statement that alters the expression of a generated by expression column.

To bring a table out of the set integrity pending state after encountering such a scenario:

- Do not use the FORCE GENERATED or the GENERATE IDENTITY option to regenerate the column values. Instead, use the IMMEDIATE CHECKED option in conjunction with the FOR EXCEPTION option to move any rows that violate the generated column expression to an exception table. Then, re-insert the rows into the table from the exception table, which will generate the correct expression and perform unique key checking. This prevents having to reprocess the entire table, because only those rows that violated the generated column expression will need to be processed again.
- If the table being processed has attached partitions, detach those partitions before performing the actions that are described in the previous bullet. Then, re-attach the partitions and execute a SET INTEGRITY statement to process integrity on the attached partitions separately.
- If a protected table is specified for the SET INTEGRITY statement along with an exception table, all of the following table criteria must be met; otherwise, an error is returned (SQLSTATE 428A5):
  - The tables must be protected by the same security policy.
  - If a column in the protected table has data type DB2SECURITYLABEL, the corresponding column in the exception table must also have data type DB2SECURITYLABEL.

- If a column in the protected table is protected by a security label, the corresponding column in the exception table must also be protected by the same security label.
- **Compatibilities:** For compatibility with previous versions of DB2:
  - SET CONSTRAINTS can be specified in place of SET INTEGRITY
  - SUMMARY can be specified in place of MATERIALIZED QUERY

## Examples

*Example 1:* The following is an example of a query that provides information about the set integrity pending state and the set integrity-related access restriction states of tables. SUBSTR is used to extract individual bytes of the CONST\_CHECKED column of SYSCAT.TABLES. The first byte represents foreign key constraints; the second byte represents check constraints; the fifth byte represents materialized query table integrity; the sixth byte represents generated column constraints; the seventh byte represents staging table integrity; and the eighth byte represents data partitioning constraints. STATUS gives the set integrity pending state, and ACCESS\_MODE gives the set integrity-related access restriction state.

```
SELECT TABNAME, STATUS, ACCESS_MODE,
 SUBSTR(CONST_CHECKED,1,1) AS FK_CHECKED,
 SUBSTR(CONST_CHECKED,2,1) AS CC_CHECKED,
 SUBSTR(CONST_CHECKED,5,1) AS MQT_CHECKED,
 SUBSTR(CONST_CHECKED,6,1) AS GC_CHECKED,
 SUBSTR(CONST_CHECKED,7,1) AS STG_CHECKED,
 SUBSTR(CONST_CHECKED,8,1) AS DP_CHECKED
FROM SYSCAT.TABLES
```

*Example 2:* Put the PARENT table in set integrity pending no access state, and immediately cascade the set integrity pending state to its descendants.

```
SET INTEGRITY FOR PARENT OFF
NO ACCESS CASCADE IMMEDIATE
```

*Example 3:* Put the PARENT table in set integrity pending read access state without immediately cascading the set integrity pending state to its descendants.

```
SET INTEGRITY FOR PARENT OFF
READ ACCESS CASCADE DEFERRED
```

*Example 4:* Check integrity for a table named FACT\_TABLE. If there are no integrity violations detected, the table is brought out of set integrity pending state. If any integrity violations are detected, the entire statement is rolled back, and the table remains in set integrity pending state.

```
SET INTEGRITY FOR FACT_TABLE IMMEDIATE CHECKED
```

*Example 5:* Check integrity for the SALES and PRODUCTS tables, and move the rows that violate integrity into exception tables named SALES\_EXCEPTIONS and PRODUCTS\_EXCEPTIONS. Both the SALES and PRODUCTS tables are brought out of set integrity pending state, whether or not there are any integrity violations.

```
SET INTEGRITY FOR SALES, PRODUCTS IMMEDIATE CHECKED
FOR EXCEPTION IN SALES USE SALES_EXCEPTIONS,
IN PRODUCTS USE PRODUCTS_EXCEPTIONS
```

*Example 6:* Enable FOREIGN KEY constraint checking in the MANAGER table, and CHECK constraint checking in the EMPLOYEE table, to be bypassed with the IMMEDIATE UNCHECKED option.

```
SET INTEGRITY FOR MANAGER FOREIGN KEY,
EMPLOYEE CHECK IMMEDIATE UNCHECKED
```

*Example 7:* Add a check constraint and a foreign key to the EMP\_ACT table, using two ALTER TABLE statements. The SET INTEGRITY statement with the OFF option is used to put the table in set integrity pending state, so that the constraints are not checked immediately upon execution of the two ALTER TABLE statements. The single SET INTEGRITY statement with the IMMEDIATE CHECKED option is used to check both of the added constraints during a single pass through the table.

```
SET INTEGRITY FOR EMP_ACT OFF;
ALTER TABLE EMP_ACT ADD CHECK
(EMSTDATE <= EMENDATE);
ALTER TABLE EMP_ACT ADD FOREIGN KEY
(EMPNO) REFERENCES EMPLOYEE;
SET INTEGRITY FOR EMP_ACT IMMEDIATE CHECKED
FOR EXCEPTION IN EMP_ACT USE EMP_ACT_EXCEPTIONS
```

*Example 8:* Update generated columns with the correct values.

```
SET INTEGRITY FOR SALES IMMEDIATE CHECKED
FORCE GENERATED
```

*Example 9:* Append (using LOAD INSERT) from different sources into an underlying table (SALES) of a REFRESH IMMEDIATE materialized query table (SALES\_SUMMARY). Check SALES incrementally for data integrity, and refresh SALES\_SUMMARY incrementally. In this scenario, integrity checking for SALES and refreshing of SALES\_SUMMARY are incremental, because the system chooses incremental processing. The ALLOW READ ACCESS option is used on the SALES table to allow concurrent reads of existing data while integrity checking of the loaded portion of the table is taking place.

```
LOAD FROM 2000_DATA.DEL OF DEL
INSERT INTO SALES ALLOW READ ACCESS;
LOAD FROM 2001_DATA.DEL OF DEL
INSERT INTO SALES ALLOW READ ACCESS;
SET INTEGRITY FOR SALES ALLOW READ ACCESS IMMEDIATE CHECKED
FOR EXCEPTION IN SALES USE SALES_EXCEPTIONS;
REFRESH TABLE SALES_SUMMARY;
```

*Example 10:* Attach a new partition to a data partitioned table named SALES. Incrementally check for constraints violations in the attached data of the SALES table and incrementally refresh the dependent SALES\_SUMMARY table. The ALLOW WRITE ACCESS option is used on both tables to allow concurrent updates while integrity checking is taking place.

```
ALTER TABLE SALES
ATTACH PARTITION STARTING (100) ENDING (200)
FROM SOURCE;
SET INTEGRITY FOR SALES ALLOW WRITE ACCESS, SALES_SUMMARY ALLOW WRITE ACCESS
IMMEDIATE CHECKED FOR EXCEPTION IN SALES
USE SALES_EXCEPTIONS;
```

*Example 11:* Detach a partition from a data partitioned table named SALES. Incrementally refresh the dependent SALES\_SUMMARY table.

```
ALTER TABLE SALES
DETACH PARTITION 2000_PART INTO ARCHIVE_TABLE;
SET INTEGRITY FOR SALES_SUMMARY
IMMEDIATE CHECKED;
```

*Example 12:* Bring a new user-maintained materialized query table out of set integrity pending state.

```
CREATE TABLE YEARLY_SALES
AS (SELECT YEAR, SUM(SALES)AS SALES
FROM FACT_TABLE GROUP BY YEAR)
```

DATA INITIALLY DEFERRED REFRESH DEFERRED MAINTAINED BY USER

SET INTEGRITY FOR YEARLY\_SALES  
ALL IMMEDIATE UNCHECKED

---

## SET SCHEMA

The SET SCHEMA statement changes the value of the CURRENT SCHEMA special register. It is not under transaction control. If the package is bound with the DYNAMICRULES BIND option, this statement does not affect the qualifier used for unqualified database object references.

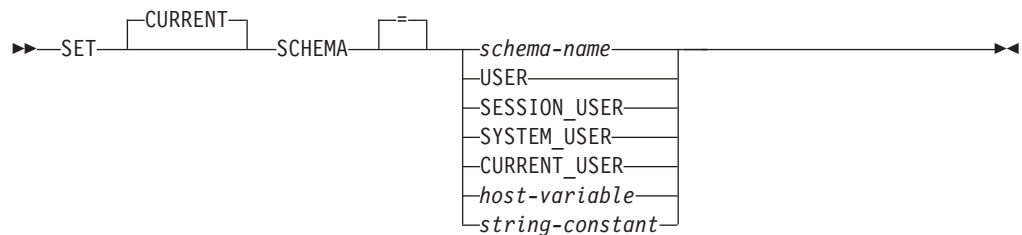
### Invocation

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

None required.

### Syntax



### Description

#### *schema-name*

This one-part name identifies a schema that exists at the application server. The length must not exceed 128 bytes (SQLSTATE 42815). No validation that the schema exists is made at the time that the schema is set. If a *schema-name* is misspelled, the error will not be caught, and that could affect the way that subsequent SQL statements execute.

#### **USER**

The value in the USER special register.

#### **SESSION\_USER**

The value in the SESSION\_USER special register.

#### **SYSTEM\_USER**

The value in the SYSTEM\_USER special register.

#### **CURRENT\_USER**

The value in the CURRENT\_USER special register.

#### *host-variable*

A variable of type CHAR or VARCHAR. The length of the contents of the *host-variable* must not exceed 128 bytes (SQLSTATE 42815). It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

The characters of the *host-variable* must be left justified. When specifying the *schema-name* with a *host-variable*, all characters must be specified in the exact case intended as there is no conversion to uppercase characters.

*string-constant*

A character string constant with a maximum length of 128 bytes.

## Rules

- If the value specified does not conform to the rules for a *schema-name*, an error (SQLSTATE 3F000) is raised.
- The value of the CURRENT SCHEMA special register is used as the schema name in all dynamic SQL statements, with the exception of the CREATE SCHEMA statement, where an unqualified reference to a database object exists.
- The QUALIFIER bind option specifies the schema name for use as the qualifier for unqualified database object names in static SQL statements.

## Notes

- The initial value of the CURRENT SCHEMA special register is equivalent to USER.
- Setting the CURRENT SCHEMA special register does not effect the CURRENT PATH special register. Hence, the CURRENT SCHEMA will not be included in the SQL path and functions, procedures and user-defined type resolution may not find these objects. To include the current schema value in the SQL path, whenever the SET SCHEMA statement is issued, also issue the SET PATH statement including the schema name from the SET SCHEMA statement.
- CURRENT SQLID is accepted as a synonym for CURRENT SCHEMA and the effect of a SET CURRENT SQLID statement will be identical to that of a SET CURRENT SCHEMA statement. No other effects, such as statement authorization changes, will occur.

## Examples

*Example 1:* The following statement sets the CURRENT SCHEMA special register.

```
SET SCHEMA RICK
```

*Example 2:* The following example retrieves the current value of the CURRENT SCHEMA special register into the host variable called CURSCHEMA.

```
EXEC SQL VALUES (CURRENT SCHEMA) INTO :CURSCHEMA;
```

The value would be RICK, set by the previous example.

---

## SQL queries

A *query* specifies a result table. A query is a component of certain SQL statements. The three forms of a query are:

- subselect
- fullselect
- select-statement.

### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- For each table or view identified in the query, one of the following:
  - SELECT privilege on the table or view
  - CONTROL privilege on the table or view
- DATAACCESS authority

For each global variable used as an expression in the query, the privileges held by the authorization ID of the statement must include one of the following:

- READ privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

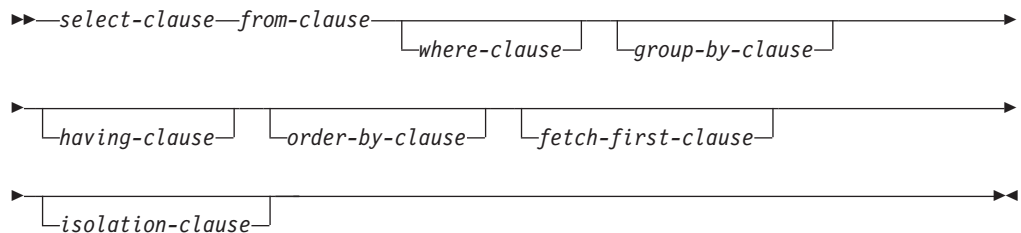
If the query contains an SQL data change statement, the authorization requirements of that statement also apply to the query.

Group privileges, with the exception of PUBLIC, are not checked for queries that are contained in static SQL statements or DDL statements.

For nicknames, authorization requirements of the data source for the object referenced by the nickname are applied when the query is processed. The authorization ID of the statement may be mapped to a different authorization ID at the data source.

---

## subselect



The *subselect* is a component of the fullselect.

A subselect specifies a result table derived from the tables, views or nicknames identified in the FROM clause. The derivation can be described as a sequence of operations in which the result of each operation is input for the next. (This is only a way of describing the subselect. The method used to perform the derivation can be quite different from this description. If portions of the subselect do not actually need to be executed for the correct result to be obtained, they might or might not be executed.)

The authorization for a *subselect* is described in the Authorization section in "SQL queries".

The clauses of the subselect are processed in the following sequence:

1. FROM clause
2. WHERE clause
3. GROUP BY clause
4. HAVING clause
5. SELECT clause
6. ORDER BY clause

## 7. FETCH FIRST clause

A subselect that contains an ORDER BY or FETCH FIRST clause cannot be specified:

- In the outermost fullselect of a view.
- In a materialized query table.
- Unless the subselect is enclosed in parenthesis.

For example, the following is not valid (SQLSTATE 428FJ):

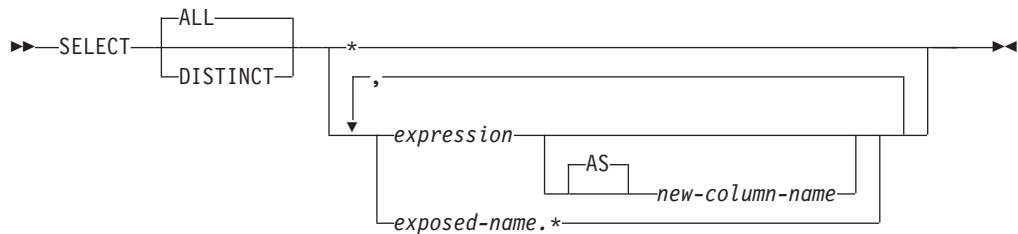
```
SELECT * FROM T1
 ORDER BY C1
UNION
SELECT * FROM T2
 ORDER BY C1
```

The following example *is* valid:

```
(SELECT * FROM T1
 ORDER BY C1)
UNION
(SELECT * FROM T2
 ORDER BY C1)
```

**Note:** An ORDER BY clause in a subselect does not affect the order of the rows returned by a query. An ORDER BY clause only affects the order of the rows returned if it is specified in the outermost fullselect.

## select-clause



The SELECT clause specifies the columns of the final result table, R. The column values are produced by the application of the *select list* to R. The select list is the names or expressions specified in the SELECT clause, and R is the result of the previous operation of the subselect. For example, if the only clauses specified are SELECT, FROM, and WHERE, R is the result of that WHERE clause.

### ALL

Retains all rows of the final result table, and does not eliminate redundant duplicates. This is the default.

### DISTINCT

Eliminates all but one of each set of duplicate rows of the final result table. If DISTINCT is used, no string column of the result table can be a LOB type, distinct type based on LOB, or structured type. DISTINCT may be used more than once in a subselect. This includes SELECT DISTINCT, the use of DISTINCT in an aggregate function of the select list or HAVING clause, and subqueries of the subselect.

Two rows are duplicates of one another only if each value in the first is equal to the corresponding value in the second. For determining duplicates, two null values are considered equal, and two different decimal floating-point



representations of the same number are considered equal. For example, -0 is equal to +0 and 2.0 is equal to 2.00. Each of the decimal floating-point special values are also considered equal: -NAN equals -NAN, -SNAN equals -SNAN, -INFINITY equals -INFINITY, INFINITY equals INFINITY, SNAN equals SNAN, and NAN equals NAN.

When the data type of a column is decimal floating-point, and multiple representations of the same number exist in the column, the particular value that is returned for a SELECT DISTINCT can be any one of the representations in the column. For more information, see “Numeric comparisons” on page 733.

For compatibility with other SQL implementations, UNIQUE can be specified as a synonym for DISTINCT.

## Select list notation

- \* Represents a list of names that identify the columns of table R, excluding any columns defined as IMPLICITLY HIDDEN. The first name in the list identifies the first column of R, the second name identifies the second column of R, and so on.

The list of names is established when the program containing the SELECT clause is bound. Hence \* (the asterisk) does not identify any columns that have been added to a table after the statement containing the table reference has been bound.

### *expression*

Specifies the values of a result column. Can be any expression that is a valid SQL language element, but commonly includes column names. Each column name used in the select list must unambiguously identify a column of R. The result type of the expression cannot be a row type (SQLSTATE 428H2).

### *new-column-name* or AS *new-column-name*

Names or renames the result column. The name must not be qualified and does not have to be unique. Subsequent usage of column-name is limited as follows:

- A *new-column-name* specified in the AS clause can be used in the order-by-clause, provided the name is unique.
- A *new-column-name* specified in the AS clause of the select list cannot be used in any other clause within the subselect (where-clause, group-by-clause or having-clause).
- A *new-column-name* specified in the AS clause cannot be used in the update-clause.
- A *new-column-name* specified in the AS clause is known outside the fullselect of nested table expressions, common table expressions and CREATE VIEW.

### *name.\**

Represents the list of names that identify the columns of the result table identified by *exposed-name*, excluding any columns defined as IMPLICITLY HIDDEN. The *exposed-name* may be a table name, view name, nickname, or correlation name, and must designate a table, view or nickname named in the FROM clause. The first name in the list identifies the first column of the table, view or nickname, the second name in the list identifies the second column of the table, view or nickname, and so on.

The list of names is established when the statement containing the SELECT clause is bound. Therefore, \* does not identify any columns that have been added to a table after the statement has been bound.

The number of columns in the result of SELECT is the same as the number of expressions in the operational form of the select list (that is, the list established when the statement is prepared), and cannot exceed 500 for a 4K page size or 1012 for an 8K, 16K, or 32K page size.

## Limitations on string columns

For limitations on the select list, see “Restrictions Using Varying-Length Character Strings”.

## Applying the select list

Some of the results of applying the select list to R depend on whether or not GROUP BY or HAVING is used. The results are described in two separate lists.

### If GROUP BY or HAVING is used

- An expression *X* (not an aggregate function) used in the select list must have a GROUP BY clause with:
  - a *grouping-expression* in which each expression or column-name unambiguously identifies a column of R (see “group-by-clause” on page 435) or
  - each column of R referenced in *X* as a separate *grouping-expression*.
- The select list is applied to each group of R, and the result contains as many rows as there are groups in R. When the select list is applied to a group of R, that group is the source of the arguments of the aggregate functions in the select list.

### If neither GROUP BY nor HAVING is used

- Either the select list must not include any aggregate functions, or each *column-name* in the select list must be specified within an aggregate function or must be a correlated column reference.
- If the select does not include aggregate functions, then the select list is applied to each row of R and the result contains as many rows as there are rows in R.
- If the select list is a list of aggregate functions, then R is the source of the arguments of the functions and the result of applying the select list is one row.

In either case the *n*th column of the result contains the values specified by applying the *n*th expression in the operational form of the select list.

## Null attributes of result columns

Result columns do not allow null values if they are derived from:

- A column that does not allow null values
- A constant
- The COUNT or COUNT\_BIG function
- A host variable that does not have an indicator variable
- A scalar function or expression that does not include an operand that allows nulls

Result columns allow null values if they are derived from:

- Any aggregate function except COUNT or COUNT\_BIG
- A column that allows null values

- A scalar function or expression that includes an operand that allows nulls
- A NULLIF function with arguments containing equal values
- A host variable that has an indicator variable, an SQL parameter, an SQL variable, or a global variable
- A result of a set operation if at least one of the corresponding items in the select list is nullable
- An arithmetic expression or view column that is derived from an arithmetic expression and the database is configured with DFT\_SQLMATHWARN set to Yes
- A scalar subselect
- A dereference operation
- A GROUPING SETS *grouping-expression*

### Names of result columns

- If the AS clause is specified, the name of the result column is the name specified on the AS clause.
- If the AS clause is not specified and a column list is specified in the correlation clause, the name of the result column is the corresponding name in the correlation column list.
- If neither an AS clause nor a column list in the correlation clause is specified and the result column is derived only from a single column (without any functions or operators), then the result column name is the unqualified name of that column.
- If neither an AS clause nor a column list in the correlation clause is specified and the result column is derived only from a single SQL variable or SQL parameter (without any functions or operators), then the result column name is the unqualified name of that SQL variable or SQL parameter.
- If neither an AS clause nor a column list in the correlation clause is specified and the result column is derived using a dereference operation, then the result column name is the unqualified name of the target column of the dereference operation.
- All other result column names are unnamed. The system assigns temporary numbers (as character strings) to these columns.

### Data types of result columns

Each column of the result of SELECT acquires a data type from the expression from which it is derived.

| When the expression is ...       | The data type of the result column is ...                                                                                                         |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| the name of any numeric column   | the same as the data type of the column, with the same precision and scale for DECIMAL columns, or the same precision for DECFLOAT columns.       |
| a constant                       | the same as the data type of the constant.                                                                                                        |
| the name of any numeric variable | the same as the data type of the variable, with the same precision and scale for DECIMAL variables, or the same precision for DECFLOAT variables. |
| the name of any string column    | the same as the data type of the column, with the same length attribute.                                                                          |

| When the expression is ...             | The data type of the result column is ...                                                                                                                                                                                                   |
|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| the name of any string variable        | the same as the data type of the variable, with the same length attribute; if the data type of the variable is not identical to an SQL data type (for example, a NUL-terminated string in C), the result column is a varying-length string. |
| the name of a datetime column          | the same as the data type of the column.                                                                                                                                                                                                    |
| the name of a user-defined type column | the same as the data type of the column.                                                                                                                                                                                                    |
| the name of a reference type column    | the same as the data type of the column.                                                                                                                                                                                                    |

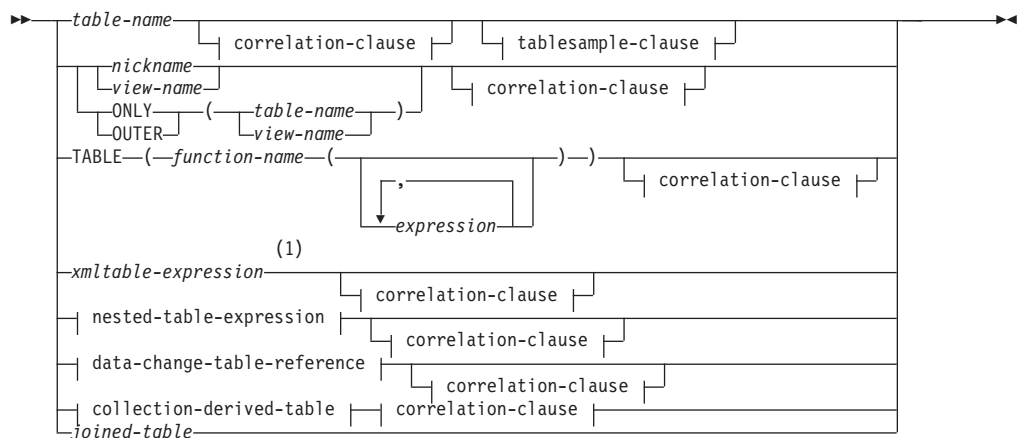
## from-clause



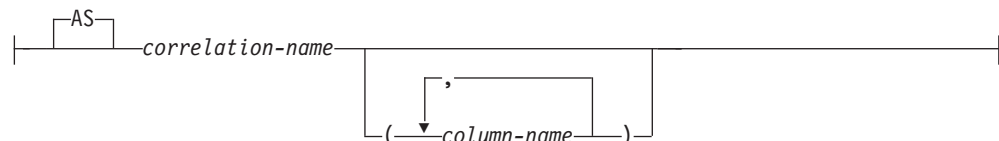
The FROM clause specifies an intermediate result table.

If only one *table-reference* is specified, the intermediate result table is simply the result of that *table-reference*. If more than one *table-reference* is specified, the intermediate result table consists of all possible combinations of the rows of the specified *table-reference* (the Cartesian product). Each row of the result is a row from the first *table-reference* concatenated with a row from the second *table-reference*, concatenated in turn with a row from the third, and so on. The number of rows in the result is the product of the number of rows in all the individual table references. For a description of *table-reference*, see “table-reference.”

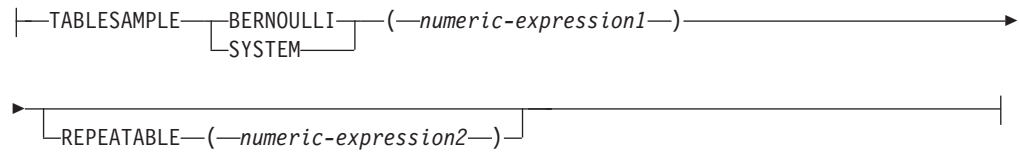
## table-reference



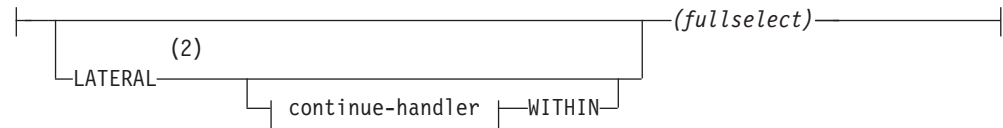
## correlation-clause:



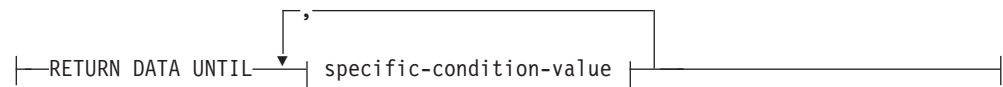
**tablesample-clause:**



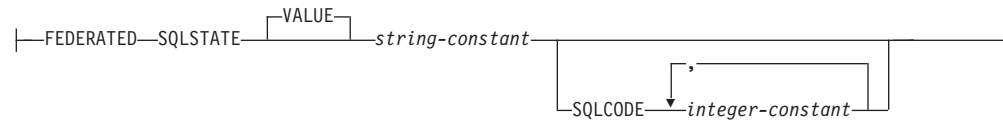
**nested-table-expression:**



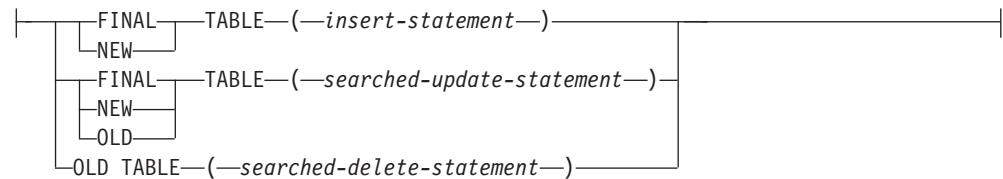
**continue-handler:**



**specific-condition-value:**



**data-change-table-reference:**



**collection-derived-table:**



**Notes:**

- 1 An XMLTABLE expression can be part of a table-reference. In this case, subexpressions within the XMLTABLE expression are in-scope of prior range variables in the FROM clause. For more information, see the description of "XMLTABLE".
- 2 TABLE can be specified in place of LATERAL.
- 3 WITH ORDINALITY can be specified only if the argument to the UNNEST

table function is one or more ordinary array variables; an associative array variable cannot be specified (SQLSTATE 428HT).

Each *table-name*, *view-name* or *nickname* specified as a table-reference must identify an existing table, view or nickname at the application server or the *table-name* of a common table expression defined preceding the fullselect containing the table-reference. If the *table-name* references a typed table, the name denotes the UNION ALL of the table with all its subtables, with only the columns of the *table-name*. Similarly, if the *view-name* references a typed view, the name denotes the UNION ALL of the view with all its subviews, with only the columns of the *view-name*.

The use of ONLY(*table-name*) or ONLY(*view-name*) means that the rows of the proper subtables or subviews are not included. If the *table-name* used with ONLY does not have subtables, then ONLY(*table-name*) is equivalent to specifying *table-name*. If the *view-name* used with ONLY does not have subviews, then ONLY(*view-name*) is equivalent to specifying *view-name*.

The use of OUTER(*table-name*) or OUTER(*view-name*) represents a virtual table. If the *table-name* or *view-name* used with OUTER does not have subtables or subviews, then specifying OUTER is equivalent to not specifying OUTER. OUTER(*table-name*) is derived from *table-name* as follows:

- The columns include the columns of *table-name* followed by the additional columns introduced by each of its subtables (if any). The additional columns are added on the right, traversing the subtable hierarchy in depth-first order. Subtables that have a common parent are traversed in creation order of their types.
- The rows include all the rows of *table-name* and all the rows of its subtables. Null values are returned for columns that are not in the subtable for the row.

The previous points also apply to OUTER(*view-name*), substituting *view-name* for *table-name* and subview for subtable.

The use of ONLY or OUTER requires the SELECT privilege on every subtable of *table-name* or subview of *view-name*.

Each *function-name* together with the types of its arguments, specified as a table reference must resolve to an existing table function at the application server.

A fullselect in parentheses is called a *nested table expression*.

A *joined-table* specifies an intermediate result set that is the result of one or more join operations. For more information, see “joined-table” on page 433.

The exposed names of all table references should be unique. An exposed name is:

- A *correlation-name*
- A *table-name* that is not followed by a *correlation-name*
- A *view-name* that is not followed by a *correlation-name*
- A *nickname* that is not followed by a *correlation-name*
- An *alias-name* that is not followed by a *correlation-name*

If a *correlation-clause* does not follow a *function-name* reference, *xmltable-expression*, nested table expression, or *data-change-table-reference*, there is no exposed name for that table reference.

Each *correlation-name* is defined as a designator of the immediately preceding *table-name*, *view-name*, *nickname*, *function-name* reference, *xmltable-expression*, nested table expression, or *data-change-table-reference*. Any qualified reference to a column must use the exposed name. If the same table name, view or nickname name is specified twice, at least one specification should be followed by a *correlation-name*. The *correlation-name* is used to qualify references to the columns of the table, view or nickname. When a *correlation-name* is specified, *column-names* can also be specified to give names to the columns of the table reference. If the *correlation-clause* does not include *column-names*, the exposed column names are determined as follows:.

- Column names of the referenced table, view, or nickname when the *table-reference* is a *table-name*, *view-name*, *nickname*, or *alias-name*
- Column names specified in the RETURNS clause of the CREATE FUNCTION statement when the *table-reference* is a *function-name* reference
- Column names specified in the COLUMNS clause of the *xmltable-expression* when the *table-reference* is an *xmltable-expression*
- Column names exposed by the fullselect when the *table-reference* is a *nested-table-expression*
- Column names from the target table of the data change statement, along with any defined INCLUDE columns when the *table-reference* is a *data-change-table-reference*

In general, collection-derived tables, table functions, and nested table expressions can be specified on any from-clause. Columns from the table functions, nested table expressions, or collection-derived tables can be referenced in the select list and in the rest of the subselect using the correlation name. The scope of this correlation name is the same as correlation names for other tables, views, or nicknames in the FROM clause. A nested table expression can be used:

- In place of a view to avoid creating the view (when general use of the view is not required)
- When the desired result table is based on host variables

A *collection-derived-table* can be used to convert the elements of an array into values of a column in separate rows. If WITH ORDINALITY is specified, an extra column of data type INTEGER is appended. This column contains the position of the element in the array. The columns can be referenced in the select list and the rest of the subselect by using the names specified for the columns in the correlation-clause. The *collection-derived-table* clause can only be used in a context where arrays are supported (SQLSTATE 42887). See the “UNNEST table function” for details.

An expression in the select list of a nested table expression that is referenced within, or is the target of, a data change statement within a fullselect is only valid when it does not include:

- A function that reads or modifies SQL data
- A function that is non-deterministic
- A function that has external action
- An OLAP function

If a view is referenced directly in, or as the target of a nested table expression in a data change statement within a FROM clause, the view must either be symmetric (have WITH CHECK OPTION specified) or satisfy the restriction for a WITH CHECK OPTION view.



If the target of a data change statement within a FROM clause is a nested table expression, the modified rows are not requalified, WHERE clause predicates are not re-evaluated, and ORDER BY or FETCH FIRST operations are not redone.

The optional *tablesample-clause* can be used to obtain a random subset (a sample) of the rows from the specified *table-name*, rather than the entire contents of that *table-name*, for this query. This sampling is in addition to any predicates that are specified in the *where-clause*. Unless the optional REPEATABLE clause is specified, each execution of the query will usually yield a different sample, except in degenerate cases where the table is so small relative to the sample size that any sample must return the same rows. The size of the sample is controlled by the *numeric-expression1* in parentheses, representing an approximate percentage (P) of the table to be returned. The method by which the sample is obtained is specified after the TABLESAMPLE keyword, and can be either BERNOULLI or SYSTEM. For both methods, the exact number of rows in the sample may be different for each execution of the query, but on average should be approximately P percent of the table, before any predicates further reduce the number of rows.

The *table-name* must be a stored table. It can be a materialized query table (MQT) name, but not a subselect or table expression for which an MQT has been defined, because there is no guarantee that the database manager will route to the MQT for that subselect.

Semantically, sampling of a table occurs before any other query processing, such as applying predicates or performing joins. Repeated accesses of a sampled table within a single execution of a query (such as in a nested-loop join or a correlated subquery) will return the same sample. More than one table may be sampled in a query.

BERNOULLI sampling considers each row individually. It includes each row in the sample with probability  $P/100$  (where P is the value of *numeric-expression1*), and excludes each row with probability  $1 - P/100$ , independently of the other rows. So if the *numeric-expression1* evaluated to the value 10, representing a ten percent sample, each row would be included with probability 0.1, and excluded with probability 0.9.

SYSTEM sampling permits the database manager to determine the most efficient manner in which to perform the sampling. In most cases, SYSTEM sampling applied to a *table-name* means that each page of *table-name* is included in the sample with probability  $P/100$ , and excluded with probability  $1 - P/100$ . All rows on each page that is included qualify for the sample. SYSTEM sampling of a *table-name* generally executes much faster than BERNOULLI sampling, because fewer data pages need to be retrieved; however, it can often yield less accurate estimates for aggregate functions (SUM(SALES), for example), especially if the rows of *table-name* are clustered on any columns referenced in that query. The optimizer may in certain circumstances decide that it is more efficient to perform SYSTEM sampling as if it were BERNOULLI sampling, for example when a predicate on *table-name* can be applied by an index and is much more selective than the sampling rate P.

The *numeric-expression1* specifies the size of the sample to be obtained from *table-name*, expressed as a percentage. It must be a constant numeric expression that cannot contain columns. The expression must evaluate to a positive number that is less than or equal to 100, but can be between 1 and 0. For example, a value of 0.01 represents one one-hundredth of a percent, meaning that 1 row in 10 000 would be sampled, on average. A *numeric-expression1* that evaluates to 100 is

handled as if the `tablesample`-clause were not specified. If *numeric-expression1* evaluates to the null value, or to a value that is greater than 100 or less than 0, an error is returned (SQLSTATE 2202H).

It is sometimes desirable for sampling to be repeatable from one execution of the query to the next; for example, during regression testing or query "debugging". This can be accomplished by specifying the `REPEATABLE` clause. The `REPEATABLE` clause requires the specification of a *numeric-expression2* in parentheses, which serves the same role as the seed in a random number generator. Adding the `REPEATABLE` clause to the `tablesample`-clause of any *table-name* ensures that repeated executions of that query (using the same value for *numeric-expression2*) return the same sample, assuming, of course, that the data itself has not been updated, reorganized, or repartitioned. To guarantee that the same sample of *table-name* is used across multiple queries, use of a global temporary table is recommended. Alternatively, the multiple queries could be combined into one query, with multiple references to a sample that is defined using the `WITH` clause.

Following are some examples:

*Example 1:* Request a 10% Bernoulli sample of the Sales table for auditing purposes.

```
SELECT * FROM Sales
TABLESAMPLE BERNOULLI(10)
```

*Example 2:* Compute the total sales revenue in the Northeast region for each product category, using a random 1% SYSTEM sample of the Sales table. The semantics of `SUM` are for the sample itself, so to extrapolate the sales to the entire Sales table, the query must divide that `SUM` by the sampling rate (0.01).

```
SELECT SUM(Sales.Revenue) / (0.01)
FROM Sales TABLESAMPLE SYSTEM(1)
WHERE Sales.RegionName = 'Northeast'
GROUP BY Sales.ProductCategory
```

*Example 3:* Using the `REPEATABLE` clause, modify the previous query to ensure that the same (yet random) result is obtained each time the query is executed. (The value of the constant enclosed by parentheses is arbitrary.)

```
SELECT SUM(Sales.Revenue) / (0.01)
FROM Sales TABLESAMPLE SYSTEM(1) REPEATABLE(3578231)
WHERE Sales.RegionName = 'Northeast'
GROUP BY Sales.ProductCategory
```

## Table function references

In general, a table function, together with its argument values, can be referenced in the `FROM` clause of a `SELECT` in exactly the same way as a table or view. There are, however, some special considerations which apply.

- Table Function Column Names

Unless alternate column names are provided following the *correlation-name*, the column names for the table function are those specified in the `RETURNS` clause of the `CREATE FUNCTION` statement. This is analogous to the names of the columns of a table, which are defined in the `CREATE TABLE` statement.

- Table Function Resolution

The arguments specified in a table function reference, together with the function name, are used by an algorithm called *function resolution* to determine the exact function to be used. This is no different from what happens with other functions (such as scalar functions) that are used in a statement.

- Table Function Arguments

As with scalar function arguments, table function arguments can in general be any valid SQL expression. The following examples are valid syntax:

```
Example 1: SELECT c1
 FROM TABLE(tf1('Zachary')) AS z
 WHERE c2 = 'FLORIDA';
```

```
Example 2: SELECT c1
 FROM TABLE(tf2 (:hostvar1, CURRENT DATE)) AS z;
```

```
Example 3: SELECT c1
 FROM t
 WHERE c2 IN
 (SELECT c3 FROM
 TABLE(tf5(t.c4)) AS z -- correlated reference
) -- to previous FROM clause
```

- Table Functions That Modify SQL Data

Table functions that are specified with the `MODIFIES SQL DATA` option can only be used as the last table reference in a *select-statement*, *common-table-expression*, or `RETURN` statement that is a subselect, a `SELECT INTO`, or a *row-fullselect* in a `SET` statement. Only one table function is allowed in one `FROM` clause, and the table function arguments must be correlated to all other table references in the subselect (SQLSTATE 429BL). The following examples have valid syntax for a table function with the `MODIFIES SQL DATA` property:

```
Example 1: SELECT c1
 FROM TABLE(tfmod('Jones')) AS z
```

```
Example 2: SELECT c1
 FROM t1, t2, TABLE(tfmod(t1.c1, t2.c1)) AS z
```

```
Example 3: SET var =
 (SELECT c1
 FROM TABLE(tfmod('Jones')) AS z
```

```
Example 4: RETURN SELECT c1
 FROM TABLE(tfmod('Jones')) AS z
```

```
Example 5: WITH v1(c1) AS
 (SELECT c1
 FROM TABLE(tfmod(:hostvar1)) AS z)
 SELECT c1
 FROM v1, t1 WHERE v1.c1 = t1.c1
```

## Error tolerant nested-table-expression

Certain errors that occur within a *nested-table-expression* can be tolerated, and instead of returning an error, the query can continue and return a result.

Specifying the `RETURN DATA UNTIL` clause will cause any rows that are returned from the fullselect before the indicated condition is encountered to make up the result set from the fullselect. This means that a partial result set (which could also be an empty result set) from the fullselect is acceptable as the result for the *nested-table-expression*.

The `FEDERATED` keyword restricts the condition to handle only errors that occur at a remote data source.

The condition can be specified as an `SQLSTATE` value, with a *string-constant* length of 5. You can optionally specify an `SQLCODE` value for each specified `SQLSTATE`

value. For portable applications, specify SQLSTATE values as much as possible, because SQLCODE values are generally not portable across platforms and are not part of the SQL standard.

Only certain conditions can be tolerated. Errors that do not allow the rest of the query to be executed cannot be tolerated, and an error is returned for the whole query. The *specific-condition-value* might specify conditions that cannot actually be tolerated by the database manager, even if a specific SQLSTATE or SQLCODE value is specified, and for these cases, an error is returned.

The following SQLSTATE values and SQLCODE values have the potential, when specified, to be tolerated by the database manager:

- SQLSTATE 08001; SQLCODEs -1336, -30080, -30081, -30082
- SQLSTATE 08004
- SQLSTATE 42501
- SQLSTATE 42704; SQLCODE -204
- SQLSTATE 42720
- SQLSTATE 28000

A query or view containing an error tolerant *nested-table-expression* is read-only.

The fullselect of an error tolerant *nested-table-expression* is not optimized using materialized query tables.

## Correlated references in table-references

Correlated references can be used in nested table expressions or as arguments to table functions. The basic rule that applies for both these cases is that the correlated reference must be from a *table-reference* at a higher level in the hierarchy of subqueries. This hierarchy includes the table-references that have already been resolved in the left-to-right processing of the FROM clause. For nested table expressions, the LATERAL keyword must appear before the fullselect. So the following examples are valid syntax:

```
Example 1: SELECT t.c1, z.c5
 FROM t, TABLE(tf3(t.c2)) AS z -- t precedes tf3
 WHERE t.c3 = z.c4; -- in FROM, so t.c2
 -- is known

Example 2: SELECT t.c1, z.c5
 FROM t, TABLE(tf4(2 * t.c2)) AS z -- t precedes tf4
 WHERE t.c3 = z.c4; -- in FROM, so t.c2
 -- is known

Example 3: SELECT d.deptno, d.deptname,
 empinfo.avgsal, empinfo.empcount
 FROM department d,
 LATERAL (SELECT AVG(e.salary) AS avgsal,
 COUNT(*) AS empcount
 FROM employee e -- department precedes
 WHERE e.workdept=d.deptno -- and TABLE is
) AS empinfo; -- specified, so
 -- d.deptno is known
```

But the following examples are not valid:

```
Example 4: SELECT t.c1, z.c5
 FROM TABLE(tf6(t.c2)) AS z, t -- cannot resolve t in t.c2!
 WHERE t.c3 = z.c4; -- compare to Example 1 above.
```

```

Example 5: SELECT a.c1, b.c5
 FROM TABLE(tf7a(b.c2)) AS a, TABLE(tf7b(a.c6)) AS b
 WHERE a.c3 = b.c4; -- cannot resolve b in b.c2!

```

```

Example 6: SELECT d.deptno, d.deptname,
 empinfo.avgsal, empinfo.empcount
 FROM department d,
 (SELECT AVG(e.salary) AS avgsal,
 COUNT(*) AS empcount
 FROM employee e -- department precedes
 WHERE e.workdept=d.deptno -- but TABLE is not
) AS empinfo; -- specified, so
 -- d.deptno is unknown

```

## Data change table references

A *data-change-table-reference* clause specifies an intermediate result table. This table is based on the rows that are directly changed by the searched UPDATE, searched DELETE, or INSERT statement that is included in the clause. A *data-change-table-reference* can be specified as the only *table-reference* in the FROM clause of the outer fullselect that is used in a *select-statement*, a SELECT INTO statement, or a common table expression. A *data-change-table-reference* can be specified as the only table reference in the only fullselect in a SET Variable statement (SQLSTATE 428FL). The target table or view of the data change statement is considered to be a table or view that is referenced in the query; therefore, the authorization ID of the query must have SELECT privilege on that target table or view. A *data-change-table-reference* clause cannot be specified in a view definition, materialized query table definition, or FOR statement (SQLSTATE 428FL).

The target of the UPDATE, DELETE, or INSERT statement cannot be a temporary view defined in a common table expression (SQLSTATE 42807).

### FINAL TABLE

Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement as they appear at the completion of the data change statement. If there are AFTER triggers or referential constraints that result in further operations on the table that is the target of the SQL data change statement, an error is returned (SQLSTATE 560C6). If the target of the SQL data change statement is a view that is defined with an INSTEAD OF trigger for the type of data change, an error is returned (SQLSTATE 428G3).

### NEW TABLE

Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement prior to the application of referential constraints and AFTER triggers. Data in the target table at the completion of the statement might not match the data in the intermediate result table because of additional processing for referential constraints and AFTER triggers.

### OLD TABLE

Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement as they existed prior to the application of the data change statement.

*(searched-update-statement)*

Specifies a searched UPDATE statement. A WHERE clause or a SET clause in the UPDATE statement cannot contain correlated references to columns outside of the UPDATE statement.

(*searched-delete-statement*)

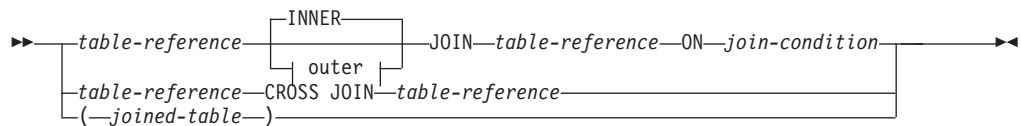
Specifies a searched DELETE statement. A WHERE clause in the DELETE statement cannot contain correlated references to columns outside of the DELETE statement.

(*insert-statement*)

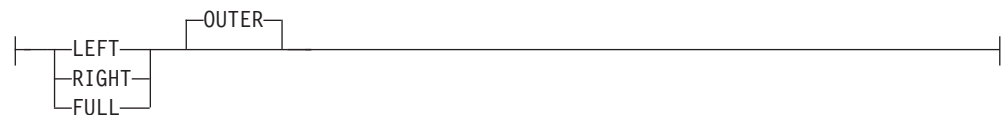
Specifies an INSERT statement. A fullselect in the INSERT statement cannot contain correlated references to columns outside of the fullselect of the INSERT statement.

The content of the intermediate result table for a *data-change-table-reference* is determined when the cursor opens. The intermediate result table contains all manipulated rows, including all the columns in the specified target table or view. All the columns of the target table or view for an SQL data change statement are accessible using the column names from the target table or view. If an INCLUDE clause was specified within a data change statement, the intermediate result table will contain these additional columns.

## joined-table



**outer:**



A *joined table* specifies an intermediate result table that is the result of either an inner join or an outer join. The table is derived by applying one of the join operators: CROSS, INNER, LEFT OUTER, RIGHT OUTER, or FULL OUTER to its operands.

Cross joins represent the cross product of the tables, where each row of the left table is combined with every row of the right table. Inner joins can be thought of as the cross product of the tables, keeping only the rows where the join condition is true. The result table might be missing rows from either or both of the joined tables. Outer joins include the inner join and preserve these missing rows. There are three types of outer joins:

- **Left outer join** includes rows from the left table that were missing from the inner join.
- **Right outer join** includes rows from the right table that were missing from the inner join.
- **Full outer join** includes rows from both the left and right tables that were missing from the inner join.

If a join-operator is not specified, INNER is implicit. The order in which multiple joins are performed can affect the result. Joins can be nested within other joins. The order of processing for joins is generally from left to right, but based on the



position of the required join-condition. Parentheses are recommended to make the order of nested joins more readable. For example:

```
TB1 LEFT JOIN TB2 ON TB1.C1=TB2.C1
 RIGHT JOIN TB3 LEFT JOIN TB4 ON TB3.C1=TB4.C1
 ON TB1.C1=TB3.C1
```

is the same as:

```
(TB1 LEFT JOIN TB2 ON TB1.C1=TB2.C1)
 RIGHT JOIN (TB3 LEFT JOIN TB4 ON TB3.C1=TB4.C1)
 ON TB1.C1=TB3.C1
```

A joined table can be used in any context in which any form of the SELECT statement is used. A view or a cursor is read-only if its SELECT statement includes a joined table.

A *join-condition* is a *search-condition*, except that:

- It cannot contain any subqueries, scalar or otherwise
- It cannot include any dereference operations or the Deref function, where the reference value is other than the object identifier column
- It cannot include an SQL function
- Any column referenced in an expression of the *join-condition* must be a column of one of the operand tables of the associated join (in the scope of the same joined-table clause)
- Any function referenced in an expression of the *join-condition* of a full outer join must be deterministic and have no external action
- It cannot include an XMLQUERY or XMLEXISTS expression

An error occurs if the join condition does not comply with these rules (SQLSTATE 42972).

Column references are resolved using the rules for resolution of column name qualifiers. The same rules that apply to predicates apply to join conditions.

## Join operations

A *join-condition* specifies pairings of T1 and T2, where T1 and T2 are the left and right operand tables of the JOIN operator of the *join-condition*. For all possible combinations of rows of T1 and T2, a row of T1 is paired with a row of T2 if the *join-condition* is true. When a row of T1 is joined with a row of T2, a row in the result consists of the values of that row of T1 concatenated with the values of that row of T2. The execution might involve the generation of a null row. The null row of a table consists of a null value for each column of the table, regardless of whether the columns allow null values.

The following summarizes the result of the join operations:

- The result of T1 CROSS JOIN T2 consists of all possible pairings of their rows.
- The result of T1 INNER JOIN T2 consists of their paired rows where the join-condition is true.
- The result of T1 LEFT OUTER JOIN T2 consists of their paired rows where the join-condition is true and, for each unpaired row of T1, the concatenation of that row with the null row of T2. All columns derived from T2 allow null values.
- The result of T1 RIGHT OUTER JOIN T2 consists of their paired rows where the join-condition is true and, for each unpaired row of T2, the concatenation of that row with the null row of T1. All columns derived from T1 allow null values.



- The result of T1 FULL OUTER JOIN T2 consists of their paired rows and, for each unpaired row of T2, the concatenation of that row with the null row of T1 and, for each unpaired row of T1, the concatenation of that row with the null row of T2. All columns derived from T1 and T2 allow null values.

## where-clause

►—WHERE—*search-condition*—◄

The WHERE clause specifies an intermediate result table that consists of those rows of R for which the *search-condition* is true. R is the result of the FROM clause of the subselect.

The *search-condition* must conform to the following rules:

- Each *column-name* must unambiguously identify a column of R or be a correlated reference. A *column-name* is a correlated reference if it identifies a column of a *table-reference* in an outer subselect.
- An aggregate function must not be specified unless the WHERE clause is specified in a subquery of a HAVING clause and the argument of the function is a correlated reference to a group.

Any subquery in the *search-condition* is effectively executed for each row of R, and the results are used in the application of the *search-condition* to the given row of R. A subquery is actually executed for each row of R only if it includes a correlated reference. In fact, a subquery with no correlated references may be executed just once, whereas a subquery with a correlated reference may have to be executed once for each row.

## group-by-clause

►—GROUP BY—  
┌───┐  
grouping-expression  
grouping-sets  
super-groups  
└───┘
◄

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

In its simplest form, a GROUP BY clause contains a *grouping expression*. A grouping expression is an *expression* used in defining the grouping of R. Each expression or *column name* included in grouping-expression must unambiguously identify a column of R (SQLSTATE 42702 or 42703). A grouping expression cannot include a scalar fullselect or an XMLQUERY or XMLEXISTS expression (SQLSTATE 42822), or any expression or function that is not deterministic or has an external action (SQLSTATE 42845).

**Note:** The following expressions, which do not contain an explicit column reference, can be used in a *grouping-expression* to identify a column of R:

- ROW CHANGE TIMESTAMP FOR *table-designator*
- ROW CHANGE TOKEN FOR *table-designator*
- RID\_BIT or RID scalar function

More complex forms of the GROUP BY clause include *grouping-sets* and *super-groups*. For a description of these forms, see “grouping-sets” and “super-groups” on page 437, respectively.

The result of GROUP BY is a set of groups of rows. Each row in this result represents the set of rows for which the *grouping-expression* is equal. For grouping, all null values from a *grouping-expression* are considered equal.

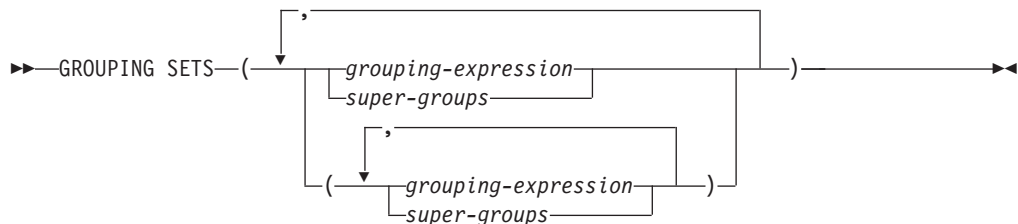
If a *grouping-expression* contains decimal floating-point columns, and multiple representations of the same number exist in these columns, the number that is returned can be any of the representations of the number.

A *grouping-expression* can be used in a search condition in a HAVING clause, in an expression in a SELECT clause or in a *sort-key-expression* of an ORDER BY clause (see “order-by-clause” on page 442 for details). In each case, the reference specifies only one value for each group. For example, if the *grouping-expression* is *col1+col2*, then an allowed expression in the select list would be *col1+col2+3*. Associativity rules for expressions would disallow the similar expression, *3+col1+col2*, unless parentheses are used to ensure that the corresponding expression is evaluated in the same order. Thus, *3+(col1+col2)* would also be allowed in the select list. If the concatenation operator is used, the *grouping-expression* must be used exactly as the expression was specified in the select list.

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

As noted, there are some cases where the GROUP BY clause cannot refer directly to a column that is specified in the SELECT clause as an expression (scalar-fullselect, not deterministic or external action functions). To group using such an expression, use a nested table expression or a common table expression to first provide a result table with the expression as a column of the result. For an example using nested table expressions, see Example A9.

## grouping-sets



A *grouping-sets* specification allows multiple grouping clauses to be specified in a single statement. This can be thought of as the union of two or more groups of rows into a single result set. It is logically equivalent to the union of multiple subselects with the group by clause in each subselect corresponding to one grouping set. A grouping set can be a single element or can be a list of elements

delimited by parentheses, where an element is either a grouping-expression or a super-group. Using *grouping-sets* allows the groups to be computed with a single pass over the base table.

The *grouping-sets* specification allows either a simple *grouping-expression* to be used, or the more complex forms of *super-groups*. For a description of *super-groups*, see “super-groups.”

Note that grouping sets are the fundamental building blocks for GROUP BY operations. A simple GROUP BY with a single column can be considered a grouping set with one element. For example:

**GROUP BY a**

is the same as

**GROUP BY GROUPING SETS((a))**

and

**GROUP BY a,b,c**

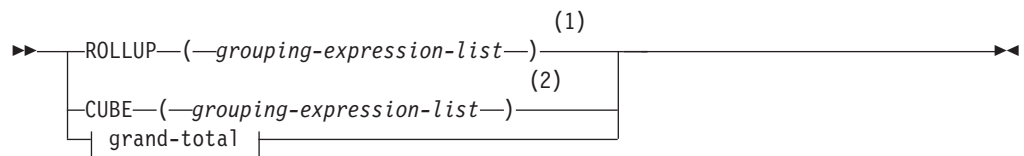
is the same as

**GROUP BY GROUPING SETS((a,b,c))**

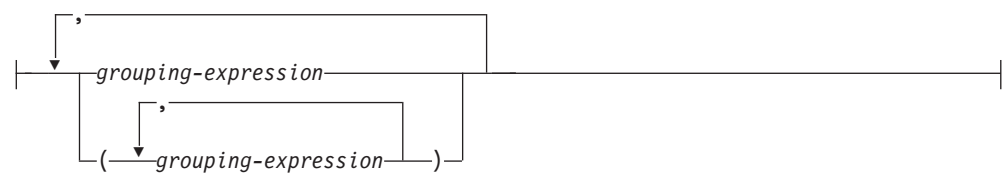
Non-aggregation columns from the select list of the subselect that are excluded from a grouping set will return a null for such columns for each row generated for that grouping set. This reflects the fact that aggregation was done without considering the values for those columns.

Example C2 through Example C7 illustrate the use of grouping sets.

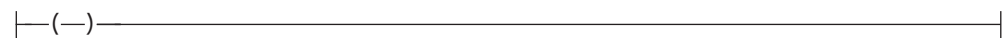
### super-groups



### grouping-expression-list:



### grand-total:



### Notes:

- 1 Alternate specification when used alone in group-by-clause is: grouping-expression-list WITH ROLLUP.

- 2 Alternate specification when used alone in group-by-clause is:  
grouping-expression-list WITH CUBE.

**ROLLUP** ( *grouping-expression-list* )

A *ROLLUP grouping* is an extension to the GROUP BY clause that produces a result set containing *sub-total* rows in addition to the “regular” grouped rows. *Sub-total* rows are “super-aggregate” rows that contain further aggregates whose values are derived by applying the same aggregate functions that were used to obtain the grouped rows. These rows are called sub-total rows, because that is their most common use; however, any aggregate function can be used for the aggregation. For instance, MAX and AVG are used in Example C8. The GROUPING aggregate function can be used to indicate if a row was generated by the super-group.

A ROLLUP grouping is a series of *grouping-sets*. The general specification of a ROLLUP with *n* elements

**GROUP BY ROLLUP**(*C*<sub>1</sub>, *C*<sub>2</sub>, . . . , *C*<sub>*n*-1</sub>, *C*<sub>*n*</sub>)

is equivalent to

**GROUP BY GROUPING SETS**((*C*<sub>1</sub>, *C*<sub>2</sub>, . . . , *C*<sub>*n*-1</sub>, *C*<sub>*n*</sub>)  
(*C*<sub>1</sub>, *C*<sub>2</sub>, . . . , *C*<sub>*n*-1</sub>)  
.  
(*C*<sub>1</sub>, *C*<sub>2</sub>)  
(*C*<sub>1</sub>)  
( ) )

Note that the *n* elements of the ROLLUP translate to *n*+1 grouping sets. Note also that the order in which the *grouping-expressions* is specified is significant for ROLLUP. For example:

**GROUP BY ROLLUP**(*a*, *b*)

is equivalent to

**GROUP BY GROUPING SETS**((*a*, *b*)  
(*a*)  
( ) )

while

**GROUP BY ROLLUP**(*b*, *a*)

is the same as

**GROUP BY GROUPING SETS**((*b*, *a*)  
(*b*)  
( ) )

The ORDER BY clause is the only way to guarantee the order of the rows in the result set. Example C3 illustrates the use of ROLLUP.

**CUBE** ( *grouping-expression-list* )

A *CUBE grouping* is an extension to the GROUP BY clause that produces a result set that contains all the rows of a ROLLUP aggregation and, in addition, contains “cross-tabulation” rows. *Cross-tabulation* rows are additional “super-aggregate” rows that are not part of an aggregation with sub-totals. The GROUPING aggregate function can be used to indicate if a row was generated by the super-group.

Like a ROLLUP, a CUBE grouping can also be thought of as a series of *grouping-sets*. In the case of a CUBE, all permutations of the cubed

*grouping-expression-list* are computed along with the grand total. Therefore, the  $n$  elements of a CUBE translate to  $2^{*n}$  (2 to the power  $n$ ) *grouping-sets*. For example, a specification of:

```
GROUP BY CUBE(a,b,c)
```

is equivalent to:

```
GROUP BY GROUPING SETS((a,b,c)
 (a,b)
 (a,c)
 (b,c)
 (a)
 (b)
 (c)
 ())
```

Note that the three elements of the CUBE translate into eight grouping sets.

The order of specification of elements does not matter for CUBE. 'CUBE (DayOfYear, Sales\_Person)' and 'CUBE (Sales\_Person, DayOfYear)' yield the same result sets. The use of the word 'same' applies to content of the result set, not to its order. The ORDER BY clause is the only way to guarantee the order of the rows in the result set. Example C4 illustrates the use of CUBE.

#### *grouping-expression-list*

A *grouping-expression-list* is used within a CUBE or ROLLUP clause to define the number of elements in the CUBE or ROLLUP operation. This is controlled by using parentheses to delimit elements with multiple *grouping-expressions*.

The rules for a *grouping-expression* are described in "group-by-clause" on page 435. For example, suppose that a query is to return the total expenses for the ROLLUP of City within a Province but not within a County. However, the clause:

```
GROUP BY ROLLUP(Province, County, City)
```

results in unwanted subtotal rows for the County. In the clause:

```
GROUP BY ROLLUP(Province, (County, City))
```

the composite (County, City) forms one element in the ROLLUP and, therefore, a query that uses this clause will yield the desired result. In other words, the two-element ROLLUP:

```
GROUP BY ROLLUP(Province, (County, City))
```

generates:

```
GROUP BY GROUPING SETS((Province, County, City)
 (Province)
 ())
```

and the three-element ROLLUP generates:

```
GROUP BY GROUPING SETS((Province, County, City)
 (Province, County)
 (Province)
 ())
```

Example C2 also utilizes composite column values.

#### **grand-total**

Both CUBE and ROLLUP return a row which is the overall (grand total) aggregation. This may be separately specified with empty parentheses within

the GROUPING SET clause. It may also be specified directly in the GROUP BY clause, although there is no effect on the result of the query. Example C4 uses the grand-total syntax.

## Combining grouping sets

This can be used to combine any of the types of GROUP BY clauses. When simple *grouping-expression* fields are combined with other groups, they are "appended" to the beginning of the resulting *grouping sets*. When ROLLUP or CUBE expressions are combined, they operate like "multipliers" on the remaining expression, forming additional grouping set entries according to the definition of either ROLLUP or CUBE.

For instance, combining *grouping-expression* elements acts as follows:

```
GROUP BY a, ROLLUP(b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
 (a,b)
 (a))
```

Or similarly,

```
GROUP BY a, b, ROLLUP(c,d)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c,d)
 (a,b,c)
 (a,b))
```

Combining of *ROLLUP* elements acts as follows:

```
GROUP BY ROLLUP(a), ROLLUP(b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
 (a,b)
 (a)
 (b,c)
 (b)
 ())
```

Similarly,

```
GROUP BY ROLLUP(a), CUBE(b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
 (a,b)
 (a,c)
 (a)
 (b,c)
 (b)
 (c)
 ())
```

Combining of *CUBE* and *ROLLUP* elements acts as follows:

```
GROUP BY CUBE(a,b), ROLLUP(c,d)
```

is equivalent to

```

GROUP BY GROUPING SETS((a,b,c,d)
 (a,b,c)
 (a,b)
 (a,c,d)
 (a,c)
 (a)
 (b,c,d)
 (b,c)
 (b)
 (c,d)
 (c)
 ())

```

Like a simple *grouping-expression*, combining grouping sets also eliminates duplicates within each grouping set. For instance,

```

GROUP BY a, ROLLUP(a,b)

```

is equivalent to

```

GROUP BY GROUPING SETS((a,b)
 (a))

```

A more complete example of combining grouping sets is to construct a result set that eliminates certain rows that would be returned for a full CUBE aggregation.

For example, consider the following GROUP BY clause:

```

GROUP BY Region,
 ROLLUP(Sales_Person, WEEK(Sales_Date)),
 CUBE(YEAR(Sales_Date), MONTH (Sales_Date))

```

The column listed immediately to the right of GROUP BY is simply grouped, those within the parenthesis following ROLLUP are rolled up, and those within the parenthesis following CUBE are cubed. Thus, the above clause results in a cube of MONTH within YEAR which is then rolled up within WEEK within Sales\_Person within the Region aggregation. It does not result in any grand total row or any cross-tabulation rows on Region, Sales\_Person or WEEK(Sales\_Date) so produces fewer rows than the clause:

```

GROUP BY ROLLUP (Region, Sales_Person, WEEK(Sales_Date),
 YEAR(Sales_Date), MONTH(Sales_Date))

```

## having-clause

►—HAVING—*search-condition*—◄

The HAVING clause specifies an intermediate result table that consists of those groups of R for which the *search-condition* is true. R is the result of the previous clause of the subselect. If this clause is not GROUP BY, R is considered to be a single group with no grouping columns.

Each *column-name* in the search condition must do one of the following:

- Unambiguously identify a grouping column of R.
- Be specified within an aggregate function.
- Be a correlated reference. A *column-name* is a correlated reference if it identifies a column of a *table-reference* in an outer subselect.



A group of R to which the search condition is applied supplies the argument for each aggregate function in the search condition, except for any function whose argument is a correlated reference.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a group of R, and the results used in applying the search condition. In actuality, the subquery is executed for each group only if it contains a correlated reference. For an illustration of the difference, see Example A6 and Example A7.

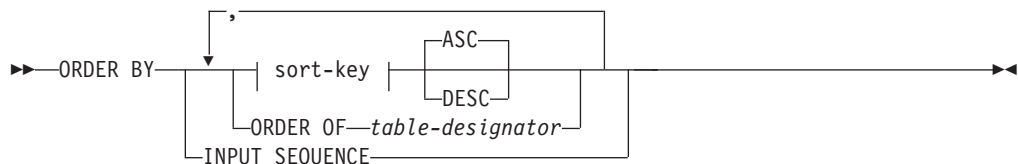
A correlated reference to a group of R must either identify a grouping column or be contained within an aggregate function.

When HAVING is used without GROUP BY, the select list can only include column names when they are arguments to an aggregate function, correlated column references, global variables, host variables, literals, special registers, SQL variables, or SQL parameters.

**Note:** The following expressions can only be specified in a HAVING clause if they are contained within an aggregate function (SQLSTATE 42803):

- ROW CHANGE TIMESTAMP FOR *table-designator*
- ROW CHANGE TOKEN FOR *table-designator*
- RID\_BIT or RID scalar function

### order-by-clause



### sort-key:



The ORDER BY clause specifies an ordering of the rows of the result table. If a single sort specification (one *sort-key* with associated direction) is identified, the rows are ordered by the values of that sort specification. If more than one sort specification is identified, the rows are ordered by the values of the first identified sort specification, then by the values of the second identified sort specification, and so on. Each *sort-key* cannot have a data type of CLOB, DBCLOB, BLOB, distinct type or any of these types, or structured type (SQLSTATE 42907).

A named column in the select list may be identified by a *sort-key* that is a *simple-integer* or a *simple-column-name*. An unnamed column in the select list must be identified by a *simple-integer* or, in some cases, by a *sort-key-expression* that matches the expression in the select list (see details of *sort-key-expression*). A column is unnamed if the AS clause is not specified and it is derived from a constant, an expression with operators, or a function.

Ordering is performed in accordance with comparison rules. If an ORDER BY clause contains decimal floating-point columns, and multiple representations of the same number exist in these columns, the ordering of the multiple representations of the same number is unspecified. The null value is higher than all other values. If the ORDER BY clause does not completely order the rows, rows with duplicate values of all identified columns are displayed in an arbitrary order.

*simple-column-name*

Usually identifies a column of the result table. In this case, *simple-column-name* must be the column name of a named column in the select list.

The *simple-column-name* can also identify a column name of a table, view, or nested table identified in the FROM clause if the query is a subselect. This includes columns defined as implicitly hidden. An error occurs if the subselect:

- Specifies DISTINCT in the select-clause (SQLSTATE 42822)
- Produces a grouped result and the *simple-column-name* is not a *grouping-expression* (SQLSTATE 42803).

Determining which column is used for ordering the result is described under “Column names in sort keys” below.

*simple-integer*

Must be greater than 0 and not greater than the number of columns in the result table (SQLSTATE 42805). The integer *n* identifies the *n*th column of the result table.

*sort-key-expression*

An expression that is not simply a column name or an unsigned integer constant. The query to which ordering is applied must be a *subselect* to use this form of sort-key. The *sort-key-expression* cannot include a correlated scalar fullselect (SQLSTATE 42703), an XMLQUERY or XMLEXISTS expression (SQLSTATE 42822), or a function with an external action (SQLSTATE 42845).

Any *column-name* within a *sort-key-expression* must conform to the rules described under “Column names in sort keys” below.

There are a number of special cases that further restrict the expressions that can be specified.

- DISTINCT is specified in the SELECT clause of the subselect (SQLSTATE 42822).

The sort-key-expression must match exactly with an expression in the select list of the subselect (scalar-fullselects are never matched).

- The subselect is grouped (SQLSTATE 42803).

The sort-key-expression can:

- be an expression in the select list of the subselect,
- include a *grouping-expression* from the GROUP BY clause of the subselect
- include an aggregate function, constant or host variable.

**ASC**

Uses the values of the column in ascending order. This is the default.

**DESC**

Uses the values of the column in descending order.

**ORDER OF** *table-designator*

Specifies that the same ordering used in *table-designator* should be applied to the result table of the subselect. There must be a table reference matching *table-designator* in the FROM clause of the subselect that specifies this clause (SQLSTATE 42703). The subselect (or fullselect) corresponding to the specified

*table-designator* must include an ORDER BY clause that is dependant on the data (SQLSTATE 428FI). The ordering that is applied is the same as if the columns of the ORDER BY clause in the nested subselect (or fullselect) were included in the outer subselect (or fullselect), and these columns were specified in place of the ORDER OF clause.

Note that this form is not allowed in a fullselect (other than the degenerative form of a fullselect). For example, the following is not valid:

```
(SELECT C1 FROM T1
 ORDER BY C1)
UNION
SELECT C1 FROM T2
 ORDER BY ORDER OF T1
```

The following example *is* valid:

```
SELECT C1 FROM
 (SELECT C1 FROM T1
 UNION
 SELECT C1 FROM T2
 ORDER BY C1) AS UTABLE
ORDER BY ORDER OF UTABLE
```

### INPUT SEQUENCE

Specifies that, for an INSERT statement, the result table will reflect the input order of ordered data rows. INPUT SEQUENCE ordering can only be specified if an INSERT statement is used in a FROM clause (SQLSTATE 428G4). See “table-reference” on page 424. If INPUT SEQUENCE is specified and the input data is not ordered, the INPUT SEQUENCE clause is ignored.

### Notes

- **Column names in sort keys:**

- The column name is qualified.

The query must be a *subselect* (SQLSTATE 42877). The column name must unambiguously identify a column of some table, view or nested table in the FROM clause of the subselect (SQLSTATE 42702). The value of the column is used to compute the value of the sort specification.

- The column name is unqualified.

- The query is a subselect.

If the column name is identical to the name of more than one column of the result table, the column name must unambiguously identify a column of some table, view or nested table in the FROM clause of the ordering subselect (SQLSTATE 42702). If the column name is identical to one column, that column is used to compute the value of the sort specification. If the column name is not identical to a column of the result table, then it must unambiguously identify a column of some table, view or nested table in the FROM clause of the fullselect in the select-statement (SQLSTATE 42702).

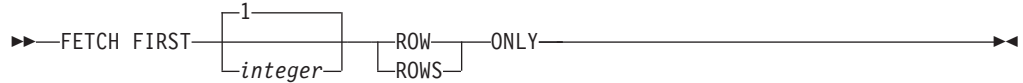
- The query is not a subselect (it includes set operations such as union, except or intersect).

The column name must not be identical to the name of more than one column of the result table (SQLSTATE 42702). The column name must be identical to exactly one column of the result table (SQLSTATE 42707), and this column is used to compute the value of the sort specification.

- **Limits:** The use of a *sort-key-expression* or a *simple-column-name* where the column is not in the select list may result in the addition of the column or expression to the temporary table used for sorting. This may result in reaching the limit of the

number of columns in a table or the limit on the size of a row in a table. Exceeding these limits will result in an error if a temporary table is required to perform the sorting operation.

### fetch-first-clause



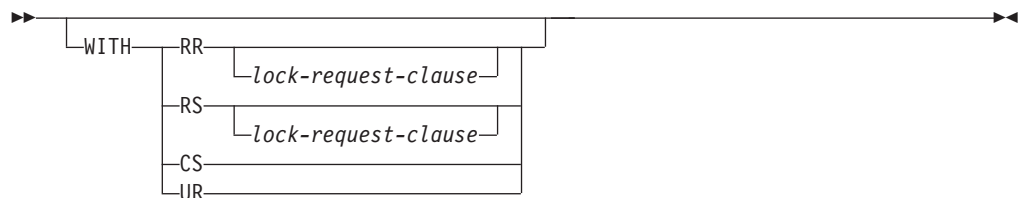
The *fetch-first-clause* sets a maximum number of rows that can be retrieved. It lets the database manager know that the application does not want to retrieve more than *integer* rows, regardless of how many rows there might be in the result table when this clause is not specified. An attempt to fetch beyond *integer* rows is handled the same way as normal end of data (SQLSTATE 02000). The value of *integer* must be a positive integer (not zero).

Use of the *fetch-first-clause* influences query optimization of the subselect or fullselect, based on the fact that at most *integer* rows will be retrieved. If both the *fetch-first-clause* is specified in the outermost fullselect and the *optimize-for-clause* is specified for the select statement, the database manager will use the *integer* from the *optimize-for-clause* to influence query optimization of the outermost fullselect.

Limiting the result table to the first *integer* rows can improve performance. The database manager will cease processing the query once it has determined the first *integer* rows. If both the *fetch-first-clause* and the *optimize-for-clause* are specified, the lower of the *integer* values from these clauses is used to influence the communications buffer size.

If the fullselect contains an SQL data change statement in the FROM clause, all the rows are modified regardless of the limit on the number of rows to fetch.

### isolation-clause



The optional *isolation-clause* specifies the isolation level at which the subselect or fullselect is executed, and whether a specific type of lock is to be acquired.

- RR - Repeatable-Read
- RS - Read Stability
- CS - Cursor Stability
- UR - Uncommitted Read

## lock-request-clause



The *lock-request-clause* applies only to queries and to positioning read operations within an insert, update, or delete operation. The insert, update, and delete operations themselves will execute using locking determined by the database manager.

The optional *lock-request-clause* specifies the type of lock that the database manager is to acquire and hold:

### SHARE

Concurrent processes can acquire SHARE or UPDATE locks on the data.

### UPDATE

Concurrent processes can acquire SHARE locks on the data, but no concurrent process can acquire an UPDATE or EXCLUSIVE lock.

### EXCLUSIVE

Concurrent processes cannot acquire a lock on the data.

### *isolation-clause* restrictions:

- The *isolation-clause* is not supported for a CREATE TABLE, CREATE VIEW, or ALTER TABLE statement (SQLSTATE 42601).
- The *isolation-clause* cannot be specified for a subselect or fullselect that will cause trigger invocation, referential integrity scans, or MQT maintenance (SQLSTATE 42601).
- A subselect or fullselect cannot include a *lock-request-clause* if that subselect or fullselect references any SQL functions that are not declared with the option INHERIT ISOLATION LEVEL WITH LOCK REQUEST (SQLSTATE 42601).
- A subselect or fullselect that contains a *lock-request-clause* are not be eligible for MQT routing.
- If an *isolation-clause* is specified for a *subselect* or *fullselect* within the body of an SQL function, SQL method, or trigger, the clause is ignored and a warning is returned.
- If an *isolation-clause* is specified for a *subselect* or *fullselect* that is used by a scrollable cursor, the clause is ignored and a warning is returned.
- Neither *isolation-clause* nor *lock-request-clause* can be specified in the context where they will cause conflict isolation or lock intent on a common table expression (SQLSTATE 42601). This restriction does not apply to aliases or base tables. The following examples create an isolation conflict on *a* and returns an error:

– View:

```
create view a as (...);
(select * from a with RR USE AND KEEP SHARE LOCKS)
UNION ALL
(select * from a with UR);
```

– Common table expression:

```
WITH a as (...)
(select * from a with RR USE AND KEEP SHARE LOCKS)
UNION ALL
(select * from a with UR);
```

- An *isolation\_clause* cannot be specified in an XML context (SQLSTATE 2200M).

## Examples of subselects

*Example A1:* Select all columns and rows from the EMPLOYEE table.

```
SELECT * FROM EMPLOYEE
```

*Example A2:* Join the EMP\_ACT and EMPLOYEE tables, select all the columns from the EMP\_ACT table and add the employee's surname (LASTNAME) from the EMPLOYEE table to each row of the result.

```
SELECT EMP_ACT.*, LASTNAME
FROM EMP_ACT, EMPLOYEE
WHERE EMP_ACT.EMPNO = EMPLOYEE.EMPNO
```

*Example A3:* Join the EMPLOYEE and DEPARTMENT tables, select the employee number (EMPNO), employee surname (LASTNAME), department number (WORKDEPT in the EMPLOYEE table and DEPTNO in the DEPARTMENT table) and department name (DEPTNAME) of all employees who were born (BIRTHDATE) earlier than 1930.

```
SELECT EMPNO, LASTNAME, WORKDEPT, DEPTNAME
FROM EMPLOYEE, DEPARTMENT
WHERE WORKDEPT = DEPTNO
AND YEAR(BIRTHDATE) < 1930
```

*Example A4:* Select the job (JOB) and the minimum and maximum salaries (SALARY) for each group of rows with the same job code in the EMPLOYEE table, but only for groups with more than one row and with a maximum salary greater than or equal to 27000.

```
SELECT JOB, MIN(SALARY), MAX(SALARY)
FROM EMPLOYEE
GROUP BY JOB
HAVING COUNT(*) > 1
AND MAX(SALARY) >= 27000
```

*Example A5:* Select all the rows of EMP\_ACT table for employees (EMPNO) in department (WORKDEPT) 'E11'. (Employee department numbers are shown in the EMPLOYEE table.)

```
SELECT *
FROM EMP_ACT
WHERE EMPNO IN
 (SELECT EMPNO
 FROM EMPLOYEE
 WHERE WORKDEPT = 'E11')
```

*Example A6:* From the EMPLOYEE table, select the department number (WORKDEPT) and maximum departmental salary (SALARY) for all departments whose maximum salary is less than the average salary for all employees.

```
SELECT WORKDEPT, MAX(SALARY)
FROM EMPLOYEE
GROUP BY WORKDEPT
HAVING MAX(SALARY) < (SELECT AVG(SALARY)
 FROM EMPLOYEE)
```

The subquery in the HAVING clause would only be executed once in this example.

*Example A7:* Using the EMPLOYEE table, select the department number (WORKDEPT) and maximum departmental salary (SALARY) for all departments whose maximum salary is less than the average salary in all other departments.

```

SELECT WORKDEPT, MAX(SALARY)
FROM EMPLOYEE EMP_COR
GROUP BY WORKDEPT
HAVING MAX(SALARY) < (SELECT AVG(SALARY)
FROM EMPLOYEE
WHERE NOT WORKDEPT = EMP_COR.WORKDEPT)

```

In contrast to Example A6, the subquery in the HAVING clause would need to be executed for each group.

*Example A8:* Determine the employee number and salary of sales representatives along with the average salary and head count of their departments.

This query must first create a nested table expression (DINFO) in order to get the AVGSALARY and EMPCOUNT columns, as well as the DEPTNO column that is used in the WHERE clause.

```

SELECT THIS_EMP.EMPNO, THIS_EMP.SALARY, DINFO.AVGSALARY, DINFO.EMPCOUNT
FROM EMPLOYEE THIS_EMP,
 (SELECT OTHERS.WORKDEPT AS DEPTNO,
 AVG(OTHERS.SALARY) AS AVGSALARY,
 COUNT(*) AS EMPCOUNT
 FROM EMPLOYEE OTHERS
 GROUP BY OTHERS.WORKDEPT
) AS DINFO
WHERE THIS_EMP.JOB = 'SALESREP'
AND THIS_EMP.WORKDEPT = DINFO.DEPTNO

```

Using a nested table expression for this case saves the overhead of creating the DINFO view as a regular view. During statement preparation, accessing the catalog for the view is avoided and, because of the context of the rest of the query, only the rows for the department of the sales representatives need to be considered by the view.

*Example A9:* Display the average education level and salary for 5 random groups of employees.

This query requires the use of a nested table expression to set a random value for each employee so that it can subsequently be used in the GROUP BY clause.

```

SELECT RANDID , AVG(EDLEVEL), AVG(SALARY)
FROM (SELECT EDLEVEL, SALARY, INTEGER(RAND()*5) AS RANDID
 FROM EMPLOYEE
) AS EMPRAND
GROUP BY RANDID

```

*Example A10:* Query the EMP\_ACT table and return those project numbers that have an employee whose salary is in the top 10 of all employees.

```

SELECT EMP_ACT.EMPNO, PROJNO
FROM EMP_ACT
WHERE EMP_ACT.EMPNO IN
 (SELECT EMPLOYEE.EMPNO
 FROM EMPLOYEE
 ORDER BY SALARY DESC
 FETCH FIRST 10 ROWS ONLY)

```

*Example A11:* Assuming that PHONES and IDS are two SQL variables with array values of the same cardinality, turn these arrays into a table with three columns (one for each array and one for the position), and one row per array element.

```

SELECT T.PHONE, T.ID, T.INDEX FROM UNNEST(PHONES, IDS)
WITH ORDINALITY AS T(PHONE, ID, INDEX)
ORDER BY T.INDEX

```



## Examples of joins

*Example B1:* This example illustrates the results of the various joins using tables J1 and J2. These tables contain rows as shown.

```
SELECT * FROM J1
```

| W | X  |
|---|----|
| A | 11 |
| B | 12 |
| C | 13 |

```
SELECT * FROM J2
```

| Y | Z  |
|---|----|
| A | 21 |
| C | 22 |
| D | 23 |

The following query does an inner join of J1 and J2 matching the first column of both tables.

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y
```

| W | X  | Y | Z  |
|---|----|---|----|
| A | 11 | A | 21 |
| C | 13 | C | 22 |

In this inner join example the row with column W='C' from J1 and the row with column Y='D' from J2 are not included in the result because they do not have a match in the other table. Note that the following alternative form of an inner join query produces the same result.

```
SELECT * FROM J1, J2 WHERE W=Y
```

The following left outer join will get back the missing row from J1 with nulls for the columns of J2. Every row from J1 is included.

```
SELECT * FROM J1 LEFT OUTER JOIN J2 ON W=Y
```

| W | X  | Y | Z  |
|---|----|---|----|
| A | 11 | A | 21 |
| B | 12 | - | -  |
| C | 13 | C | 22 |

The following right outer join will get back the missing row from J2 with nulls for the columns of J1. Every row from J2 is included.

```
SELECT * FROM J1 RIGHT OUTER JOIN J2 ON W=Y
```

| W | X  | Y | Z  |
|---|----|---|----|
| A | 11 | A | 21 |
| C | 13 | C | 22 |
| - | -  | D | 23 |

The following full outer join will get back the missing rows from both J1 and J2 with nulls where appropriate. Every row from both J1 and J2 is included.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
```

| W | X | Y | Z |
|---|---|---|---|
|---|---|---|---|

|   |    |   |    |
|---|----|---|----|
| A | 11 | A | 21 |
| C | 13 | C | 22 |
| - | -  | D | 23 |
| B | 12 | - | -  |

*Example B2:* Using the tables J1 and J2 from the previous example, examine what happens when an additional predicate is added to the search condition.

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y AND X=13
```

| W     | X  | Y | Z  |
|-------|----|---|----|
| ----- |    |   |    |
| C     | 13 | C | 22 |

The additional condition caused the inner join to select only 1 row compared to the inner join in Example B1.

Notice what the impact of this is on the full outer join.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y AND X=13
```

| W     | X  | Y | Z  |
|-------|----|---|----|
| ----- |    |   |    |
| -     | -  | A | 21 |
| C     | 13 | C | 22 |
| -     | -  | D | 23 |
| A     | 11 | - | -  |
| B     | 12 | - | -  |

The result now has 5 rows (compared to 4 without the additional predicate) since there was only 1 row in the inner join and all rows of both tables must be returned.

The following query illustrates that placing the same additional predicate in WHERE clause has completely different results.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
WHERE X=13
```

| W     | X  | Y | Z  |
|-------|----|---|----|
| ----- |    |   |    |
| C     | 13 | C | 22 |

The WHERE clause is applied after the intermediate result of the full outer join. This intermediate result would be the same as the result of the full outer join query in Example B1. The WHERE clause is applied to this intermediate result and eliminates all but the row that has X=13. Choosing the location of a predicate when performing outer joins can have significant impact on the results. Consider what happens if the predicate was X=12 instead of X=13. The following inner join returns no rows.

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y AND X=12
```

Hence, the full outer join would return 6 rows, 3 from J1 with nulls for the columns of J2 and 3 from J2 with nulls for the columns of J1.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y AND X=12
```

| W     | X | Y | Z  |
|-------|---|---|----|
| ----- |   |   |    |
| -     | - | A | 21 |
| -     | - | C | 22 |

```

- - D 23
A 11 - -
B 12 - -
C 13 - -

```

If the additional predicate is in the WHERE clause instead, 1 row is returned.

```

SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
WHERE X=12

```

```

W X Y Z
--- -----
B 12 - -

```

*Example B3:* List every department with the employee number and last name of the manager, including departments without a manager.

```

SELECT DEPTNO, DEPTNAME, EMPNO, LASTNAME
FROM DEPARTMENT LEFT OUTER JOIN EMPLOYEE
ON MGRNO = EMPNO

```

*Example B4:* List every employee number and last name with the employee number and last name of their manager, including employees without a manager.

```

SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME
FROM EMPLOYEE E LEFT OUTER JOIN
 DEPARTMENT INNER JOIN EMPLOYEE M
ON MGRNO = M.EMPNO
ON E.WORKDEPT = DEPTNO

```

The inner join determines the last name for any manager identified in the DEPARTMENT table and the left outer join guarantees that each employee is listed even if a corresponding department is not found in DEPARTMENT.

## Examples of grouping sets, cube, and rollup

The queries in Example C1 through Example C4 use a subset of the rows in the SALES tables based on the predicate 'WEEK(SALES\_DATE) = 13'.

```

SELECT WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 SALES_PERSON, SALES AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13

```

which results in:

```

WEEK DAY_WEEK SALES_PERSON UNITS_SOLD

 13 6 LUCCHESI 3
 13 6 LUCCHESI 1
 13 6 LEE 2
 13 6 LEE 2
 13 6 LEE 3
 13 6 LEE 5
 13 6 GOUNOT 3
 13 6 GOUNOT 1
 13 6 GOUNOT 7
 13 7 LUCCHESI 1
 13 7 LUCCHESI 2
 13 7 LUCCHESI 1
 13 7 LEE 7
 13 7 LEE 3
 13 7 LEE 7
 13 7 LEE 4

```

|  |    |          |    |
|--|----|----------|----|
|  | 13 | 7 GOUNOT | 2  |
|  | 13 | 7 GOUNOT | 18 |
|  | 13 | 7 GOUNOT | 1  |

*Example C1:* Here is a query with a basic GROUP BY clause over 3 columns:

```

SELECT WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON
ORDER BY WEEK, DAY_WEEK, SALES_PERSON

```

This results in:

| WEEK  | DAY_WEEK | SALES_PERSON | UNITS_SOLD |
|-------|----------|--------------|------------|
| ----- | -----    | -----        | -----      |
| 13    | 6        | GOUNOT       | 11         |
| 13    | 6        | LEE          | 12         |
| 13    | 6        | LUCCHESI     | 4          |
| 13    | 7        | GOUNOT       | 21         |
| 13    | 7        | LEE          | 21         |
| 13    | 7        | LUCCHESI     | 4          |

*Example C2:* Produce the result based on two different grouping sets of rows from the SALES table.

```

SELECT WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY GROUPING SETS ((WEEK(SALES_DATE), SALES_PERSON),
 (DAYOFWEEK(SALES_DATE), SALES_PERSON))
ORDER BY WEEK, DAY_WEEK, SALES_PERSON

```

This results in:

| WEEK  | DAY_WEEK | SALES_PERSON | UNITS_SOLD |
|-------|----------|--------------|------------|
| ----- | -----    | -----        | -----      |
| 13    | -        | GOUNOT       | 32         |
| 13    | -        | LEE          | 33         |
| 13    | -        | LUCCHESI     | 8          |
| -     | 6        | GOUNOT       | 11         |
| -     | 6        | LEE          | 12         |
| -     | 6        | LUCCHESI     | 4          |
| -     | 7        | GOUNOT       | 21         |
| -     | 7        | LEE          | 21         |
| -     | 7        | LUCCHESI     | 4          |

The rows with WEEK 13 are from the first grouping set and the other rows are from the second grouping set.

*Example C3:* If you use the 3 distinct columns involved in the grouping sets of Example C2 and perform a ROLLUP, you can see grouping sets for (WEEK, DAY\_WEEK, SALES\_PERSON), (WEEK, DAY\_WEEK), (WEEK) and grand total.

```

SELECT WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY ROLLUP (WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON)
ORDER BY WEEK, DAY_WEEK, SALES_PERSON

```

This results in:

| WEEK | DAY_WEEK | SALES_PERSON | UNITS_SOLD |
|------|----------|--------------|------------|
| 13   | 6        | GOUNOT       | 11         |
| 13   | 6        | LEE          | 12         |
| 13   | 6        | LUCCHESI     | 4          |
| 13   | 6        | -            | 27         |
| 13   | 7        | GOUNOT       | 21         |
| 13   | 7        | LEE          | 21         |
| 13   | 7        | LUCCHESI     | 4          |
| 13   | 7        | -            | 46         |
| 13   | -        | -            | 73         |
| -    | -        | -            | 73         |

*Example C4:* If you run the same query as Example C3 only replace ROLLUP with CUBE, you can see additional grouping sets for (WEEK,SALES\_PERSON), (DAY\_WEEK,SALES\_PERSON), (DAY\_WEEK), (SALES\_PERSON) in the result.

```

SELECT WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY CUBE (WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON)
ORDER BY WEEK, DAY_WEEK, SALES_PERSON

```

This results in:

| WEEK | DAY_WEEK | SALES_PERSON | UNITS_SOLD |
|------|----------|--------------|------------|
| 13   | 6        | GOUNOT       | 11         |
| 13   | 6        | LEE          | 12         |
| 13   | 6        | LUCCHESI     | 4          |
| 13   | 6        | -            | 27         |
| 13   | 7        | GOUNOT       | 21         |
| 13   | 7        | LEE          | 21         |
| 13   | 7        | LUCCHESI     | 4          |
| 13   | 7        | -            | 46         |
| 13   | -        | GOUNOT       | 32         |
| 13   | -        | LEE          | 33         |
| 13   | -        | LUCCHESI     | 8          |
| 13   | -        | -            | 73         |
| -    | 6        | GOUNOT       | 11         |
| -    | 6        | LEE          | 12         |
| -    | 6        | LUCCHESI     | 4          |
| -    | 6        | -            | 27         |
| -    | 7        | GOUNOT       | 21         |
| -    | 7        | LEE          | 21         |
| -    | 7        | LUCCHESI     | 4          |
| -    | 7        | -            | 46         |
| -    | -        | GOUNOT       | 32         |
| -    | -        | LEE          | 33         |
| -    | -        | LUCCHESI     | 8          |
| -    | -        | -            | 73         |

*Example C5:* Obtain a result set which includes a grand-total of selected rows from the SALES table together with a group of rows aggregated by SALES\_PERSON and MONTH.

```

SELECT SALES_PERSON,
 MONTH(SALES_DATE) AS MONTH,
 SUM(SALES) AS UNITS_SOLD
FROM SALES

```

```

GROUP BY GROUPING SETS ((SALES_PERSON, MONTH(SALES_DATE)),
 ()
)
ORDER BY SALES_PERSON, MONTH

```

This results in:

| SALES_PERSON | MONTH | UNITS_SOLD |
|--------------|-------|------------|
| GOUNOT       | 3     | 35         |
| GOUNOT       | 4     | 14         |
| GOUNOT       | 12    | 1          |
| LEE          | 3     | 60         |
| LEE          | 4     | 25         |
| LEE          | 12    | 6          |
| LUCCHESSI    | 3     | 9          |
| LUCCHESSI    | 4     | 4          |
| LUCCHESSI    | 12    | 1          |
| -            | -     | 155        |

*Example C6:* This example shows two simple ROLLUP queries followed by a query which treats the two ROLLUPs as grouping sets in a single result set and specifies row ordering for each column involved in the grouping sets.

*Example C6-1:*

```

SELECT WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY ROLLUP (WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE))
ORDER BY WEEK, DAY_WEEK

```

results in:

| WEEK | DAY_WEEK | UNITS_SOLD |
|------|----------|------------|
| 13   | 6        | 27         |
| 13   | 7        | 46         |
| 13   | -        | 73         |
| 14   | 1        | 31         |
| 14   | 2        | 43         |
| 14   | -        | 74         |
| 53   | 1        | 8          |
| 53   | -        | 8          |
| -    | -        | 155        |

*Example C6-2:*

```

SELECT MONTH(SALES_DATE) AS MONTH,
 REGION,
 SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY ROLLUP (MONTH(SALES_DATE), REGION);
ORDER BY MONTH, REGION

```

results in:

| MONTH | REGION        | UNITS_SOLD |
|-------|---------------|------------|
| 3     | Manitoba      | 22         |
| 3     | Ontario-North | 8          |
| 3     | Ontario-South | 34         |
| 3     | Quebec        | 40         |
| 3     | -             | 104        |
| 4     | Manitoba      | 17         |
| 4     | Ontario-North | 1          |

```

4 Ontario-South 14
4 Quebec 11
4 - 43
12 Manitoba 2
12 Ontario-South 4
12 Quebec 2
12 - 8
- - 155

```

Example C6-3:

```

SELECT WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 MONTH(SALES_DATE) AS MONTH,
 REGION,
 SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY GROUPING SETS (ROLLUP(WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE)),
 ROLLUP(MONTH(SALES_DATE), REGION))
ORDER BY WEEK, DAY_WEEK, MONTH, REGION

```

results in:

| WEEK | DAY_WEEK | MONTH | REGION           | UNITS_SOLD |
|------|----------|-------|------------------|------------|
| 13   | 6        | -     | -                | 27         |
| 13   | 7        | -     | -                | 46         |
| 13   | -        | -     | -                | 73         |
| 14   | 1        | -     | -                | 31         |
| 14   | 2        | -     | -                | 43         |
| 14   | -        | -     | -                | 74         |
| 53   | 1        | -     | -                | 8          |
| 53   | -        | -     | -                | 8          |
| -    | -        | -     | 3 Manitoba       | 22         |
| -    | -        | -     | 3 Ontario-North  | 8          |
| -    | -        | -     | 3 Ontario-South  | 34         |
| -    | -        | -     | 3 Quebec         | 40         |
| -    | -        | -     | 3 -              | 104        |
| -    | -        | -     | 4 Manitoba       | 17         |
| -    | -        | -     | 4 Ontario-North  | 1          |
| -    | -        | -     | 4 Ontario-South  | 14         |
| -    | -        | -     | 4 Quebec         | 11         |
| -    | -        | -     | 4 -              | 43         |
| -    | -        | -     | 12 Manitoba      | 2          |
| -    | -        | -     | 12 Ontario-South | 4          |
| -    | -        | -     | 12 Quebec        | 2          |
| -    | -        | -     | 12 -             | 8          |
| -    | -        | -     | -                | 155        |
| -    | -        | -     | -                | 155        |

Using the two ROLLUPs as grouping sets causes the result to include duplicate rows. There are even two grand total rows.

Observe how the use of ORDER BY has affected the results:

- In the first grouped set, week 53 has been repositioned to the end.
- In the second grouped set, month 12 has now been positioned to the end and the regions now appear in alphabetic order.
- Null values are sorted high.

Example C7: In queries that perform multiple ROLLUPs in a single pass (such as Example C6-3) you may want to be able to indicate which grouping set produced each row. The following steps demonstrate how to provide a column (called GROUP) which indicates the origin of each row in the result set. By origin, we mean which one of the two grouping sets produced the row in the result set.



Step 1: Introduce a way of "generating" new data values, using a query which selects from a VALUES clause (which is an alternate form of a fullselect). This query shows how a table can be derived called "X" having 2 columns "R1" and "R2" and 1 row of data.

```
SELECT R1,R2
FROM (VALUES('GROUP 1','GROUP 2')) AS X(R1,R2);
```

results in:

```
R1 R2

GROUP 1 GROUP 2
```

Step 2: Form the cross product of this table "X" with the SALES table. This add columns "R1" and "R2" to every row.

```
SELECT R1, R2, WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 MONTH(SALES_DATE) AS MONTH,
 REGION,
 SALES AS UNITS_SOLD
FROM SALES, (VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
```

This add columns "R1" and "R2" to every row.

Step 3: Now we can combine these columns with the grouping sets to include these columns in the rollup analysis.

```
SELECT R1, R2,
 WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 MONTH(SALES_DATE) AS MONTH,
 REGION, SUM(SALES) AS UNITS_SOLD
FROM SALES, (VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
GROUP BY GROUPING SETS ((R1, ROLLUP(WEEK(SALES_DATE),
 DAYOFWEEK(SALES_DATE))),
 (R2, ROLLUP(MONTH(SALES_DATE), REGION)))
ORDER BY WEEK, DAY_WEEK, MONTH, REGION
```

results in:

| R1      | R2      | WEEK | DAY_WEEK | MONTH | REGION           | UNITS_SOLD |
|---------|---------|------|----------|-------|------------------|------------|
| GROUP 1 | -       | 13   | 6        | -     | -                | 27         |
| GROUP 1 | -       | 13   | 7        | -     | -                | 46         |
| GROUP 1 | -       | 13   | -        | -     | -                | 73         |
| GROUP 1 | -       | 14   | 1        | -     | -                | 31         |
| GROUP 1 | -       | 14   | 2        | -     | -                | 43         |
| GROUP 1 | -       | 14   | -        | -     | -                | 74         |
| GROUP 1 | -       | 53   | 1        | -     | -                | 8          |
| GROUP 1 | -       | 53   | -        | -     | -                | 8          |
| -       | GROUP 2 | -    | -        | -     | 3 Manitoba       | 22         |
| -       | GROUP 2 | -    | -        | -     | 3 Ontario-North  | 8          |
| -       | GROUP 2 | -    | -        | -     | 3 Ontario-South  | 34         |
| -       | GROUP 2 | -    | -        | -     | 3 Quebec         | 40         |
| -       | GROUP 2 | -    | -        | -     | 3 -              | 104        |
| -       | GROUP 2 | -    | -        | -     | 4 Manitoba       | 17         |
| -       | GROUP 2 | -    | -        | -     | 4 Ontario-North  | 1          |
| -       | GROUP 2 | -    | -        | -     | 4 Ontario-South  | 14         |
| -       | GROUP 2 | -    | -        | -     | 4 Quebec         | 11         |
| -       | GROUP 2 | -    | -        | -     | 4 -              | 43         |
| -       | GROUP 2 | -    | -        | -     | 12 Manitoba      | 2          |
| -       | GROUP 2 | -    | -        | -     | 12 Ontario-South | 4          |
| -       | GROUP 2 | -    | -        | -     | 12 Quebec        | 2          |

```

- GROUP 2 - - 12 - 8
- GROUP 2 - - - - 155
GROUP 1 - - - - - 155

```

*Step 4:* Notice that because R1 and R2 are used in different grouping sets, whenever R1 is non-null in the result, R2 is null and whenever R2 is non-null in the result, R1 is null. That means you can consolidate these columns into a single column using the COALESCE function. You can also use this column in the ORDER BY clause to keep the results of the two grouping sets together.

```

SELECT COALESCE(R1,R2) AS GROUP,
 WEEK(SALES_DATE) AS WEEK,
 DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
 MONTH(SALES_DATE) AS MONTH,
 REGION, SUM(SALES) AS UNITS_SOLD
FROM SALES, (VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
GROUP BY GROUPING SETS ((R1, ROLLUP(WEEK(SALES_DATE),
 DAYOFWEEK(SALES_DATE))),
 (R2, ROLLUP(MONTH(SALES_DATE), REGION)))
ORDER BY GROUP, WEEK, DAY_WEEK, MONTH, REGION;

```

results in:

| GROUP   | WEEK | DAY_WEEK | MONTH | REGION           | UNITS_SOLD |
|---------|------|----------|-------|------------------|------------|
| GROUP 1 |      | 13       | 6     | - -              | 27         |
| GROUP 1 |      | 13       | 7     | - -              | 46         |
| GROUP 1 |      | 13       | -     | - -              | 73         |
| GROUP 1 |      | 14       | 1     | - -              | 31         |
| GROUP 1 |      | 14       | 2     | - -              | 43         |
| GROUP 1 |      | 14       | -     | - -              | 74         |
| GROUP 1 |      | 53       | 1     | - -              | 8          |
| GROUP 1 |      | 53       | -     | - -              | 8          |
| GROUP 1 |      | -        | -     | - -              | 155        |
| GROUP 2 |      | -        | -     | 3 Manitoba       | 22         |
| GROUP 2 |      | -        | -     | 3 Ontario-North  | 8          |
| GROUP 2 |      | -        | -     | 3 Ontario-South  | 34         |
| GROUP 2 |      | -        | -     | 3 Quebec         | 40         |
| GROUP 2 |      | -        | -     | 3 -              | 104        |
| GROUP 2 |      | -        | -     | 4 Manitoba       | 17         |
| GROUP 2 |      | -        | -     | 4 Ontario-North  | 1          |
| GROUP 2 |      | -        | -     | 4 Ontario-South  | 14         |
| GROUP 2 |      | -        | -     | 4 Quebec         | 11         |
| GROUP 2 |      | -        | -     | 4 -              | 43         |
| GROUP 2 |      | -        | -     | 12 Manitoba      | 2          |
| GROUP 2 |      | -        | -     | 12 Ontario-South | 4          |
| GROUP 2 |      | -        | -     | 12 Quebec        | 2          |
| GROUP 2 |      | -        | -     | 12 -             | 8          |
| GROUP 2 |      | -        | -     | - -              | 155        |

*Example C8:* The following example illustrates the use of various aggregate functions when performing a CUBE. The example also makes use of cast functions and rounding to produce a decimal result with reasonable precision and scale.

```

SELECT MONTH(SALES_DATE) AS MONTH,
 REGION,
 SUM(SALES) AS UNITS_SOLD,
 MAX(SALES) AS BEST_SALE,
 CAST(ROUND(AVG(DECIMAL(SALES)),2) AS DECIMAL(5,2)) AS AVG_UNITS_SOLD
FROM SALES
GROUP BY CUBE(MONTH(SALES_DATE),REGION)
ORDER BY MONTH, REGION

```

This results in:

| MONTH | REGION        | UNITS_SOLD | BEST_SALE | AVG_UNITS_SOLD |
|-------|---------------|------------|-----------|----------------|
| 3     | Manitoba      | 22         | 7         | 3.14           |
| 3     | Ontario-North | 8          | 3         | 2.67           |
| 3     | Ontario-South | 34         | 14        | 4.25           |
| 3     | Quebec        | 40         | 18        | 5.00           |
| 3     | -             | 104        | 18        | 4.00           |
| 4     | Manitoba      | 17         | 9         | 5.67           |
| 4     | Ontario-North | 1          | 1         | 1.00           |
| 4     | Ontario-South | 14         | 8         | 4.67           |
| 4     | Quebec        | 11         | 8         | 5.50           |
| 4     | -             | 43         | 9         | 4.78           |
| 12    | Manitoba      | 2          | 2         | 2.00           |
| 12    | Ontario-South | 4          | 3         | 2.00           |
| 12    | Quebec        | 2          | 1         | 1.00           |
| 12    | -             | 8          | 3         | 1.60           |
| -     | Manitoba      | 41         | 9         | 3.73           |
| -     | Ontario-North | 9          | 3         | 2.25           |
| -     | Ontario-South | 52         | 14        | 4.00           |
| -     | Quebec        | 53         | 18        | 4.42           |
| -     | -             | 155        | 18        | 3.87           |

---

## TRANSFER OWNERSHIP

The TRANSFER OWNERSHIP statement transfers ownership of a database object.

### Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- Ownership of the object
- SECADM authority

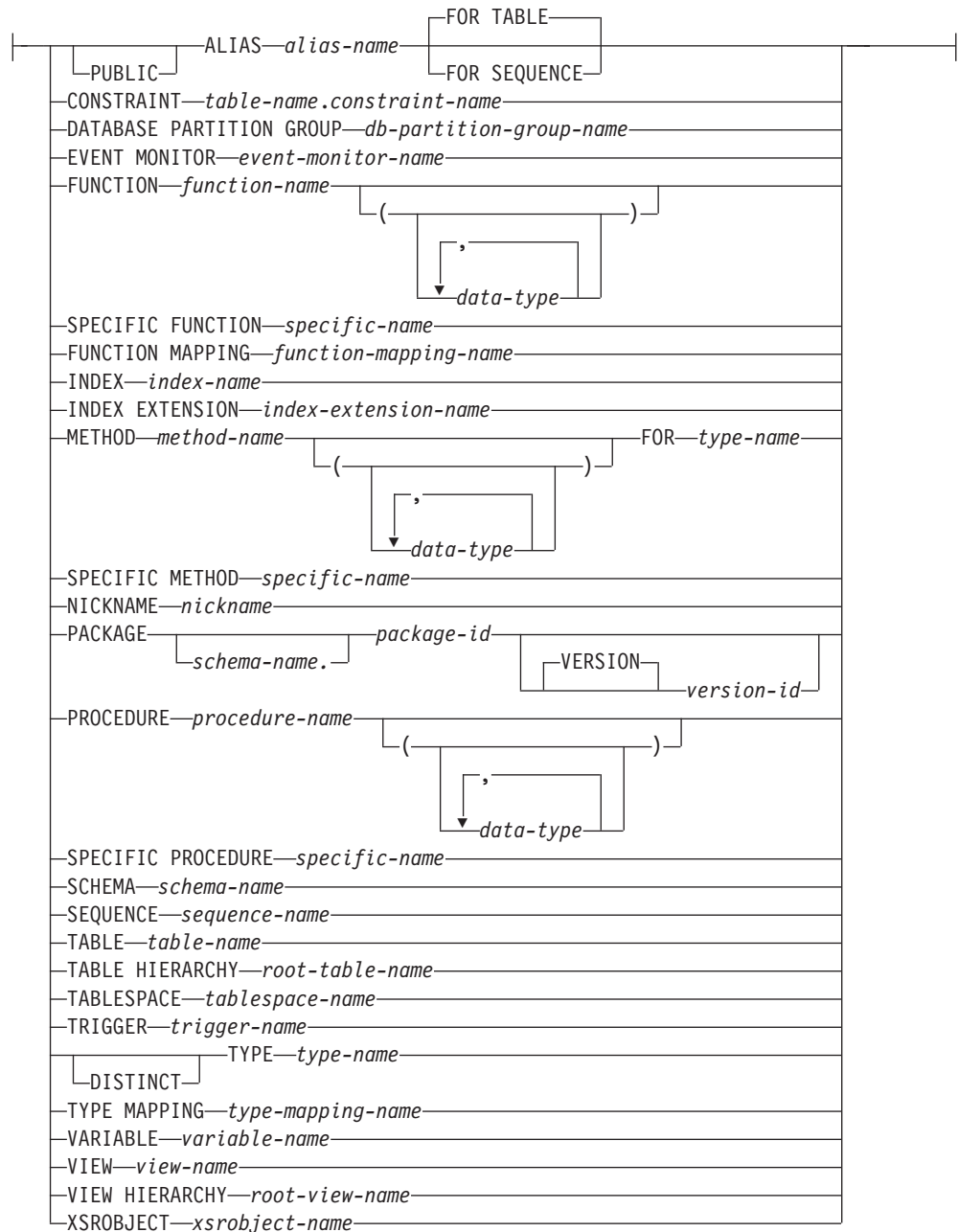
### Syntax

```

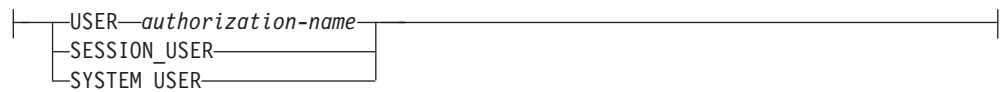
▶▶—TRANSFER OWNERSHIP OF—| objects |—TO—| new-owner |—PRESERVE PRIVILEGES—▶▶

```

**objects:**



**new-owner:**



**Description**

**ALIAS *alias-name***

Identifies the alias that is to have its ownership transferred. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704). If PUBLIC is specified, the *alias-name* must identify a public alias that exists at the current server (SQLSTATE 42704).

|  
|  
| **FOR TABLE, or FOR SEQUENCE**

| Specifies the object type for the alias.

| **FOR TABLE**

| The alias is for a table, view, or nickname. When ownership of the alias  
| is transferred, the value in the OWNER column for the alias in the  
| SYSCAT.TABLES catalog view is replaced with the authorization ID of  
| the new owner.

| **FOR SEQUENCE**

| The alias is for a sequence. When ownership of the alias is transferred,  
| the value in the OWNER column for the alias in the  
| SYSCAT.SEQUENCES catalog view is replaced with the authorization  
| ID of the new owner.

| **CONSTRAINT** *table-name.constraint-name*

| Identifies the constraint that is to have its ownership transferred. The  
| *table-name.constraint-name* combination must identify a constraint and the table  
| that it constrains. The *constraint-name* must identify a constraint that is  
| described in the catalog (SQLSTATE 42704).

| When ownership of the constraint is transferred, the value in the OWNER  
| column for the constraint in the SYSCAT.TABCONST catalog view is replaced  
| with the authorization ID of the new owner.

- If the constraint is a FOREIGN KEY constraint, the OWNER column in the SYSCAT.REFERENCES catalog view is replaced with the authorization ID of the new owner.
- If the constraint is a PRIMARY KEY or UNIQUE constraint, the OWNER column in the SYSCAT.INDEXES catalog view for the index that was created implicitly for this constraint is replaced with the authorization ID of the new owner. If the index existed, and it is reused in this case, the owner of the index is not changed.

| **DATABASE PARTITION GROUP** *db-partition-group-name*

| Identifies the database partition group that is to have its ownership  
| transferred. The *db-partition-group-name* must identify a database partition  
| group that is described in the catalog (SQLSTATE 42704).

| When ownership of the database partition group is transferred, the value in  
| the OWNER column for the database partition group in the  
| SYSCAT.DBPARTITIONGROUPS catalog view is replaced with the  
| authorization ID of the new owner.

| **EVENT MONITOR** *event-monitor-name*

| Identifies the event monitor that is to have its ownership transferred. The  
| *event-monitor-name* must identify an event monitor that is described in the  
| catalog (SQLSTATE 42704).

| When ownership of the event monitor is transferred, the value in the OWNER  
| column for the event monitor in the SYSCAT.EVENTMONITORS catalog view  
| is replaced with the authorization ID of the new owner.

| If the identified event monitor is active, an error is returned (SQLSTATE  
| 429BT).

| If there are event files in the target path of a WRITE TO FILE event monitor  
| whose ownership is being transferred, the event files are not deleted.

| When ownership of WRITE TO TABLE event monitors is transferred, table  
| information in the SYSCAT.EVENTTABLES catalog view is retained.

## FUNCTION

Identifies the function that is to have its ownership transferred. The specified function instance must be a user-defined function or function template that is described in the catalog. Ownership of functions that are implicitly generated by the CREATE TYPE (Distinct) and CREATE TYPE (Structured) statements cannot be transferred (SQLSTATE 429BT).

There are several different ways to identify the function instance.

### FUNCTION *function-name*

Identifies the particular function that is to have its ownership transferred, and is valid only if there is exactly one function instance with that *function-name*. The function thus identified can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the function in the named or implied schema, an error is returned (SQLSTATE 42725).

### FUNCTION *function-name (data-type,...)*

Provides the function signature, which uniquely identifies the function whose ownership is to be transferred. The function selection algorithm is not used.

#### *function-name*

Specifies the name of the function whose ownership is to be transferred. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names.

#### *(data-type,...)*

Specified data types must match the types and positions that were specified on the CREATE FUNCTION statement. The number of data types and the logical concatenation of the data types are used to identify the specific function whose ownership is to be transferred.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because  $0 < n < 25$  means REAL, and  $24 < n < 54$  means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

The FOR BIT DATA attribute is not considered to be part of the signature for matching purposes. So, for example, a CHAR FOR BIT

DATA specified in the signature would match a function defined with CHAR only; the reverse would also be true.

If no function with the specified signature exists in the named or implied schema, an error is returned (SQLSTATE 42883).

When ownership of the function is transferred, the value in the OWNER column for the function in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

**SPECIFIC FUNCTION** *specific-name*

Identifies the particular user-defined function that is to have its ownership transferred, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

When ownership of the specific function is transferred, the value in the OWNER column for the specific function in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

**FUNCTION MAPPING** *function-mapping-name*

Identifies the function mapping that is to have its ownership transferred. The *function-mapping-name* must identify a function mapping that is described in the catalog (SQLSTATE 42704).

When ownership of the function mapping is transferred, the value in the OWNER column for the function mapping in the SYSCAT.FUNCMAPPINGS catalog view is replaced with the authorization ID of the new owner.

**INDEX** *index-name*

Identifies the index or index specification that is to have its ownership transferred. The *index-name* must identify an index or index specification that is described in the catalog (SQLSTATE 42704).

When ownership of the index is transferred, the value in the OWNER column for the index in the SYSCAT.INDEXES catalog view is replaced with the authorization ID of the new owner.

Ownership of an index cannot be transferred if the table on which the index is defined is a global temporary table (SQLSTATE 429BT).

**INDEX EXTENSION** *index-extension-name*

Identifies the index extension that is to have its ownership transferred. The *index-extension-name* must identify an index extension that is described in the catalog (SQLSTATE 42704).

When ownership of the index extension is transferred, the value in the OWNER column for the index extension in the SYSCAT.INDEXEXTENSIONS catalog view is replaced with the authorization ID of the new owner.

**METHOD**

Identifies the method that is to have its ownership transferred. The method body specified must be a method that is described in the catalog (SQLSTATE 42704). The ownership of methods that are implicitly generated by the CREATE TYPE statement cannot be transferred (SQLSTATE 429BT).

There are several different ways to identify the method body.



**METHOD** *method-name*

Identifies the particular method that is to have its ownership transferred, and is valid only if there is exactly one method instance with name *method-name* and subject type *type-name*. Thus, the method identified can have any number of parameters. If no method by this name exists for the type *type-name*, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the method for the named data type, an error is returned (SQLSTATE 42725).

**METHOD** *method-name (data-type,...)*

Provides the method signature, which uniquely identifies the method whose ownership is to be transferred. The method selection algorithm is not used.

*method-name*

Specifies the name of the method whose ownership is to be transferred. The name must be an unqualified identifier.

*(data-type, ...)*

Specified data types must match the types and positions that were specified on the CREATE TYPE or ALTER TYPE statement. The number of data types and the logical concatenation of the data types are used to identify the specific method instance whose ownership is to be transferred.

If *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because  $0 < n < 25$  means REAL and  $24 < n < 54$  means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no method with the specified signature exists for the named data type, an error is returned (SQLSTATE 42883).

**FOR** *type-name*

Names the type for which the specified method is to have its ownership transferred. The name must identify a type that is described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified type names.

When ownership of the method is transferred, the value in the OWNER column for the method in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

**SPECIFIC METHOD** *specific-name*

Identifies the particular method that is to have its ownership transferred. If the

specific name is unqualified, the CURRENT SCHEMA special register is used as a qualifier for an unqualified specific name in dynamic SQL. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for an unqualified specific name. The *specific-name* must identify a method; otherwise, an error is returned (SQLSTATE 42704).

When ownership of the specific method is transferred, the value in the OWNER column for the specific method in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

**NICKNAME** *nickname*

Identifies the nickname that is to have its ownership transferred. The *nickname* must be a nickname that is described in the catalog (SQLSTATE 42704).

When ownership of the nickname is transferred, the value in the OWNER column for the nickname in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.

**PACKAGE** *schema-name.package-id*

Identifies the package that is to have its ownership transferred. If a schema name is not specified, the package identifier is implicitly qualified by the default schema. The schema name and package identifier, together with the implicitly or explicitly specified version identifier, must identify a package that is described in the catalog (SQLSTATE 42704).

**VERSION** *version-id*

Identifies which package version is to have its ownership transferred. If a value is not specified, the version defaults to the empty string, and the ownership of this package is transferred. If multiple packages with the same package name but different versions exist, only the ownership of the package whose *version-id* is specified in the TRANSFER OWNERSHIP statement is transferred. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

When ownership of the package is transferred, the value in the BOUNDBY column for the package in the SYSCAT.PACKAGES catalog view is replaced with the authorization ID of the new owner.

The ownership of packages that are associated with SQL procedures cannot be transferred (SQLSTATE 429BT).

**PROCEDURE**

Identifies the procedure that is to have its ownership transferred. The procedure instance specified must be a procedure that is described in the catalog.

There are several different ways to identify the procedure instance.

**PROCEDURE** *procedure-name*

Identifies the particular procedure that is to have its ownership transferred, and is valid only if there is exactly one procedure with the *procedure-name* in the schema. The procedure thus identified can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT

SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names. If no procedure by this name exists in the named or implied schema, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the procedure in the named or implied schema, an error is returned (SQLSTATE 42725).

**PROCEDURE** *procedure-name (data-type,...)*

Provides the procedure signature, which uniquely identifies the procedure whose ownership is to be transferred.

*procedure-name*

Specifies the procedure name of the procedure whose ownership is to be transferred. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names.

*(data-type,...)*

Specified data types must match the types and positions that were specified on the CREATE PROCEDURE statement. The number of data types and the logical concatenation of the data types are used to identify the specific procedure whose ownership is to be transferred.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because  $0 < n < 25$  means REAL and  $24 < n < 54$  means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in the named or implied schema, an error is returned (SQLSTATE 42883).

When ownership of the procedure is transferred, the value in the OWNER column for the procedure in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

Transferring ownership of an SQL procedure that has an associated package also implicitly transfers ownership of the package to the new owner.

**SPECIFIC PROCEDURE** *specific-name*

Identifies the particular procedure that is to have its ownership transferred, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for

unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

When ownership of the specific procedure is transferred, the value in the OWNER column for the specific procedure in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

**SCHEMA** *schema-name*

Identifies the schema that is to have its ownership transferred. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704).

When ownership of the schema is transferred, the value in the OWNER column and the DEFINER column for the schema in the SYSCAT.SCHEMATA catalog view is replaced with the authorization ID of the new owner.

Ownership of system-defined schemas (where the definer is SYSIBM) cannot be transferred (SQLSTATE 42832).

**SEQUENCE** *sequence-name*

Identifies the sequence that is to have its ownership transferred. The *sequence-name* must identify a sequence that is described in the catalog (SQLSTATE 42704).

When ownership of the sequence is transferred, the value in the OWNER column for the schema in the SYSCAT.SEQUENCES catalog view is replaced with the authorization ID of the new owner.

**TABLE** *table-name*

Identifies the table that is to have its ownership transferred. The *table-name* must identify a table that exists in the database (SQLSTATE 42704) and must not identify a declared temporary table (SQLSTATE 42995).

When ownership of the table is transferred:

- The value in the OWNER column for the table in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the table in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

Ownership of subtables in a table hierarchy cannot be transferred (SQLSTATE 429BT).

In a federated system, ownership of a remote table that was created using transparent DDL can be transferred. Transferring the ownership of a remote table will not transfer ownership of the nickname that is associated with the table. Ownership of such a nickname can be transferred explicitly using the TRANSFER OWNERSHIP statement.

**TABLE HIERARCHY** *root-table-name*

Identifies the typed table that is the root table in a typed table hierarchy that is to have its ownership transferred. The *root-table-name* must identify a typed table that is the root table in the typed table hierarchy (SQLSTATE 428DR), and must refer to a typed table that exists in the database (SQLSTATE 42704).

When ownership of the table hierarchy is transferred:

- The value in the OWNER column for the root table and all of its subtables in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.

- The value in the OWNER column for all dependent objects on the table and all of its subtables in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

**TABLESPACE** *tablespace-name*

Identifies the table space that is to have its ownership transferred. The *tablespace-name* must identify a table space that is described in the catalog (SQLSTATE 42704).

When ownership of the table space is transferred, the value in the OWNER column for the table space in the SYSCAT.TABLESPACES catalog view is replaced with the authorization ID of the new owner.

**TRIGGER** *trigger-name*

Identifies the trigger that is to have its ownership transferred. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704).

When ownership of the trigger is transferred, the value in the OWNER column for the trigger in the SYSCAT.TRIGGERS catalog view is replaced with the authorization ID of the new owner.

**TYPE** *type-name*

Identifies the user-defined type that is to have its ownership transferred. The *type-name* must identify a type that is described in the catalog (SQLSTATE 42704). If DISTINCT is specified, *type-name* must identify a distinct type that is described in the catalog (SQLSTATE 42704).

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names.

When ownership of the type is transferred, the value in the OWNER column for the type in the SYSCAT.DATATYPES catalog view is replaced with the authorization ID of the new owner.

**TYPE MAPPING** *type-mapping-name*

Identifies the user-defined data type mapping that is to have its ownership transferred. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704).

When ownership of the type mapping is transferred, the value in the OWNER column for the type mapping in the SYSCAT.TYPEMAPPINGS catalog view is replaced with the authorization ID of the new owner.

**VARIABLE** *variable-name*

Indicates that the object whose ownership is to be transferred is a created global variable. The *variable-name* must identify a global variable that exists at the current server (SQLSTATE 42704).

When the global variable is transferred, the value in the OWNER column for the global variable in the SYSCAT.VARIABLES catalog view is replaced with the authorization ID of the new owner.

**VIEW** *view-name*

Identifies the view that is to have its ownership transferred. The *view-name* must identify a view that exists in the database (SQLSTATE 42704).

When ownership of the view is transferred:

- The value in the OWNER column for the view in the SYSCAT.VIEWS catalog view is replaced with the authorization ID of the new owner.

- The value in the OWNER column for all dependent objects on the view in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

The ownership of a subview in a view hierarchy cannot be transferred (SQLSTATE 429BT).

#### **VIEW HIERARCHY** *root-view-name*

Identifies the typed view that is the root view in a typed view hierarchy that is to have its ownership transferred. The *root-view-name* must identify a typed view that is the root view in the typed view hierarchy (SQLSTATE 428DR), and must refer to a typed view that exists in the database (SQLSTATE 42704).

When ownership of the view hierarchy is transferred:

- The value in the OWNER column for the root view and all of its subviews in the SYSCAT.VIEWS catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the view and all of its subviews in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

#### **XSROBJECT** *xsobject-name*

Identifies the XSR object that is to have its ownership transferred. The *xsobject-name* must identify an XSR object that is described in the catalog (SQLSTATE 42704).

When ownership of the XSR object is transferred, the value in the OWNER column for the XSR object in the SYSCAT.XSROBJECTS catalog view is replaced with the authorization ID of the new owner.

#### **USER** *authorization-name*

Specifies the authorization ID to which ownership of the object is being transferred.

#### **SESSION\_USER**

Specifies that the value of the SESSION\_USER special register is to be used as the authorization ID to which ownership of the object is being transferred.

#### **SYSTEM\_USER**

Specifies that the value of the SYSTEM\_USER special register is to be used as the authorization ID to which ownership of the object is being transferred.

#### **PRESERVE PRIVILEGES**

Specifies that the current owner of an object that is to have its ownership transferred will continue to hold any existing privileges on the object after the transfer. For example, any privileges that were granted to the creator of a view when that view was created continue to be held by the original owner even after ownership has been transferred to another user.

### **Rules**

- Ownership of most system-defined objects (where the owner is SYSIBM) cannot be transferred (SQLSTATE 42832). However, you can transfer ownership of implicitly created schema objects that have SYSIBM in the OWNER column and do not have SYSIBM in the DEFINER column.
- Ownership of schemas whose name starts with 'SYS' cannot be transferred (SQLSTATE 42832).
- Ownership of the following objects cannot be explicitly transferred (SQLSTATE 429BT):



- Subtables in a table hierarchy (they are transferred with the root hierarchy table)
- Subviews in a view hierarchy (they are transferred with the root hierarchy view)
- Indexes that are defined on global temporary tables
- Methods or functions that are implicitly generated when a user-defined type is created
- Packages that depend on SQL procedures (they are transferred with the SQL procedure)
- Event monitors that are active (they can be transferred when they are not active)
- An authorization ID that has SECADM authority cannot transfer the ownership of an object to itself, if it is not already the owner of the object (SQLSTATE 42502).

## Notes

- All privileges that the current owner has that were granted as part of the creation of the object are transferred to the new owner. If the current owner has had a privilege on the object revoked, and that privilege was subsequently granted back, the privilege is not transferred. For implicitly created schema objects that have not already been transferred, the new owner is granted CREATEIN, DROPIN, and ALTERIN privileges on the schema and can also grant these privileges to other users.
- When the ownership of a database object is transferred, the new owner must have the set of privileges on the base objects, as indicated by the object's dependencies, that are required to maintain the object's existence unchanged. The new owner does not need the privileges required to create the object if those privileges are not required to maintain the object's existence.

For example:

- Consider a view with SELECT and INSERT dependencies on an underlying table. The privileges held by the new owner of the view must include at least SELECT (with or without the GRANT OPTION) and INSERT (with or without the GRANT OPTION) for the ownership transfer to be successful. If the dependencies were SELECT WITH GRANT OPTION and INSERT WITH GRANT OPTION, the privileges held by the new owner of the view must include at least SELECT WITH GRANT OPTION and INSERT WITH GRANT OPTION.
- Consider a view with a dependency on a routine. The privileges held by the new owner of the view must include at least EXECUTE on the dependent routine.
- Consider a trigger with a dependency on a table. The privileges held by the new owner of the trigger must include the same set of privileges on the table that are indicated by the trigger's dependencies. ALTER privilege on the table on which the trigger is defined is not required.

The following table lists the system catalog views that describe the objects on which other database objects depend.

*Table 17. Catalog Views that Describe Objects on which Other Objects Depend*

| Database Object | System Catalog View                                            |
|-----------------|----------------------------------------------------------------|
| CONSTRAINT      | SYSCAT.CONSTDEP                                                |
| FUNCTION        | SYSCAT.ROUTINEDEP; SYSCAT.ROUTINES<br>(for a sourced function) |



Table 17. Catalog Views that Describe Objects on which Other Objects Depend (continued)

| Database Object | System Catalog View      |
|-----------------|--------------------------|
| INDEX           | SYSCAT.INDEXDEP          |
| INDEX EXTENSION | SYSCAT.INDEXEXTENSIONDEP |
| METHOD          | SYSCAT.ROUTINEDEP        |
| PACKAGE         | SYSCAT.PACKAGEDEP        |
| PROCEDURE       | SYSCAT.ROUTINEDEP        |
| TABLE           | SYSCAT.TABDEP            |
| TRIGGER         | SYSCAT.TRIGDEP           |
| VIEW            | SYSCAT.TABDEP            |
| XSRBJECT        | SYSCAT.XSRBJECTDEP       |

If ownership of a database object that depends on another object is to be transferred successfully, the new owner of the database object must hold certain privileges on the dependent object of that dependency:

- If the dependent object is a sequence, the new owner must have the USAGE privilege on that sequence.
- If the dependent object is a function, method, or procedure, the new owner must have the EXECUTE privilege on that function, method, or procedure.
- If the dependent object is a package, the new owner must have the EXECUTE privilege on that package.
- If the dependent object is an XSR object, the new owner must have the USAGE privilege on that XSR object.

For any other dependent object of a dependency, use the TABAUTH column in the appropriate system catalog view to determine what privileges the new owner must hold.

- If an attempt is made to transfer ownership of an object to its owner, a warning is returned (SQLSTATE 01676).
- Ownership of the following database objects cannot be transferred, because these objects have no owner: audit policies, buffer pools, roles, security labels, security label components, security policies, servers, transformation functions, trusted contexts, user mappings, and wrappers. Note that there is no OWNER column in the SYSCAT.AUDITPOLICIES, SYSCAT.BUFFERPOOLS, SYSCAT.CONTEXTS, SYSCAT.ROLES, SYSCAT.SECURITYLABELS, SYSCAT.SECURITYLABELCOMPONENTS, SYSCAT.SECURITYPOLICIES, SYSCAT.SERVERS, SYSCAT.TRANSFORMS, SYSCAT.USEROPTIONS, and SYSCAT.WRAPPERS catalog views.
- The schema name of an object whose ownership was transferred does not automatically change.
- **Compatibilities:** For consistency with other SQL statements:
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
  - SYNONYM can be specified in place of ALIAS

## Examples

*Example 1:* Transfer ownership of table T1 to PAUL.

```
TRANSFER OWNERSHIP OF TABLE WALID.T1
TO USER PAUL PRESERVE PRIVILEGES
```

The value in the OWNER column for the table WALID.T1 in the SYSCAT.TABLES catalog view is replaced with 'PAUL'. Paul is implicitly granted the following privileges on table WALID.T1 (assuming that the previous owner of the table did not lose any privileges on it): CONTROL and ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE, REFERENCE (WITH GRANT OPTION).

*Example 2:* Assume that JOHN creates tables T1 and T2, and that MIKE holds SELECT privilege on tables JOHN.T1 and JOHN.T2. MIKE creates view V1 that depends on tables JOHN.T1 and JOHN.T2. Transfer ownership of view V1 to HENRY, who has DBADM authority.

```
TRANSFER OWNERSHIP OF VIEW V1
TO USER HENRY PRESERVE PRIVILEGES
```

The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'HENRY'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'HENRY', and TABNAME = 'V1'.

*Example 3:* Assume that HENRY, who holds DBADM authority, creates a trigger TR1 that depends on table T1. Transfer ownership of trigger TR1 to WALID, who does not hold DBADM authority.

```
TRANSFER OWNERSHIP OF TRIGGER TR1
TO USER WALID PRESERVE PRIVILEGES
```

Ownership of the trigger is transferred successfully, even though Walid does not hold DBADM authority.

*Example 4:* Assume that JOHN creates tables T1 and T2, and that MIKE holds SELECT privilege on table JOHN.T1 and CONTROL privilege on table JOHN.T2. PAUL holds SELECT privilege on tables JOHN.T1 and JOHN.T2. MIKE creates view V1 that depends on tables JOHN.T1 and JOHN.T2. The view has an entry for the SELECT privilege in SYSCAT.TABAUTH and two SELECT dependencies in SYSCAT.TABDEP for tables JOHN.T1 and JOHN.T2. Transfer ownership of view V1 to PAUL, who is a regular user.

```
TRANSFER OWNERSHIP OF VIEW V1
TO USER PAUL PRESERVE PRIVILEGES
```

Ownership of the view is transferred successfully, even though Paul does not hold CONTROL privilege on table JOHN.T2. Paul only needs SELECT privilege on tables JOHN.T1 and JOHN.T2 to maintain the view's existence. (The view only has SELECT privilege because Paul did not hold CONTROL privilege on both tables when the view was created and, as a result, he was not granted CONTROL on the view.) The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'PAUL'. The value in the OWNER column for the view V1 in the SYSCAT.TABDEP catalog view is replaced with 'PAUL'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'PAUL', and TABNAME = 'V1'.

*Example 5:* Assume that JOHN creates table T1, and that PUBLIC holds SELECT privilege on JOHN.T1. PAUL holds SELECT privilege on JOHN.T1 explicitly, and creates view V1 that depends on table JOHN.T1. Transfer ownership of view V1 to MIKE, who is not a DBADM, but who holds the required privileges to acquire view ownership through the special group PUBLIC.

```
TRANSFER OWNERSHIP OF VIEW V1
TO USER MIKE PRESERVE PRIVILEGES
```

Ownership of the view is transferred successfully, because Mike holds SELECT privilege on table JOHN.T1 through PUBLIC. The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'MIKE'. The value in the OWNER column for the view V1 in the SYSCAT.TABDEP catalog view is replaced with 'MIKE'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'MIKE', and TABNAME = 'V1'.

*Example 6:* Similar to example 5, assume that JOHN creates table T1, and that role R1 holds SELECT privilege on JOHN.T1. PAUL holds SELECT privilege on JOHN.T1 explicitly, and creates view V1 that depends on table JOHN.T1. Transfer ownership of view V1 to MIKE, who is not a DBADM, but who holds the required privileges through membership in role R1 to acquire view ownership.

```
TRANSFER OWNERSHIP OF VIEW V1
TO USER MIKE PRESERVE PRIVILEGES
```

Ownership of the view is transferred successfully, because Mike holds SELECT privilege on table JOHN.T1 through membership in role R1. The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'MIKE'. The value in the OWNER column for the view V1 in the SYSCAT.TABDEP catalog view is replaced with 'MIKE'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'MIKE', and TABNAME = 'V1'.

---

## Part 2. Functions



---

## Chapter 3. Functions

---

### Functions overview

A *function* is an operation that is denoted by a function name followed by a pair of parentheses enclosing the specification of arguments (there may be no arguments).

*Built-in functions* are provided with the database manager; they return a single result value, and are identified as part of the SYSIBM schema. Built-in functions include column functions (such as AVG), operator functions (such as "+"), casting functions (such as DECIMAL), and others (such as SUBSTR).

*User-defined functions* are registered to a database in SYSCAT.ROUTINES (using the CREATE FUNCTION statement). User-defined functions are never part of the SYSIBM schema. One such set of functions is provided with the database manager in a schema called SYSFUN, and another in a schema called SYSPROC.

Functions are classified as aggregate (column) functions, scalar functions, row functions, or table functions.

- The argument of an *aggregate function* is a collection of like values. An aggregate function returns a single value (possibly null), and can be specified in an SQL statement wherever an expression can be used.
- The arguments of a *scalar function* are individual scalar values, which can be of different types and have different meanings. A scalar function returns a single value (possibly null), and can be specified in an SQL statement wherever an expression can be used.
- The argument of a *row function* is a structured type. A row function returns a row of built-in data types and can only be specified as a transform function for a structured type.
- The arguments of a *table function* are individual scalar values, which can be of different types and have different meanings. A table function returns a table to the SQL statement, and can be specified only within the FROM clause of a SELECT statement.

The function name, combined with the schema, gives the fully qualified name of a function. The combination of schema, function name, and input parameters make up a *function signature*.

In some cases, the input parameter type is specified as a specific built-in data type, and in other cases, it is specified through a general variable like *any-numeric-type*. If a particular data type is specified, an exact match will only occur with the specified data type. If a general variable is used, each of the data types associated with that variable results in an exact match.

Additional functions may be available, because user-defined functions can be created in different schemas, using one of the function signatures as a source. You can also create external functions in your applications.

---

## DBPARTITIONNUM

►►—DBPARTITIONNUM—(—*column-name*—)—————►►

The schema is SYSIBM.

The DBPARTITIONNUM function returns the database partition number for a row. For example, if used in a SELECT clause, it returns the database partition number for each row in the result set.

The argument must be the qualified or unqualified name of any column in the table. Because row-level information is returned, the result is the same regardless of which column is specified. The column can have any data type.

If *column-name* references a column in a view, the expression for the column in the view must reference a column of the underlying base table, and the view must be deletable. A nested or common table expression follows the same rules as a view.

The specific row (and table) for which the database partition number is returned by the DBPARTITIONNUM function is determined from the context of the SQL statement that uses the function.

The database partition number returned on transition variables and tables is derived from the current transition values of the distribution key columns. For example, in a before insert trigger, the function returns the projected database partition number, given the current values of the new transition variables. However, the values of the distribution key columns might be modified by a subsequent before insert trigger. Thus, the final database partition number of the row when it is inserted into the database might differ from the projected value.

The data type of the result is INTEGER and is never null. If there is no db2nodes.cfg file, the result is 0.

This function cannot be used as a source function when creating a user-defined function. Because the function accepts any data type as an argument, it is not necessary to create additional signatures to support user-defined distinct types.

The DBPARTITIONNUM function cannot be used on replicated tables, within check constraints, or in the definition of generated columns (SQLSTATE 42881).

For compatibility with previous versions of DB2 products, NODENUMBER can be specified in place of DBPARTITIONNUM.

Examples:

- Count the number of instances in which the row for a given employee in the EMPLOYEE table is on a different database partition than the description of the employee's department in the DEPARTMENT table.

```
SELECT COUNT(*) FROM DEPARTMENT D, EMPLOYEE E
WHERE D.DEPTNO=E.WORKDEPT
AND DBPARTITIONNUM(E.LASTNAME) <> DBPARTITIONNUM(D.DEPTNO)
```

- Join the EMPLOYEE and DEPARTMENT tables so that the rows of the two tables are on the same database partition.

```
SELECT * FROM DEPARTMENT D, EMPLOYEE E
WHERE DBPARTITIONNUM(E.LASTNAME) = DBPARTITIONNUM(D.DEPTNO)
```



- Using a before trigger on the EMPLOYEE table, log the employee number and the projected database partition number of any new row in the EMPLOYEE table in a table named EMPINSERTLOG1.

```
CREATE TRIGGER EMPINSLOGTRIG1
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AS NEWTABLE
FOR EACH ROW
INSERT INTO EMPINSERTLOG1
VALUES (NEWTABLE.EMPNO, DBPARTITIONNUM
(NEWTABLE.EMPNO))
```

---

## DECRYPT\_BIN and DECRYPT\_CHAR

```

>> [DECRYPT_BIN] ([encrypted-data])
 [DECRYPT_CHAR] [password-string-expression] <<<

```

The schema is SYSIBM.

The DECRYPT\_BIN and DECRYPT\_CHAR functions both return a value that is the result of decrypting *encrypted-data*. The password used for decryption is either the *password-string-expression* value or the encryption password value that was assigned by the SET ENCRYPTION PASSWORD statement. To maintain the best level of security on your system, it is recommended that you do not pass the encryption password explicitly with the DECRYPT\_BIN and DECRYPT\_CHAR functions in your query; instead, use the SET ENCRYPTION PASSWORD statement to set the password, and use a host variable or dynamic parameter markers when you use the SET ENCRYPTION PASSWORD statement, rather than a literal string.

The DECRYPT\_BIN and DECRYPT\_CHAR functions can only decrypt values that are encrypted using the ENCRYPT function (SQLSTATE 428FE).

### *encrypted-data*

An expression that returns a CHAR FOR BIT DATA or VARCHAR FOR BIT DATA value as a complete, encrypted data string. The data string must have been encrypted using the ENCRYPT function.

### *password-string-expression*

An expression that returns a CHAR or VARCHAR value with at least 6 bytes and no more than 127 bytes (SQLSTATE 428FC). This expression must be the same password used to encrypt the data (SQLSTATE 428FD). If the value of the password argument is null or not provided, the data will be decrypted using the encryption password value that was assigned for the session by the SET ENCRYPTION PASSWORD statement (SQLSTATE 51039).

The result of the DECRYPT\_BIN function is VARCHAR FOR BIT DATA. The result of the DECRYPT\_CHAR function is VARCHAR. If *encrypted-data* included a hint, the hint is not returned by the function. The length attribute of the result is the length of the data type of *encrypted-data* minus 8 bytes. The actual length of the value returned by the function will match the length of the original string that was encrypted. If *encrypted-data* includes bytes beyond the encrypted string, these bytes are not returned by the function.

If the first argument can be null, the result can be null. If the first argument is null, the result is the null value.

If the data is decrypted on a different system, which uses a code page that is different from the code page in which the data was encrypted, expansion might occur when converting the decrypted value to the database code page. In such situations, the *encrypted-data* value should be cast to a VARCHAR string with a larger number of bytes.

## Examples

The following example demonstrates the use of the DECRYPT\_CHAR function by showing code fragments from an embedded SQL application.

```
EXEC SQL BEGIN DECLARE SECTION;
 char hostVarCreateTableStmt[100];
 char hostVarSetEncPassStmt[200];
 char hostVarPassword[128];
 char hostVarInsertStmt1[200];
 char hostVarInsertStmt2[200];
 char hostVarSelectStmt1[200];
 char hostVarSelectStmt2[200];
EXEC SQL END DECLARE SECTION;

/* prepare the statement */
strcpy(hostVarCreateTableStmt, "CREATE TABLE EMP (SSN VARCHAR(24) FOR BIT DATA)");
EXEC SQL PREPARE hostVarCreateTableStmt FROM :hostVarCreateTableStmt;

/* execute the statement */
EXEC SQL EXECUTE hostVarCreateTableStmt;
```

Use the SET ENCRYPTION PASSWORD statement to set an encryption password for the session:

```
/* prepare the statement with a parameter marker */
strcpy(hostVarSetEncPassStmt, "SET ENCRYPTION PASSWORD = ?");
EXEC SQL PREPARE hostVarSetEncPassStmt FROM :hostVarSetEncPassStmt;

/* execute the statement for hostVarPassword = 'Pac1f1c' */
strcpy(hostVarPassword, "Pac1f1c");
EXEC SQL EXECUTE hostVarSetEncPassStmt USING :hostVarPassword;

/* prepare the statement */
strcpy(hostVarInsertStmt1, "INSERT INTO EMP(SSN) VALUES ENCRYPT('289-46-8832')");
EXEC SQL PREPARE hostVarInsertStmt1 FROM :hostVarInsertStmt1;

/* execute the statement */
EXEC SQL EXECUTE hostVarInsertStmt1;

/* prepare the statement */
strcpy(hostVarSelectStmt1, "SELECT DECRYPT_CHAR(SSN) FROM EMP");
EXEC SQL PREPARE hostVarSelectStmt1 FROM :hostVarSelectStmt1;

/* execute the statement */
EXEC SQL EXECUTE hostVarSelectStmt1;
```

This query returns the value '289-46-8832'.

Pass the encryption password explicitly:

```
/* prepare the statement */
strcpy(hostVarInsertStmt2, "INSERT INTO EMP (SSN) VALUES ENCRYPT('289-46-8832',?)");
EXEC SQL PREPARE hostVarInsertStmt2 FROM :hostVarInsertStmt2;

/* execute the statement for hostVarPassword = 'Pac1f1c' */
strcpy(hostVarPassword, "Pac1f1c");
EXEC SQL EXECUTE hostVarInsertStmt2 USING :hostVarPassword;

/* prepare the statement */
```

```

| strcpy(hostVarSelectStmt2, "SELECT DECRYPT_CHAR(SSN,?) FROM EMP");
| EXEC SQL PREPARE hostVarSelectStmt2 FROM :hostVarSelectStmt2;
|
| /* execute the statement for hostVarPassword = 'Pacific' */
| strcpy(hostVarPassword, "Pacific");
| EXEC SQL EXECUTE hostVarSelectStmt2 USING :hostVarPassword;
|

```

This query returns the value '289-46-8832'.

---

## ENCRYPT

```

| ►►—ENCRYPT—►►
| ►—(—data-string-expression— [,—password-string-expression— [,—hint-string-expression—]])—►►

```

The schema is SYSIBM.

The ENCRYPT function returns a value that is the result of encrypting *data-string-expression*. The password used for encryption is either the *password-string-expression* value or the encryption password value that was assigned by the SET ENCRYPTION PASSWORD statement. To maintain the best level of security on your system, it is recommended that you do not pass the encryption password explicitly with the ENCRYPT function in your query; instead, use the SET ENCRYPTION PASSWORD statement to set the password, and use a host variable or dynamic parameter markers when you use the SET ENCRYPTION PASSWORD statement, rather than a literal string.

In a Unicode database, if a supplied argument is a graphic string, it is first converted to a character string before the function is executed.

### *data-string-expression*

An expression that returns a CHAR or a VARCHAR value that is to be encrypted. The length attribute for the data type of *data-string-expression* is limited to 32663 without a *hint-string-expression* argument, and 32631 when the *hint-string-expression* argument is specified (SQLSTATE 42815).

### *password-string-expression*

An expression that returns a CHAR or a VARCHAR value with at least 6 bytes and no more than 127 bytes (SQLSTATE 428FC). The value represents the password used to encrypt *data-string-expression*. If the value of the password argument is null or not provided, the data is encrypted using the encryption password value that was assigned for the session by the SET ENCRYPTION PASSWORD statement (SQLSTATE 51039).

### *hint-string-expression*

An expression that returns a CHAR or a VARCHAR value with at most 32 bytes that will help data owners remember passwords (for example, 'Ocean' as a hint to remember 'Pacific'). If a hint value is given, the hint is embedded into the result and can be retrieved using the GETHINT function. If this argument is null or not provided, no hint will be embedded in the result.

The result data type of the function is VARCHAR FOR BIT DATA.

- When the optional hint parameter is specified, the length attribute of the result is equal to the length attribute of the unencrypted data + 8 bytes + the number of bytes until the next 8-byte boundary + 32 bytes for the length of the hint.

- When the optional hint parameter is not specified, the length attribute of the result is equal to the length attribute of the unencrypted data + 8 bytes + the number of bytes until the next 8-byte boundary.

If the first argument can be null, the result can be null. If the first argument is null, the result is the null value.

Note that the encrypted result is longer than the *data-string-expression* value. Therefore, when assigning encrypted values, ensure that the target is declared with sufficient size to contain the entire encrypted value.

## Notes

- **Encryption algorithm:** The internal encryption algorithm is RC2 block cipher with padding; the 128-bit secret key is derived from the password using an MD5 message digest.

- **Encryption passwords and data:** Password management is the user's responsibility. Once the data is encrypted, only the password that was used when encrypting it can be used to decrypt it (SQLSTATE 428FD).

The encrypted result might contain null terminator and other unprintable characters. Any assignment or cast to a length that is shorter than the suggested data length might result in failed decryption in the future, and lost data. Blanks are valid encrypted data values that might be truncated when stored in a column that is too short.

- **Administration of encrypted data:** Encrypted data can only be decrypted on servers that support the decryption functions corresponding to the ENCRYPT function. Therefore, replication of columns with encrypted data should only be done to servers that support the DECRYPT\_BIN or the DECRYPT\_CHAR function.

## Examples

The following example demonstrates the use of the ENCRYPT function by showing code fragments from an embedded SQL application.

```
EXEC SQL BEGIN DECLARE SECTION;
 char hostVarCreateTableStmt[100];
 char hostVarSetEncPassStmt[200];
 char hostVarPassword[128];
 char hostVarInsertStmt1[200];
 char hostVarInsertStmt2[200];
 char hostVarInsertStmt3[200];
EXEC SQL END DECLARE SECTION;

/* prepare the statement */
strcpy(hostVarCreateTableStmt, "CREATE TABLE EMP (SSN VARCHAR(24) FOR BIT DATA)");
EXEC SQL PREPARE hostVarCreateTableStmt FROM :hostVarCreateTableStmt;

/* execute the statement */
EXEC SQL EXECUTE hostVarCreateTableStmt;
```

Use the SET ENCRYPTION PASSWORD statement to set an encryption password for the session:

```
/* prepare the statement with a parameter marker */
strcpy(hostVarSetEncPassStmt, "SET ENCRYPTION PASSWORD = ?");
EXEC SQL PREPARE hostVarSetEncPassStmt FROM :hostVarSetEncPassStmt;

/* execute the statement for hostVarPassword = 'Pac1f1c' */
strcpy(hostVarPassword, "Pac1f1c");
EXEC SQL EXECUTE hostVarSetEncPassStmt USING :hostVarPassword;
```

```

/* prepare the statement */
strcpy(hostVarInsertStmt1, "INSERT INTO EMP(SSN) VALUES ENCRYPT('289-46-8832')");
EXEC SQL PREPARE hostVarInsertStmt1 FROM :hostVarInsertStmt1;

/* execute the statement */
EXEC SQL EXECUTE hostVarInsertStmt1;

Pass the encryption password explicitly:

/* prepare the statement */
strcpy(hostVarInsertStmt2, "INSERT INTO EMP(SSN) VALUES ENCRYPT('289-46-8832',?)");
EXEC SQL PREPARE hostVarInsertStmt2 FROM :hostVarInsertStmt2;

/* execute the statement for hostVarPassword = 'Pac1f1c' */
strcpy(hostVarPassword, "Pac1f1c");
EXEC SQL EXECUTE hostVarInsertStmt2 USING :hostVarPassword;

Define a password hint:

/* prepare the statement */
strcpy(hostVarInsertStmt3, "INSERT INTO EMP(SSN) VALUES ENCRYPT('289-46-8832',?,'Ocean')");
EXEC SQL PREPARE hostVarInsertStmt3 FROM :hostVarInsertStmt3;

/* execute the statement for hostVarPassword = 'Pac1f1c' */
strcpy(hostVarPassword, "Pac1f1c");
EXEC SQL EXECUTE hostVarInsertStmt3 USING :hostVarPassword;

```

---

## GETHINT

►►—GETHINT—(—*encrypted-data*—)—————◄◄

The schema is SYSIBM.

The GETHINT function will return the password hint if one is found in the *encrypted-data*. A password hint is a phrase that will help data owners remember passwords; for example, 'Ocean' as a hint to remember 'Pacific'. In a Unicode database, if a supplied argument is a graphic string, it is first converted to a character string before the function is executed.

*encrypted-data*

An expression that returns a CHAR FOR BIT DATA or VARCHAR FOR BIT DATA value that is a complete, encrypted data string. The data string must have been encrypted using the ENCRYPT function (SQLSTATE 428FE).

The result of the function is VARCHAR(32). The result can be null; if the hint parameter was not added to the *encrypted-data* by the ENCRYPT function or the first argument is null, the result is the null value.

Example:

In this example the hint 'Ocean' is stored to help the user remember the encryption password 'Pacific'.

```

INSERT INTO EMP (SSN) VALUES ENCRYPT('289-46-8832', 'Pacific','Ocean');
SELECT GETHINT(SSN)
FROM EMP;

```

The value returned is 'Ocean'.

---

## HASHEDVALUE

►►—HASHEDVALUE—(—*column-name*—)—————◄◄

The schema is SYSIBM.

The HASHEDVALUE function returns the distribution map index of the row obtained by applying the partitioning function on the distribution key value of the row. For example, if used in a SELECT clause, it returns the distribution map index for each row of the table that was used to form the result of the SELECT statement.

The distribution map index returned on transition variables and tables is derived from the current transition values of the distribution key columns. For example, in a before insert trigger, the function will return the projected distribution map index given the current values of the new transition variables. However, the values of the distribution key columns may be modified by a subsequent before insert trigger. Thus, the final distribution map index of the row when it is inserted into the database may differ from the projected value.

The argument must be the qualified or unqualified name of a column in a table. The column can have any data type. (This function cannot be used as a source function when creating a user-defined function. Because it accepts any data type as an argument, it is not necessary to create additional signatures to support user-defined distinct types.) If *column-name* references a column of a view the expression in the view for the column must reference a column of the underlying base table and the view must be deletable. A nested or common table expression follows the same rules as a view.

The specific row (and table) for which the distribution map index is returned by the HASHEDVALUE function is determined from the context of the SQL statement that uses the function.

The data type of the result is INTEGER in the range 0 to 32767. For a table with no distribution key, the result is always 0. A null value is never returned. Since row-level information is returned, the results are the same, regardless of which column is specified for the table.

The HASHEDVALUE function cannot be used on replicated tables, within check constraints, or in the definition of generated columns (SQLSTATE 42881).

For compatibility with versions earlier than Version 8, the function name PARTITION can be substituted for HASHEDVALUE.

Example:

- List the employee numbers (EMPNO) from the EMPLOYEE table for all rows with a distribution map index of 100.

```
SELECT EMPNO FROM EMPLOYEE
WHERE HASHEDVALUE(PHONENO) = 100
```

- Log the employee number and the projected distribution map index of the new row into a table called EMPINSERTLOG2 for any insertion of employees by creating a before trigger on the table EMPLOYEE.

```

CREATE TRIGGER EMPINSLOGTRIG2
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AS NEWTABLE
FOR EACH ROW
INSERT INTO EMPINSERTLOG2
VALUES(NEWTABLE.EMPNO, HASHEDVALUE(NEWTABLE.EMPNO))

```

---

## SECLABEL

►►—SECLABEL—(—*security-policy-name*—,—*security-label-string*—)————►►

The schema is SYSIBM.

The SECLABEL function returns an unnamed security label with a data type of DB2SECURITYLABEL. Use the SECLABEL function to insert a security label with given component values without having to create a named security label.

*security-policy-name*

A string that specifies a security policy that exists at the current server (SQLSTATE 42704). The string must be a character or graphic string constant or host variable.

*security-label-string*

An expression that returns a valid representation of a security label for the security policy named by *security-policy-name* (SQLSTATE 4274I). The expression must return a value that is a built-in CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC data type.

Examples:

- The following statement inserts a row in table REGIONS which is protected by the security policy named CONTRIBUTIONS. The security label for the row to be inserted is given by the SECLABEL function. The security policy CONTRIBUTIONS has two components. The security label given has the element LIFE MEMBER for first component, the elements BLUE and YELLOW for the second component.

```

INSERT INTO REGIONS
VALUES (SECLABEL('CONTRIBUTIONS', 'LIFE MEMBER:(BLUE,YELLOW)'),
1, 'Northeast')

```

- The following statement inserts a row in table CASE\_IDS which is protected by the security policy named TS\_SECPOLICY, which has three components. The security label is provided by the SECLABEL function. The security label inserted has the element HIGH PROFILE for the first component, the empty value for the second component and the element G19 for the third component.

```

INSERT INTO CASE_IDS
VALUES (SECLABEL('TS_SECPOLICY', 'HIGH PROFILE:():G19') , 3, 'KLB')

```

---

## SECLABEL\_BY\_NAME

►►—SECLABEL\_BY\_NAME—(—*security-policy-name*—,—*security-label-name*—)————►►

The schema is SYSIBM.



The SECLABEL\_BY\_NAME function returns the specified security label. The security label returned has a data type of DB2SECURITYLABEL. Use this function to insert a named security label.

*security-policy-name*

A string that specifies a security policy that exists at the current server (SQLSTATE 42704). The string must be a character or graphic string constant or host variable.

*security-label-name*

An expression that returns the name of a security label that exists at the current server for the security policy named by *security-policy-name* (SQLSTATE 4274I). The expression must return a value that is a built-in CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC data type.

Examples:

- User Tina is trying to insert a row in table REGIONS which is protected by the security policy named CONTRIBUTIONS. Tina wants the row to be protected by the security label named EMPLOYEESECLABEL. This statement fails because CONTRIBUTIONS.EMPLOYEESECLABEL is an unknown identifier:

```
INSERT INTO REGIONS
VALUES (CONTRIBUTIONS.EMPLOYEESECLABEL, 1, 'Southwest') -- incorrect
```

This statement fails because the first value is a string, it does not have a data type of DB2SECURITYLABEL:

```
INSERT INTO REGIONS
VALUES ('CONTRIBUTIONS.EMPLOYEESECLABEL', 1, 'Southwest') -- incorrect
```

This statement succeeds because the SECLABEL\_BY\_NAME function returns a security label that has a data type of DB2SECURITYLABEL:

```
INSERT INTO REGIONS
VALUES (SECLABEL_BY_NAME('CONTRIBUTIONS', 'EMPLOYEESECLABEL'),
1, 'Southwest') -- correct
```

---

## SECLABEL\_TO\_CHAR

►►—SECLABEL\_TO\_CHAR—(—*security-policy-name*—,—*security-label*—)—————►

The schema is SYSIBM.

The SECLABEL\_TO\_CHAR function accepts a security label and returns a string that contains all elements in the security label. The string is in the security label string format.

*security-policy-name*

A string that specifies a security policy that exists at the current server (SQLSTATE 42704). The string must be a character or graphic string constant or host variable.

*security-label*

An expression that returns a security label value that is valid for the security policy named by *security-policy-name* (SQLSTATE 4274I). The expression must return a value that is a built-in SYSPROC.DB2SECURITYLABEL distinct type.

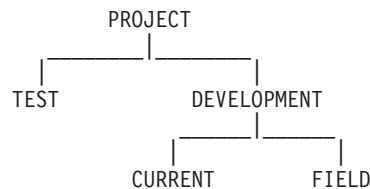
## Notes

- If the authorization ID of the statement executes this function on a security label being read from a column with a data type of DB2SECURITYLABEL then that authorization ID's LBAC credentials might affect the output of the function. In such a case an element is not included in the output if the authorization ID does not have read access to that element. An authorization ID has read access to an element if its LBAC credentials would allow it to read data that was protected by a security label containing only that element, and no others.

For the rule set DB2LBACRULES only components of type TREE can contain elements that you do not have read access to. For other types of component, if any one of the elements block read access then you will not be able to read the row at all. So only components of type tree will have elements excluded in this way.

Example:

- The EMP table has two columns, RECORDNUM and LABEL; RECORDNUM has data type INTEGER, and LABEL has type DB2SECURITYLABEL. Table EMP is protected by security policy DATA\_ACCESSPOLICY, which uses the DB2LBACRULES rule set and has only one component (GROUPS, of type TREE). GROUPS has five elements: PROJECT, TEST, DEVELOPMENT, CURRENT, AND FIELD. The following diagram shows the relationship of these elements to one another:



The EMP table contains the following data:

| RECORDNUM | LABEL            |
|-----------|------------------|
| 1         | PROJECT          |
| 2         | (TEST, FIELD)    |
| 3         | (CURRENT, FIELD) |

The user whose ID is Djavan holds a security label for reading that contains only the DEVELOPMENT element. This means that Djavan has read access to the DEVELOPMENT, CURRENT, and FIELD elements:

```
SELECT RECORDNUM, SECLABEL_TO_CHAR('DATA_ACCESSPOLICY', LABEL) FROM EMP
```

returns:

| RECORDNUM | LABEL            |
|-----------|------------------|
| 2         | FIELD            |
| 3         | (CURRENT, FIELD) |

The row with a RECORDNUM value of 1 is not included in the output, because Djavan's LBAC credentials do not allow him to read that row. In the row with a RECORDNUM value of 2, element TEST is not included in the output, because Djavan does not have read access to that element; Djavan would not have been able to access the row at all if TEST were the only element in the security label. Because Djavan has read access to elements CURRENT and FIELD, both elements appear in the output.

Now Djavan is granted an exemption to the DB2LBACREADTREE rule. This means that no element of a TREE type component will block read access. The same query returns:

| RECORDNUM | LABEL            |
|-----------|------------------|
| 1         | PROJECT          |
| 2         | (TEST, FIELD)    |
| 3         | (CURRENT, FIELD) |

This time the output includes all rows and all elements, because the exemption gives Djavan read access to all of the elements.

---

## TABLE\_NAME

►►—TABLE\_NAME—(—*object-name*—, —*object-schema*—)

The schema is SYSIBM.

The TABLE\_NAME function returns an unqualified name of the object found after any alias chains have been resolved. The specified *object-name* (and *object-schema*) are used as the starting point of the resolution. If the starting point does not refer to an alias, the unqualified name of the starting point is returned. The resulting name may be of a table, view, or undefined object. In a Unicode database, if a supplied argument is a graphic string, it is first converted to a character string before the function is executed.

### *object-name*

A character expression representing the unqualified name (usually of an existing alias) to be resolved. *object-name* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 bytes.

### *object-schema*

A character expression representing the schema used to qualify the supplied *object-name* value before resolution. *object-schema* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 bytes.

If *object-schema* is not supplied, the default schema is used for the qualifier.

The data type of the result of the function is VARCHAR(128). If *object-name* can be null, the result can be null; if *object-name* is null, the result is the null value. If *object-schema* is the null value, the default schema name is used. The result is the character string representing an unqualified name. The result name could represent one of the following:

**table** The value for *object-name* was either a table name (the input value is returned) or an alias name that resolved to the table whose name is returned.

**view** The value for *object-name* was either a view name (the input value is returned) or an alias name that resolved to the view whose name is returned.

### **undefined object**

The value for *object-name* was either an undefined object (the input value is returned) or an alias name that resolved to the undefined object whose name is returned.

Therefore, if a non-null value is given to this function, a value is always returned, even if no object with the result name exists.

**Note:** To improve performance in partitioned database configurations by avoiding the unnecessary communication that occurs between the coordinator partition and catalog partition when using the TABLE\_SCHEMA and TABLE\_NAME scalar functions, the BASE\_TABLE table function can be used instead.

---

## TABLE\_SCHEMA

►►—TABLE\_SCHEMA—(—*object-name*—, —*object-schema*—)

The schema is SYSIBM.

The TABLE\_SCHEMA function returns the schema name of the object found after any alias chains have been resolved. The specified *object-name* (and *object-schema*) are used as the starting point of the resolution. If the starting point does not refer to an alias, the schema name of the starting point is returned. The resulting schema name may be of a table, view, or undefined object. In a Unicode database, if a supplied argument is a graphic string, it is first converted to a character string before the function is executed.

### *object-name*

A character expression representing the unqualified name (usually of an existing alias) to be resolved. *object-name* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 bytes.

### *object-schema*

A character expression representing the schema used to qualify the supplied *object-name* value before resolution. *object-schema* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 bytes.

If *object-schema* is not supplied, the default schema is used for the qualifier.

The data type of the result of the function is VARCHAR(128). If *object-name* can be null, the result can be null; if *object-name* is null, the result is the null value. If *object-schema* is the null value, the default schema name is used. The result is the character string representing a schema name. The result schema could represent the schema name for one of the following:

**table** The value for *object-name* was either a table name (the input or default value of *object-schema* is returned) or an alias name that resolved to a table for which the schema name is returned.

**view** The value for *object-name* was either a view name (the input or default value of *object-schema* is returned) or an alias name that resolved to a view for which the schema name is returned.

### **undefined object**

The value for *object-name* was either an undefined object (the input or default value of *object-schema* is returned) or an alias name that resolved to an undefined object for which the schema name is returned.

Therefore, if a non-null *object-name* value is given to this function, a value is always returned, even if the object name with the result schema name does not exist. For example, TABLE\_SCHEMA('DEPT', 'PEOPLE') returns 'PEOPLE' if the catalog entry is not found.

**Note:** To improve performance in partitioned database configurations by avoiding the unnecessary communication that occurs between the coordinator partition and catalog partition when using the TABLE\_SCHEMA and TABLE\_NAME scalar functions, the BASE\_TABLE table function can be used instead.

Examples:

- PBIRD tries to select the statistics for a given table from SYSCAT.TABLES using an alias PBIRD.A1 defined on the table HEDGES.T1.

```
SELECT NPAGES, CARD FROM SYSCAT.TABLES
WHERE TABNAME = TABLE_NAME ('A1')
AND TABSCHEMA = TABLE_SCHEMA ('A1')
```

The requested statistics for HEDGES.T1 are retrieved from the catalog.

- Select the statistics for an object called HEDGES.X1 from SYSCAT.TABLES using HEDGES.X1. Use TABLE\_NAME and TABLE\_SCHEMA since it is not known whether HEDGES.X1 is an alias or a table.

```
SELECT NPAGES, CARD FROM SYSCAT.TABLES
WHERE TABNAME = TABLE_NAME ('X1','HEDGES')
AND TABSCHEMA = TABLE_SCHEMA ('X1','HEDGES')
```

Assuming that HEDGES.X1 is a table, the requested statistics for HEDGES.X1 are retrieved from the catalog.

- Select the statistics for a given table from SYSCAT.TABLES using an alias PBIRD.A2 defined on HEDGES.T2 where HEDGES.T2 does not exist.

```
SELECT NPAGES, CARD FROM SYSCAT.TABLES
WHERE TABNAME = TABLE_NAME ('A2','PBIRD')
AND TABSCHEMA = TABLE_SCHEMA ('A2','PBIRD')
```

The statement returns 0 records as no matching entry is found in SYSCAT.TABLES where TABNAME = 'T2' and TABSCHEMA = 'HEDGES'.

- Select the qualified name of each entry in SYSCAT.TABLES along with the final referenced name for any alias entry.

```
SELECT TABSCHEMA AS SCHEMA, TABNAME AS NAME,
TABLE_SCHEMA (BASE_TABNAME, BASE_TABSCHEMA) AS REAL_SCHEMA,
TABLE_NAME (BASE_TABNAME, BASE_TABSCHEMA) AS REAL_NAME
FROM SYSCAT.TABLES
```

The statement returns the qualified name for each object in the catalog and the final referenced name (after alias has been resolved) for any alias entries. For all non-alias entries, BASE\_TABNAME and BASE\_TABSCHEMA are null so the REAL\_SCHEMA and REAL\_NAME columns will contain nulls.

---

## Part 3. Applications





---

## Chapter 4. Application considerations

---

### About SQL statements

#### How SQL statements are invoked

SQL statements are classified as executable or non-executable.

An *executable statement* can be invoked in four ways. It can be:

- Embedded in an application program
- Embedded in an SQL procedure.
- Prepared and executed dynamically
- Issued interactively

Depending on the statement, some or all of these methods can be used. (Statements embedded in REXX are prepared and executed dynamically.)

A *non-executable statement* can only be embedded in an application program.

Another SQL statement construct is the select-statement. A *select-statement* can be invoked in three ways. It can be:

- Included in DECLARE CURSOR, and executed implicitly by OPEN, FETCH and CLOSE (static invocation)
- Prepared dynamically, referenced in DECLARE CURSOR, and executed implicitly by OPEN, FETCH and CLOSE (dynamic invocation)
- Issued interactively

#### Embedding a statement in an application program

SQL statements can be included in a source program that will be submitted to a precompiler. Such statements are said to be *embedded* in the program. An embedded statement can be placed anywhere in the program where a host language statement is allowed. Each embedded statement must be preceded by the keywords EXEC SQL.

An executable statement embedded in an application program is executed every time a statement of the host language would be executed if it were specified in the same place. Thus, a statement within a loop is executed every time the loop is executed, and a statement within a conditional construct is executed only when the condition is satisfied.

An embedded statement can contain references to host variables. A host variable referenced in this way can be used in two ways. It can be used:

- As input (the current value of the host variable is used in the execution of the statement)
- As output (the variable is assigned a new value as a result of executing the statement)

In particular, all references to host variables in expressions and predicates are effectively replaced by current values of the variables; that is, the variables are used as input.

All executable statements should be followed by a test of the SQL return code. Alternatively, the WHENEVER statement (which is itself non-executable) can be used to change the flow of control immediately after the execution of an embedded statement.

All objects referenced in data manipulation language (DML) statements must exist when the statements are bound to a database.

An embedded non-executable statement is processed only by the precompiler. The precompiler reports any errors encountered in the statement. The statement is *never* processed during program execution; therefore, such statements should not be followed by a test of the SQL return code.

Statements can be included in the SQL-procedure-body portion of the CREATE PROCEDURE statement. Such statements are said to be embedded in the SQL procedure. Whenever an SQL statement description refers to a *host-variable*, an *SQL-variable* can be used if the statement is embedded in an SQL procedure.

## **Dynamic preparation and execution**

An application program can dynamically build an SQL statement in the form of a character string placed in a host variable. In general, the statement is built from some data available to the program (for example, input from a workstation). The statement (not a select-statement) constructed can be prepared for execution by means of the (embedded) PREPARE statement, and executed by means of the (embedded) EXECUTE statement. Alternatively, an (embedded) EXECUTE IMMEDIATE statement can be used to prepare and execute the statement in one step.

A statement that is going to be dynamically prepared must not contain references to host variables. It can instead contain parameter markers. (For rules concerning parameter markers, see "PREPARE".) When the prepared statement is executed, the parameter markers are effectively replaced by current values of the host variables specified in the EXECUTE statement. Once prepared, a statement can be executed several times with different values for the host variables. Parameter markers are not allowed in the EXECUTE IMMEDIATE statement.

Successful or unsuccessful execution of the statement is indicated by the setting of an SQL return code in the SQLCA after the EXECUTE (or EXECUTE IMMEDIATE) statement completes. The SQL return code should be checked, as described above. For more information, see "SQL return codes (SQLCODE and SQLSTATE)" on page 493.

## **Static invocation of a select-statement**

A select-statement can be included as a part of the (non-executable) DECLARE CURSOR statement. Such a statement is executed every time the cursor is opened by means of the (embedded) OPEN statement. After the cursor is open, the result table can be retrieved, one row at a time, by successive executions of the FETCH statement.

Used in this way, the select-statement can contain references to host variables. These references are effectively replaced by the values that the variables have when the OPEN statement executes.

### **Dynamic invocation of a select-statement**

An application program can dynamically build a select-statement in the form of a character string placed in a host variable. In general, the statement is built from some data available to the program (for example, a query obtained from a workstation). The statement so constructed can be prepared for execution by means of the (embedded) PREPARE statement, and referenced by a (non-executable) DECLARE CURSOR statement. The statement is then executed every time the cursor is opened by means of the (embedded) OPEN statement. After the cursor is open, the result table can be retrieved, one row at a time, by successive executions of the FETCH statement.

Used in this way, the select-statement must not contain references to host variables. It can contain parameter markers instead. The parameter markers are effectively replaced by the values of the host variables specified in the OPEN statement.

### **Interactive invocation**

A capability for entering SQL statements from a workstation is part of the architecture of the database manager. A statement entered in this way is said to be issued interactively. Such a statement must be an executable statement that does not contain parameter markers or references to host variables, because these make sense only in the context of an application program.

### **SQL use with other host systems**

SQL statement syntax exhibits minor variations among different types of host systems (DB2 for z/OS, DB2 for System i, DB2 Database for Linux, UNIX, and Windows). Regardless of whether the SQL statements in an application are static or dynamic, it is important — if the application is meant to access different database host systems — to ensure that the SQL statements and precompile/bind options are supported on the database systems that the application will access.

Further information about SQL statements used in other host systems can be found in the *DB2 for System i SQL Reference* and the *DB2 for z/OS SQL Reference*.

### **SQL return codes (SQLCODE and SQLSTATE)**

An application program containing executable SQL statements can use either SQLCODE or SQLSTATE values to handle return codes from SQL statements. There are two ways in which an application can get access to these values.

- Include a structure named SQLCA. The SQLCA includes an integer variable named SQLCODE and a character string variable named SQLSTATE. In REXX, an SQLCA is provided automatically. In other languages, an SQLCA can be obtained by using the INCLUDE SQLCA statement.
- If LANGLEVEL SQL92E is specified as a precompile option, a variable named SQLCODE or SQLSTATE can be declared in the SQL declare section of the program. If neither of these variables is declared in the SQL declare section, it is assumed that a variable named SQLCODE is declared elsewhere in the program. With LANGLEVEL SQL92E, the program should not have an INCLUDE SQLCA statement.

An SQLCODE is set by the database manager after each SQL statement executes. All database managers conform to the ISO/ANSI SQL standard, as follows:

- If SQLCODE = 0 and SQLWARN0 is blank, execution was successful.
- If SQLCODE = 100, “no data” was found. For example, a FETCH statement returned no data, because the cursor was positioned after the last row of the result table.
- If SQLCODE > 0 and not = 100, execution was successful with a warning.
- If SQLCODE = 0 and SQLWARN0 = 'W', execution was successful, but one or more warning indicators were set.
- If SQLCODE < 0, execution was not successful.

The meaning of SQLCODE values other than 0 and 100 is product-specific.

An SQLSTATE is set by the database manager after each SQL statement executes. Application programs can check the execution of SQL statements by testing SQLSTATE instead of SQLCODE. SQLSTATE provides common codes for common error conditions. Application programs can test for specific errors or classes of errors. The coding scheme is the same for all IBM database managers, and is based on the ISO/ANSI SQL92 standard.

## SQL comments

Static SQL statements can include host language or SQL comments. Dynamic SQL statements can include SQL comments. There are two types of SQL comments:

### simple comments

Simple comments are introduced by two consecutive hyphens (--) and end with the end of line.

### bracketed comments

Bracketed comments are introduced by /\* and end with \*/.

The following rules apply to the use of simple comments:

- The two hyphens must be on the same line and must not be separated by a space.
- Simple comments can be started wherever a space is valid (except within a delimiter token or between 'EXEC' and 'SQL').
- Simple comments cannot be continued to the next line.
- In COBOL, the hyphens must be preceded by a space.

The following rules apply to the use of bracketed comments:

- The /\* must be on the same line and must not be separated by a space.
- The \*/ must be on the same line and must not be separated by a space.
- Bracketed comments can be started wherever a space is valid (except within a delimiter token or between 'EXEC' and 'SQL').
- Bracketed comments can be continued to subsequent lines.

*Example 1:* This example shows how to include simple comments in a statement:

```
CREATE VIEW PRJ_MAXPER -- PROJECTS WITH MOST SUPPORT PERSONNEL
AS SELECT PROJNO, PROJNAME -- NUMBER AND NAME OF PROJECT
FROM PROJECT
WHERE DEPTNO = 'E21' -- SYSTEMS SUPPORT DEPT CODE
AND PRSTAFF > 1
```

*Example 2:* This example shows how to include bracketed comments in a statement:

```
CREATE VIEW PRJ_MAXPER /* PROJECTS WITH MOST SUPPORT
 PERSONNEL */
AS SELECT PROJNO, PROJNAME /* NUMBER AND NAME OF PROJECT */
FROM PROJECT
WHERE DEPTNO = 'E21' /* SYSTEMS SUPPORT DEPT CODE */
AND PRSTAFF > 1
```

## About SQL control statements

SQL control statements, also called SQL Procedural Language (SQL PL), are SQL statements that allow SQL to be used in a manner similar to writing a program in a structured programming language. SQL control statements provide the capability to control the logic flow, declare, and set variables, and handle warnings and exceptions. Some SQL control statements include other nested SQL statements. SQL control statements can be used in the body of a routine, trigger or a compound statement.

### References to SQL parameters, SQL variables, and global variables

SQL parameters, SQL variables, and global variables can be referenced anywhere in an SQL procedure statement where an expression or variable can be specified. Host variables cannot be specified in SQL routines, SQL triggers or dynamic compound statements. SQL parameters can be referenced anywhere in the routine, and can be qualified with the routine name. SQL variables can be referenced anywhere in the compound statement in which they are declared, and can be qualified with the label name specified at the beginning of the compound statement. If an SQL parameter or SQL variable has a row data type, fields can be referenced anywhere an SQL parameter or SQL variable can be referenced. Global variables can be referenced within any expression as long as the expression is not required to be deterministic. The following scenarios require deterministic expressions, which preclude the use of global variables:

- Check constraints
- Definitions of generated columns
- Refresh immediate MQTs

All SQL parameters, SQL variables, row variable fields, and global variables are considered nullable. The name of an SQL parameter, SQL variable, row variable field, or global variable in an SQL routine can be the same as the name of a column in a table or view referenced in the routine. The name of an SQL variable or row variable field can also be the same as the name of another SQL variable or row variable field declared in the same routine. This can occur when the two SQL variables are declared in different compound statements. The compound statement that contains the declaration of an SQL variable determines the scope of that variable. For more information, see “Compound SQL (Procedure)”.

The name of an SQL variable or SQL parameter in an SQL routine can be the same as the name of an identifier used in certain SQL statements. If the name is not qualified, the following rules describe whether the name refers to the identifier or to the SQL parameter or SQL variable:

- In the SET PATH and SET SCHEMA statements, the name is checked as an SQL parameter or SQL variable. If not found as an SQL variable or SQL parameter, it is used as an identifier.
- In the CONNECT, DISCONNECT, RELEASE, and SET CONNECTION statements, the name is used as an identifier.

Names that are the same should be explicitly qualified. Qualifying a name clearly indicates whether the name refers to a column, SQL variable, SQL parameter, row variable field, or global variable. If the name is not qualified, or qualified but still ambiguous, the following rules describe whether the name refers to a column, an SQL variable, an SQL parameter, or a global variable:

- If the tables and views specified in an SQL routine body exist at the time the routine is created, the name is first checked as a column name. If not found as a column, it is then checked as an SQL variable in the compound statement, then checked as an SQL parameter, and then, finally, checked as a global variable.
- If the referenced tables or views do not exist at the time the routine is created, the name is first checked as an SQL variable in the compound statement, then as an SQL parameter, and then as a global variable. The variable can be declared within the compound statement that contains the reference, or within a compound statement in which that compound statement is nested. If two SQL variables are within the same scope and have the same name, which can happen if they are declared in different compound statements, the SQL variable that is declared in the innermost compound statement is used. If not found, it is assumed to be a column.

## References to labels

Labels can be specified on most SQL procedure statements. The compound statement that contains the statement that defines a label determines the scope of that label name. A label name must be unique within the compound statement in which it is defined, including any labels defined in compound statements that are nested within that compound statement (SQLSTATE 42734). The label must not be the same as a label specified on the compound statement itself (SQLSTATE 42734), or the same as the name of the routine that contains the SQL procedure statement (SQLSTATE 42734).

A label name can only be referenced within the compound statement in which it is defined, including any compound statements that are nested within that compound statement. A label can be used to qualify the name of an SQL variable, or it can be specified as the target of a GOTO, LEAVE, or ITERATE statement.

## References to SQL condition names

The name of an SQL condition can be the same as the name of another SQL condition declared in the same routine. This can occur when the two SQL conditions are declared in different compound statements. The compound statement that contains the declaration of an SQL condition name determines the scope of that condition name. A condition name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A condition name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement. When there is a reference to a condition name, the condition that is declared in the innermost compound statement is the condition that is used. For more information, see “Compound SQL (Procedure)”.

## References to SQL statement names

The name of an SQL statement can be the same as the name of another SQL statement declared in the same routine. This can occur when the two SQL statements are declared in different compound statements. The compound



statement that contains the declaration of an SQL statement name determines the scope of that statement name. A statement name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A statement name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement. When there is a reference to a statement name, the statement that is declared in the innermost compound statement is the statement that is used. For more information, see “Compound SQL (Procedure)”.

## References to SQL cursor names

Cursor names include the names of declared cursors and the names of cursor variables.

The name of an SQL cursor can be the same as the name of another SQL cursor declared in the same routine. This can occur when the two SQL cursors are declared in different compound statements.

The compound statement that contains the declaration of an SQL cursor determines the scope of that cursor name. A cursor name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A cursor name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement. When there is a reference to a cursor name, the cursor that is declared in the innermost compound statement is the cursor that is used. For more information, see “Compound SQL (Procedure)”.

If the cursor constructor assigned to a cursor variable contains a reference to a local SQL variable, then any OPEN statement that uses the cursor variable must be within the scope where the local SQL variable was declared.

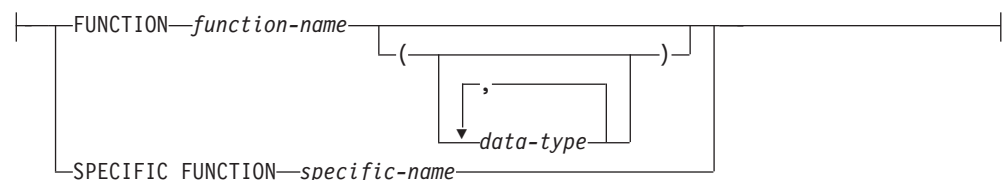
## Function, method, and procedure designators

The following sections describe syntax fragments that are used to uniquely identify a function, method, or procedure that is not defined in a module.

### Function designator

A function designator uniquely identifies a single function. Function designators typically appear in DDL statements for functions (such as DROP or ALTER). A function designator must not identify a module function (SQLSTATE 42883).

#### function-designator:



#### FUNCTION *function-name*

Identifies a particular function, and is valid only if there is exactly one function instance with the name *function-name* in the schema. The identified function

can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one instance of the function in the named or implied schema, an error (SQLSTATE 42725) is raised.

**FUNCTION** *function-name* (*data-type*,...)

Provides the function signature, which uniquely identifies the function. The function resolution algorithm is not used.

*function-name*

Specifies the name of the function. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(*data-type*,...)

Values must match the data types that were specified (in the corresponding position) on the CREATE FUNCTION statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific function instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because  $0 < n < 25$  means REAL, and  $24 < n < 54$  means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no function with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

**SPECIFIC FUNCTION** *specific-name*

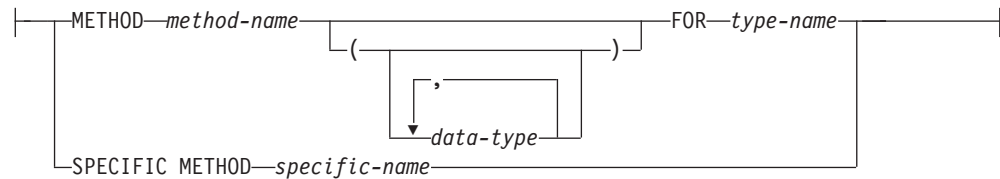
Identifies a particular user-defined function, using the name that is specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

## Method designator

A method designator uniquely identifies a single method. Method designators typically appear in DDL statements for methods (such as DROP or ALTER).



## method-designator:



### **METHOD** *method-name*

Identifies a particular method, and is valid only if there is exactly one method instance with the name *method-name* for the type *type-name*. The identified method can have any number of parameters defined for it. If no method by this name exists for the type, an error (SQLSTATE 42704) is raised. If there is more than one instance of the method for the type, an error (SQLSTATE 42725) is raised.

### **METHOD** *method-name (data-type,...)*

Provides the method signature, which uniquely identifies the method. The method resolution algorithm is not used.

#### *method-name*

Specifies the name of the method for the type *type-name*.

#### *(data-type,...)*

Values must match the data types that were specified (in the corresponding position) on the CREATE TYPE statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific method instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because  $0 < n < 25$  means REAL, and  $24 < n < 54$  means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no method with the specified signature exists for the type in the named or implied schema, an error (SQLSTATE 42883) is raised.

### **FOR** *type-name*

Names the type with which the specified method is to be associated. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

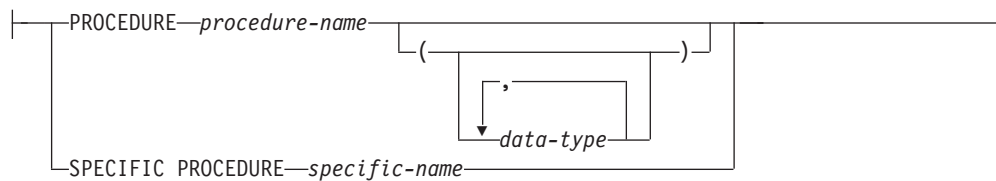
### SPECIFIC METHOD *specific-name*

Identifies a particular method, using the name that is specified or defaulted to at method creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific method instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

## Procedure designator

A procedure designator uniquely identifies a single procedure. Procedure designators typically appear in DDL statements for procedures (such as DROP or ALTER). A procedure designator must not identify a module procedure (SQLSTATE 42883).

### procedure-designator:



### PROCEDURE *procedure-name*

Identifies a particular procedure, and is valid only if there is exactly one procedure instance with the name *procedure-name* in the schema. The identified procedure can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no procedure by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one instance of the procedure in the named or implied schema, an error (SQLSTATE 42725) is raised.

### PROCEDURE *procedure-name* (*data-type*,...)

Provides the procedure signature, which uniquely identifies the procedure. The procedure resolution algorithm is not used.

#### *procedure-name*

Specifies the name of the procedure. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

#### (*data-type*,...)

Values must match the data types that were specified (in the corresponding position) on the CREATE PROCEDURE statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific procedure instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because  $0 < n < 25$  means REAL, and  $24 < n < 54$  means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

#### **SPECIFIC PROCEDURE** *specific-name*

Identifies a particular procedure, using the name that is specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

---

## **Database connection management via embedded SQL applications**

### **Connecting to DB2 databases in embedded SQL applications**

Before working with a database, you are required to establish a connection to that database. Embedded SQL provides multiple ways in which to include code for establishing database connections. Depending on the embedded SQL host programming language there might be one or more way of doing this.

Database connections can be established implicitly or explicitly. An implicit connection is a connection where the user ID is presumed to be the current user ID. This type of connection is not recommended for database applications. Explicit database connections, which require that a user ID and password be specified, are strongly recommended.

### **Connecting to DB2 databases in C and C++ Embedded SQL applications**

When working with C and C++ applications, a database connection can be established by executing the following statement.

```
EXEC SQL CONNECT TO sample;
```

If you want to use a specific user id (herrick) and password (mypassword), use the following statement:

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword;
```

## Connecting to DB2 databases in COBOL Embedded SQL applications

When working with COBOL applications, a database connection is established by executing the following statement. This statement creates a connection to the sample database using the default user name.

```
EXEC SQL CONNECT TO sample END-EXEC.
```

If you want to use a specific user id (herrick) and password (mypassword), use the following statement:

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword END-EXEC.
```

## Connecting to DB2 databases in FORTRAN Embedded SQL applications

When working with FORTRAN applications, a database connection is established by executing the following statement. This statement creates a connection to the sample database using the default user name.

```
EXEC SQL CONNECT TO sample
```

If you want to use a specific user id (herrick) and password (mypassword), use the following statement:

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword
```

## Connecting to DB2 databases in REXX Embedded SQL applications

When working with REXX applications, a database connection is established by executing the following statement. This statement creates a connection to the sample database using the default user name.

```
CALL SQLEXEC 'CONNECT TO sample'
```

If you want to use a specific user id (herrick) and password (mypassword), use the following statement:

```
CALL SQLEXEC 'CONNECT TO sample USER herrick USING mypassword'
```

## Disconnecting from embedded SQL applications

The disconnect statement is the final step in working with a database. This topic will provide examples of the disconnect statement in the supported host languages.

### Disconnecting from DB2 databases in C and C++ Embedded SQL applications

When working with C and C++ applications, a database connection is closed by issuing the following statement:

```
EXEC SQL CONNECT RESET;
```

### Disconnecting from DB2 databases in COBOL Embedded SQL applications

When working with COBOL applications, a database connection is closed by issuing the following statement:

```
EXEC SQL CONNECT RESET END-EXEC.
```

## Disconnecting from DB2 databases in REXX Embedded SQL applications

When working with REXX applications, a database connection is closed by issuing the following statement:

```
CALL SQLEXEC 'CONNECT RESET'
```

When working with FORTRAN applications, a database connection is closed by issuing the following statement:

```
EXEC SQL CONNECT RESET
```

---

## Considerations for routines

### Security of routines

The security of routines is paramount to ensure their continued functioning, to minimize the risk of tampering, and to protect the database system environment. There are a few categories of routine security considerations each with varying levels of risk. One must be aware of these risks when developing or maintaining routines so as to mitigate unfortunate outcomes as much as possible.

### Security control of who can create routines

The security of routines begins when users are given the necessary privileges to execute the CREATE statement required to create routines in the database. When granting these privileges, it is important to understand the corresponding risks:

- Users with the privilege to execute the CREATE statement for a routine can create multiple routines.
- Users with the privilege to execute the CREATE statement for a routine can create routines that can modify the database layout or database data subject to the other privileges that user has.
- Users that successfully create routines are automatically granted the EXECUTE privilege required to invoke the routine.
- Users that successfully create routines are automatically granted the ALTER ROUTINE privilege required to modify the routine.

To minimize the risk of users modifying the database and data:

- Minimize the number of users that have the privilege to create routines.
- Ensure that the user IDs of departed employees are removed, or if they are re-used, be sure to assess the procedure related privileges.

Refer to the topics on controlling access to database objects and data for more on how to grant and revoke privileges from one, many, or all database users.

### Security control of who can invoke routines

It is easy to determine when users require privileges: they are unable to do something. It is harder to determine when users no longer require these privileges. This is particularly true when it comes to users with privileges to invoke routines, as allowing them to retain their privileges can introduce risks:

- Users that have been granted the EXECUTE privilege to invoke a routine will continue to be able to invoke the routine until this privilege is removed. If the routine contains sensitive logic or acts on sensitive data this can be a business risk.

To minimize the risk of users modifying the database and data:

- Minimize the number of users that have the privilege to invoke routines.
- Ensure that the user IDs of departed employees are removed, or if they are re-used, be sure to assess the procedure related privileges.
- If you suspect that someone is maliciously invoking routines, you should revoke the EXECUTE privilege for each of those routines.

## Security control of routines defined with FENCED or NOT FENCED clauses

When formulating the CREATE statement for a routine, you must determine whether you want to specify the FENCED clause or NOT FENCED clause. Once you understand the benefits of creating a routine as fenced or unfenced it is important to assess the risks associated with running routines with external implementations as NOT FENCED.

- Routines created with the NOT FENCED clause can accidentally or maliciously corrupt the database manager's shared memory, damage the database control structures, or access database manager resources which can cause the database manager to fail. There is also the risk that they will corrupt databases and their tables.

To ensure the integrity of the database manager and its databases:

- Thoroughly screen routines you intend to create that specify the NOT FENCED clause. These routines must be fully tested, debugged, and not exhibit any unexpected side-effects. In the examination of the routine code, pay close attention to memory management and the use of static variables. The greatest potential for corruption arises when code does not properly manage memory or incorrectly uses static variables. These problems are prevalent in languages other than Java(TM) and .NET programming languages.

In order to register a NOT FENCED routine, the CREATE\_NOT\_FENCED\_ROUTINE authority is required. When granting the CREATE\_NOT\_FENCED\_ROUTINE authority, be aware that the recipient can potentially gain unrestricted access to the database manager and all its resources.

**Note:** NOT FENCED routines are not supported in Common Criteria compliant configurations.

## Securing routines

When creating routines it is important to ensure that the routines, routine libraries (in the case of external routines), and the privileges of the users that will interact with the routines are managed with routine security in mind.

Although it might not be necessary to have anything as elaborate as a routine security strategy, it helps to be mindful of the factors contributing to the security of routines and to follow a disciplined approach when securing routines.

### Prerequisites

- Read the topic, "Security of routines".
- To fully secure routines within the database system you must have:
  - Root user access on the database server operating system.
  - One of the SECADM or ACCESSCTRL authorities.

Whether you are creating a routine, or assessing an existing routine, the procedure for securing a routine is similar.

1. Limit the number of user IDs with the privileges required to create routines and ensure that these users are allowed to have these privileges.
  - Upon successful execution of the CREATE statement for a routine, this user ID will automatically be granted other privileges including the EXECUTE privilege, which allows the user to invoke the routine, and the GRANT EXECUTE privilege, which allows the user to grant the ability to invoke the routine to other users.
  - Ensure that the users with this privilege are few and that the right users get this privilege.
2. Assess the routine for potentially malicious or inadequately reviewed or tested code.
  - Consider the origin of the routine. Is the party that supplied the routine reliable?
  - Look for malicious code such as code that attempts to read or write to the database server file system and or replace files there.
  - Look for poorly implemented code related to memory management, pointer manipulation, and the use of static variables that might cause the routine to fail.
  - Verify that the code has been adequately tested.
3. Reject routines that appear to be excessively unsafe or poorly coded - the risk is not always worth it.
4. Contain the risks associated with only somewhat potentially risky routines.
  - SQL user-defined SQL routines are by default created as NOT FENCED THREADSAFE routines, because they are safe to run within the database manager memory space. For these routines you do not need to do anything.
  - Specify the FENCED clause in the CREATE statement for the routine. This will ensure that the routine operation does not affect the database manager. This is a default clause.
  - If the routine is multi-threaded, specify the NOT THREADSAFE clause in the CREATE statement for the routine. This will ensure that any failures or malicious code in the routine do not impact other routines that might run in a shared thread process.
5. If the routine is an external routine, you must put the routine implementation library or class file on the database server. Follow the general recommendations for deploying routines and the specific recommendations for deploying external routine library or class files.

---

## Guidelines for stored procedures

Stored procedures permit one call to a remote database to execute a preprogrammed procedure in a database application environment in which many situations are repetitive. For example, for receiving a fixed set of data, performing the same set of multiple requests against a database, or returning a fixed set of data might represent several accesses to the database.

Processing a single SQL statement for a remote database requires sending two transmissions: one request and one receive. Because an application contains many SQL statements it requires many transmissions to complete its work.



However, when a IBM data server client uses a stored procedure that encapsulates many SQL statements, it requires only two transmissions for the entire process.

Stored procedures usually run in processes separate from the database agents. This separation requires the stored procedure and agent processes to communicate through a router. However, a special kind of stored procedure that runs in the agent process might improve performance, although it carries significant risks of corrupting data and databases.

These risky stored procedures are those created as *not fenced*. For a not-fenced stored procedure, nothing separates the stored procedure from the database control structures that the database agent uses. If a DBA wants to ensure that the stored procedure operations will not accidentally or maliciously damage the database control structures, the *not fenced* option is omitted.

Because of the risk of damaging your database, use *not fenced* stored procedures **only** when you need the maximum possible performance benefits. In addition, make absolutely sure that the procedure is well coded and has been thoroughly tested before allowing it to run as a not-fenced stored procedure. If a fatal error occurs while running a not-fenced stored procedure, the database manager determines whether the error occurred in the application or database manager code and performs the appropriate recovery.

A not-fenced stored procedure can corrupt the database manager beyond recovery, possibly resulting in lost data and the possibility of a corrupt database. Exercise extreme caution when you run not-fenced trusted stored procedures. In almost all cases, the proper performance analysis of an application results in the good performance without using not-fenced stored procedures. For example, triggers might improve performance.

---

## Security Considerations when Using SQL in Applications

### Precompilation of embedded SQL applications with the PRECOMPILE command

Once you have created the embedded SQL application's source files, you must precompile each host language file containing SQL statements with the PREP command, using the options specific to the host language. The precompiler converts SQL statements contained in the source file to comments, and generates the DB2 run-time API calls for those statements.

You must always precompile a source file against a specific database, even if eventually you do not use the database with the application. In practice, you can use a test database for development, and after you fully test the application, you can bind its bind file to one or more production databases. This practice is known as *deferred binding*.

**Note:** Running an embedded application on an older client version than the client where precompilation occurred is not supported, regardless of where the application was compiled. For example, it is not supported to precompile an embedded application on a DB2 V9.5 client and then attempt to run the application on a DB2 V9.1 client.

If your application uses a code page that is not the same as your database code page, you need to consider which code page to use when precompiling.

If your application uses user-defined functions (UDFs) or user-defined distinct types (UDTs), you may need to use the FUNCPATH option when you precompile your application. This option specifies the function path that is used to resolve UDFs and UDTs for applications containing static SQL. If FUNCPATH is not specified, the default function path is *SYSIBM*, *SYSFUN*, *USER*, where *USER* refers to the current user ID.

Before precompiling an application you must connect to a server, either implicitly or explicitly. Although you precompile application programs at the client workstation and the precompiler generates modified source and messages on the client, the precompiler uses the server connection to perform some of the validation.

The precompiler also creates the information the database manager needs to process the SQL statements against a database. This information is stored in a package, in a bind file, or in both, depending on the precompiler options selected.

A typical example of using the precompiler follows. To precompile a C embedded SQL source file called *filename.sqc*, you can issue the following command to create a C source file with the default name *filename.c* and a bind file with the default name *filename.bnd*:

```
DB2 PREP filename.sqc BINDFILE
```

The precompiler generates up to four types of output:

#### **Modified Source**

This file is the new version of the original source file after the precompiler converts the SQL statements into DB2 run-time API calls. It is given the appropriate host language extension.

#### **Package**

If you use the PACKAGE option (the default), or do not specify any of the BINDFILE, SYNTAX, or SQLFLAG options, the package is stored in the connected database. The package contains all the information required to issue the static SQL statements of a particular source file against this database only. Unless you specify a different name with the PACKAGE USING option, the precompiler forms the package name from the first 8 characters of the source file name.

If you use the PACKAGE option without SQLERROR CONTINUE, the database used during the precompile process must contain all of the database objects referenced by the static SQL statements in the source file. For example, you cannot precompile a SELECT statement unless the table it references exists in the database.

With the VERSION option, the bindfile (if the BINDFILE option is used) and the package (either if bound at PREP time or if bound separately) will be designated with a particular version identifier. Many versions of packages with the same name and creator can exist at once.

#### **Bind File**

If you use the BINDFILE option, the precompiler creates a bind file (with extension *.bnd*) that contains the data required to create a package. This file can be used later with the BIND command to bind the application to one or more databases. If you specify BINDFILE and do not specify the PACKAGE option, binding is deferred until you invoke the BIND command. Note that for the command line processor (CLP), the default for PREP does

not specify the BINDFILE option. Thus, if you are using the CLP and want the binding to be deferred, you need to specify the BINDFILE option.

Specifying SQLERROR CONTINUE creates a package, even if errors occur when binding SQL statements. Those statements that fail to bind for authorization or existence reasons can be incrementally bound at execution time if VALIDATE RUN is also specified. Any attempt to issue them at run time generates an error.

#### Message File

If you use the MESSAGES option, the precompiler redirects messages to the indicated file. These messages include warning and error messages that describe problems encountered during precompilation. If the source file does not precompile successfully, use the warning and error messages to determine the problem, correct the source file, and then attempt to precompile the source file again. If you do not use the MESSAGES option, precompilation messages are written to the standard output.

## Compiling and linking source files containing embedded SQL

When precompiling embedded SQL source files, the PRECOMPILE command generates modified source files with a file extension applicable to the programming language.

Compile the modified source files (and any additional source files that do not contain SQL statements) using the appropriate host language compiler. The language compiler converts each modified source file into an *object module*.

Refer to the programming documentation for your operating platform for any exceptions to the default compiler options. Refer to your compiler's documentation for a complete description of available compiler options.

The host language linker creates an executable application. For example:

- On Windows operating systems, the application can be an executable file or a dynamic link library (DLL).
- On UNIX and Linux based operating systems, the application can be an executable load module or a shared library.

**Note:** Although applications can be DLLs on Windows operating systems, the DLLs are loaded directly by the application and not by the DB2 database manager. On Windows operating systems, the database manager loads embedded SQL stored procedures and user-defined functions as DLLs.

To create the executable file, link the following:

- User object modules, generated by the language compiler from the modified source files and other files not containing SQL statements.
- Host language library APIs, supplied with the language compiler.
- The database manager library containing the database manager APIs for your operating environment. Refer to the appropriate programming documentation for your operating platform for the specific name of the database manager library you need for your database manager APIs.

## Package recreation using the BIND command and an existing bind file

Binding is the process that creates the package the database manager needs to access the database when the application is executed. By default the PRECOMPILE command creates a package. Binding is done implicitly at precompile time unless the BINDFILE option is specified. The PACKAGE option allows you to specify a package name for the package created at precompile time.

A typical example of using the BIND command follows. To bind a bind file named *filename.bnd* to the database, you can issue the following command:

```
BIND filename.bnd
```

One package is created for each separately precompiled source code module. If an application has five source files, of which three require precompilation, three packages or bind files are created. By default, each package is given a name that is the same as the name of the source module from which the .bnd file originated, but truncated to 8 characters. To explicitly specify a different package name, you must use the PACKAGE USING option on the PREP command. The version of a package is given by the VERSION precompile option and defaults to the empty string. If the name and schema of this newly created package is the same as a package that currently exists in the target database, but the version identifier differs, a new package is created and the previous package still remains. However if a package exists that matches the name, schema and the version of the package being bound, then that package is dropped and replaced with the new package being bound (specifying ACTION ADD on the bind would prevent that and an error (SQL0719) would be returned instead).

## Generating sequential values

Generating sequential values is a common database application development problem. The best solution to that problem is to use sequences and sequence expressions in SQL. Each *sequence* is a uniquely named database object that can be accessed only by sequence expressions.

There are two *sequence expressions*: the PREVIOUS VALUE expression and the NEXT VALUE expression. The PREVIOUS VALUE expression returns the value most recently generated in the application process for the specified sequence. Any NEXT VALUE expressions occurring in the same statement as the PREVIOUS VALUE expression have no effect on the value generated by the PREVIOUS VALUE expression in that statement. The NEXT VALUE sequence expression increments the value of the sequence and returns the new value of the sequence.

To create a sequence, issue the CREATE SEQUENCE statement. For example, to create a sequence called *id\_values* using the default attributes, issue the following statement:

```
CREATE SEQUENCE id_values
```

To generate the first value in the application session for the sequence, issue a VALUES statement using the NEXT VALUE expression:

```
VALUES NEXT VALUE FOR id_values
```

```
1

1
```

```
1 record(s) selected.
```

To update the value of a column with the next value of the sequence, include the NEXT VALUE expression in the UPDATE statement, as follows:

```
UPDATE staff
 SET id = NEXT VALUE FOR id_values
 WHERE id = 350
```

To insert a new row into a table using the next value of the sequence, include the NEXT VALUE expression in the INSERT statement, as follows:

```
INSERT INTO staff (id, name, dept, job)
 VALUES (NEXT VALUE FOR id_values, 'Kandil', 51, 'Mgr')
```

## Managing sequence behavior

You can tailor the behavior of sequences to meet the needs of your application. You change the attributes of a sequence when you issue the CREATE SEQUENCE statement to create a new sequence, and when you issue the ALTER SEQUENCE statement for an existing sequence.

Following are some of the attributes of a sequence that you can specify:

### Data type

The AS clause of the CREATE SEQUENCE statement specifies the numeric data type of the sequence. The data type determines the possible minimum and maximum values of the sequence. The minimum and maximum values for a data type are listed in the *SQL Reference*. You cannot change the data type of a sequence; instead, you must drop the sequence by issuing the DROP SEQUENCE statement and issue a CREATE SEQUENCE statement with the new data type.

### Start value

The START WITH clause of the CREATE SEQUENCE statement sets the initial value of the sequence. The RESTART WITH clause of the ALTER SEQUENCE statement resets the value of the sequence to a specified value.

### Minimum value

The MINVALUE clause sets the minimum value of the sequence.

### Maximum value

The MAXVALUE clause sets the maximum value of the sequence.

### Increment value

The INCREMENT BY clause sets the value that each NEXT VALUE expression adds to the current value of the sequence. To decrement the value of the sequence, specify a negative value.

### Sequence cycling

The CYCLE clause causes the value of a sequence that reaches its maximum or minimum value to generate its respective minimum value or maximum value on the following NEXT VALUE expression.

**Note:** CYCLE should only be used if unique numbers are not required or if it can be guaranteed that older sequence values are not in use anymore once the sequence cycles.

For example, to create a sequence called id\_values that starts with a minimum value of 0, has a maximum value of 1000, increments by 2 with each NEXT VALUE expression, and returns to its minimum value when the maximum value is reached, issue the following statement:

```
CREATE SEQUENCE id_values
START WITH 0
INCREMENT BY 2
MAXVALUE 1000
CYCLE
```

## Sequences compared to identity columns

Although sequences and identity columns seem to serve similar purposes for DB2 applications, there is an important difference. An identity column automatically generates values for a column in a single table using the LOAD utility. A sequence generates sequential values upon request that can be used in any SQL statement using the CREATE SEQUENCE statement.

### Identity columns

Allow the database manager to automatically generate a unique numeric value for each row that is added to the table. If you are creating a table and you know you will need to uniquely identify each row that is added to that table, then you can add an identity column to the table definition as part of the CREATE TABLE statement:

```
CREATE TABLE <table name>
(<column name 1> INT,
 <column name 2>, DOUBLE,
 <column name 3> INT NOT NULL GENERATED ALWAYS AS IDENTITY
 (START WITH <value 1>, INCREMENT BY <value 2>))
```

In this example, the third column identifies the identity column. One of the attributes that you can define is the value used in the column to uniquely define each row when a row is added. The value following the INCREMENT BY clause shows by how much subsequent values of the identity column contents will be increased for every row added to the table.

Once created, the identity properties can be changed or removed using the ALTER TABLE statement. You can also use the ALTER TABLE statement to add identity properties on other columns.

### Sequences

Allow the automatic generation of values. Sequences are ideally suited to the task of generating unique key values. Applications can use sequences to avoid possible concurrency and performance problems resulting from the generation of a unique counter through other means. Unlike an identity column, a sequence is not tied to a particular table column, nor is it bound to a unique table column and only accessible through that table column.

A sequence can be created, and later altered, so that it generates values by incrementing or decrementing values either without a limit; or to a user-defined limit, and then stopping; or to a user-defined limit, then cycling back to the beginning and starting again. Sequences are only supported in single partition databases.

The following example shows how to create a sequence called orderseq:

```
CREATE SEQUENCE orderseq
START WITH 1
INCREMENT BY 1
NOMAXVALUE
NOCYCLE
CACHE 50
```

In this example, the sequence starts at 1 and increases by 1 with no upper limit. There is no reason to cycle back to the beginning and restart from 1



because there is no assigned upper limit. The CACHE parameter specifies the maximum number of sequence values that the database manager preallocates and keeps in memory.

## Authorization Considerations for Embedded SQL

An *authorization* allows a user or group to perform a general task such as connecting to a database, creating tables, or administering a system. A *privilege* gives a user or group the right to access one specific database object in a specified way. DB2® uses a set of privileges to provide protection for the information that you store in it.

Most SQL statements require some type of privilege on the database objects which the statement utilizes. Most API calls usually do not require any privilege on the database objects which the call utilizes, however, many APIs require that you possess the necessary authority to start them. You can use the DB2 APIs to perform the DB2 administrative functions from within your application program. For example, to recreate a package stored in the database without the need for a bind file, you can use the sqlarbind (or REBIND) API.

Groups provide a convenient means of performing authorization for a collection of users without having to grant or revoke privileges for each user individually. Group membership is considered for the execution of dynamic SQL statements, but not for static SQL statements. PUBLIC privileges are, however, considered for the execution of static SQL statements. For example, suppose you have an embedded SQL stored procedure with statically bound SQL queries against a table called STAFF. If you try to build this procedure with the CREATE PROCEDURE statement, and your account belongs to a group that has the select privilege for the STAFF table, the CREATE statement will fail with a SQL0551N error. For the CREATE statement to work, your account directly needs the select privilege on the STAFF table.

When you design your application, consider the privileges your users will need to run the application. The privileges required by your users depend on:

- Whether your application uses dynamic SQL, including JDBC and DB2 CLI, or static SQL. For information about the privileges required to issue a statement, see the description of that statement.
- Which APIs the application uses. For information about the privileges and authorities required for an API call, see the description of that API.

Groups provide a convenient means of performing authorization for a collection of users without having to grant or revoke privileges for each user individually. In general, group membership is considered for dynamic SQL statements, but is not considered for static SQL statements. The exception to this general case occurs when privileges are granted to PUBLIC: these are considered when static SQL statements are processed.

Consider two users, PAYROLL and BUDGET, who need to perform queries against the STAFF table. PAYROLL is responsible for paying the employees of the company, so it needs to issue a variety of SELECT statements when issuing paychecks. PAYROLL needs to be able to access each employee's salary. BUDGET is responsible for determining how much money is needed to pay the salaries. BUDGET should not, however, be able to see any particular employee's salary.

Because PAYROLL issues many different SELECT statements, the application you design for PAYROLL could probably make good use of dynamic SQL. The



dynamic SQL would require that PAYROLL have SELECT privilege on the STAFF table. This requirement is not a problem because PAYROLL requires full access to the table.

BUDGET, on the other hand, should not have access to each employee's salary. This means that you should not grant SELECT privilege on the STAFF table to BUDGET. Because BUDGET does need access to the total of all the salaries in the STAFF table, you could build a static SQL application to execute a SELECT SUM(SALARY) FROM STAFF, bind the application and grant the EXECUTE privilege on your application's package to BUDGET. This enables BUDGET to obtain the required information, without exposing the information that BUDGET should not see.

## Effect of DYNAMICRULES bind option on dynamic SQL

The PRECOMPILE command and BIND command option DYNAMICRULES determines what values apply at run-time for the following dynamic SQL attributes:

- The authorization ID that is used during authorization checking.
- The qualifier that is used for qualification of unqualified objects.
- Whether the package can be used to dynamically prepare the following statements: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY and SET EVENT MONITOR STATE statements.

In addition to the DYNAMICRULES value, the run-time environment of a package controls how dynamic SQL statements behave at run-time. The two possible run-time environments are:

- The package runs as part of a stand-alone program
- The package runs within a routine context

The combination of the DYNAMICRULES value and the run-time environment determine the values for the dynamic SQL attributes. That set of attribute values is called the dynamic SQL statement behavior. The four behaviors are:

### Run behavior

DB2 uses the authorization ID of the user (the ID that initially connected to DB2) executing the package as the value to be used for authorization checking of dynamic SQL statements and for the initial value used for implicit qualification of unqualified object references within dynamic SQL statements.

### Bind behavior

At run-time, DB2 uses all the rules that apply to static SQL for authorization and qualification. That is, take the authorization ID of the package owner as the value to be used for authorization checking of dynamic SQL statements and the package default qualifier for implicit qualification of unqualified object references within dynamic SQL statements.

### Define behavior

Define behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES DEFINEBIND or DYNAMICRULES DEFINERUN. DB2 uses the authorization ID of the routine definer (not the routine's package binder) as the value to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

## Invoke behavior

Invoke behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES INVOKEBIND or DYNAMICRULES INVOKERUN. DB2 uses the current statement authorization ID in effect when the routine is invoked as the value to be used for authorization checking of dynamic SQL and for implicit qualification of unqualified object references within dynamic SQL statements within that routine. This is summarized by the following table:

| Invoking Environment                        | ID Used                                                                                        |
|---------------------------------------------|------------------------------------------------------------------------------------------------|
| Any static SQL                              | Implicit or explicit value of the OWNER of the package the SQL invoking the routine came from. |
| Used in definition of view or trigger       | Definer of the view or trigger.                                                                |
| Dynamic SQL from a run behavior package     | ID used to make the initial connection to DB2.                                                 |
| Dynamic SQL from a define behavior package  | Definer of the routine that uses the package that the SQL invoking the routine came from.      |
| Dynamic SQL from an invoke behavior package | Current authorization ID invoking the routine.                                                 |

The following table shows the combination of the DYNAMICRULES value and the run-time environment that yields each dynamic SQL behavior.

Table 18. How DYNAMICRULES and the Run-Time Environment Determine Dynamic SQL Statement Behavior

| DYNAMICRULES Value | Behavior of Dynamic SQL Statements in a Standalone Program Environment | Behavior of Dynamic SQL Statements in a Routine 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 behavior.

Table 19. Definitions of Dynamic SQL Statement Behaviors

| Dynamic SQL Attribute | Setting for Dynamic SQL Attributes: Bind Behavior       | Setting for Dynamic SQL Attributes: Run Behavior | Setting for Dynamic SQL Attributes: Define Behavior | Setting for Dynamic SQL Attributes: Invoke Behavior         |
|-----------------------|---------------------------------------------------------|--------------------------------------------------|-----------------------------------------------------|-------------------------------------------------------------|
| Authorization ID      | The implicit or explicit value of the OWNER BIND option | ID of User Executing Package                     | Routine definer (not the routine's package owner)   | Current statement authorization ID when routine is invoked. |

Table 19. Definitions of Dynamic SQL Statement Behaviors (continued)

| Dynamic SQL Attribute                                                                                         | Setting for Dynamic SQL Attributes: Bind Behavior           | Setting for Dynamic SQL Attributes: Run Behavior | Setting for Dynamic SQL Attributes: Define Behavior | Setting for Dynamic SQL Attributes: Invoke Behavior         |
|---------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|--------------------------------------------------|-----------------------------------------------------|-------------------------------------------------------------|
| Default qualifier for unqualified objects                                                                     | The implicit or explicit value of the QUALIFIER BIND option | CURRENT SCHEMA Special Register                  | Routine definer (not the routine's package owner)   | Current statement authorization ID when routine is invoked. |
| Can execute GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY and SET EVENT MONITOR STATE | No                                                          | Yes                                              | No                                                  | No                                                          |

## Units of work and transactions

### Units of work

A transaction is commonly referred to in DB2 Database for Linux, UNIX, and Windows as a *unit of work*. A unit of work is a recoverable sequence of operations within an application process. It is used by the database manager to ensure that a database is in a consistent state. Any reading from or writing to the database is done within a unit of work.

For example, a bank transaction might involve the transfer of funds from a savings account to a checking account. After the application subtracts an amount from the savings account, the two accounts are inconsistent, and remain so until the amount is added to the checking account. When *both* steps are completed, a point of consistency is reached. The changes can be committed and made available to other applications.

A unit of work is started implicitly when the first SQL statement is issued against the database. All subsequent reads and writes by the same application are considered part of the same unit of work. The application must end the unit of work by issuing either a COMMIT or a ROLLBACK statement. The COMMIT statement makes permanent all changes made within a unit of work. The ROLLBACK statement removes these changes from the database. If the application ends normally without either of these statements being explicitly issued, the unit of work is automatically committed. If it ends abnormally in the middle of a unit of work, the unit of work is automatically rolled back. Once issued, a COMMIT or a ROLLBACK cannot be stopped. With some multi-threaded applications, or some operating systems (such as Windows), if the application ends normally without either of these statements being explicitly issued, the unit of work is automatically rolled back. It is recommended that your applications always explicitly commit or roll back complete units of work. If part of a unit of work does not complete successfully, the updates are rolled back, leaving the participating tables as they were before the transaction began. This ensures that requests are neither lost nor duplicated.

There is no physical representation of a unit of work because it is a series of instructions (SQL statements).

## Remote unit of work

A *remote unit of work* lets a user or application program read or update data at one location per unit of work. It supports access to one database within a unit of work. While an application program can update several remote databases, it can only access one database within a unit of work.

Remote unit of work has the following characteristics:

- Multiple requests (SQL statements) per unit of work are supported.
- Multiple cursors per unit of work are supported.
- Each unit of work can update only one database.
- The application program either commits or rolls back the unit of work. In certain error circumstances, the database server or DB2<sup>®</sup> Connect<sup>™</sup> might roll back the unit of work.

For example, Figure 2 shows a database client running a funds transfer application that accesses a database containing checking and savings account tables, as well as a transaction fee schedule. The application must:

- Accept the amount to transfer from the user interface.
- Subtract the amount from the savings account, and determine the new balance.
- Read the fee schedule to determine the transaction fee for a savings account with the given balance.
- Subtract the transaction fee from the savings account.
- Add the amount of the transfer to the checking account.
- Commit the transaction (unit of work).

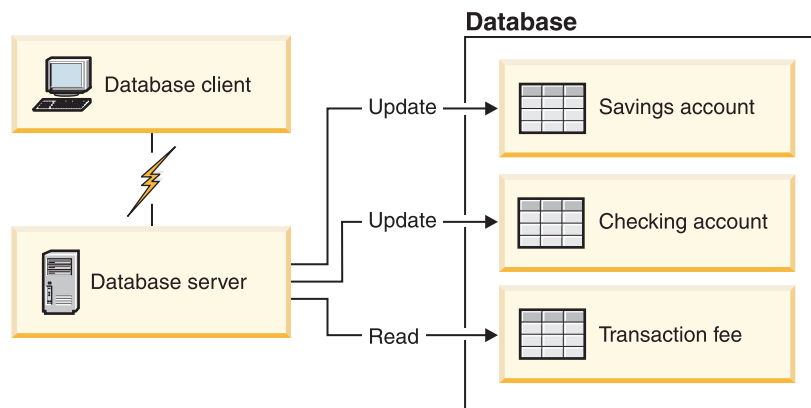


Figure 2. Using a Single Database in a Transaction

To set up such an application, you must:

1. Create the tables for the savings account, checking account and transaction fee schedule in the same database.
2. If physically remote, set up the database server to use the appropriate communications protocol.
3. If physically remote, catalog the node and the database to identify the database on the database server.
4. Precompile your application program to specify a type 1 connection; that is, specify CONNECT(1) on the PREP command.

## Concurrent transactions and multi-threaded database access in embedded SQL applications

One feature of some operating systems is the ability to run several threads of execution within a single process. The multiple threads allow an application to handle asynchronous events, and makes it easier to create event-driven applications, without resorting to polling schemes. The information that follows describes how the DB2 database manager works with multiple threads, and lists some design guidelines that you should keep in mind.

If you are not familiar with terms relating to the development of multi-threaded applications (such as critical section and semaphore), consult the programming documentation for your operating system.

A DB2 embedded SQL application can execute SQL statements from multiple threads using *contexts*. A context is the environment from which an application runs all SQL statements and API calls. All connections, units of work, and other database resources are associated with a specific context. Each context is associated with one or more threads within an application. Developing multi-threaded embedded SQL applications with thread-safe code is only supported in C and C++. It is possible to write your own precompiler, that along with features supplied by the language allows concurrent multithread database access.

For each executable SQL statement in a context, the first run-time services call always tries to obtain a latch. If it is successful, it continues processing. If not (because an SQL statement in another thread of the same context already has the latch), the call is blocked on a signaling semaphore until that semaphore is posted, at which point the call gets the latch and continues processing. The latch is held until the SQL statement has completed processing, at which time it is released by the last run-time services call that was generated for that particular SQL statement.

The net result is that each SQL statement within a context is executed as an atomic unit, even though other threads may also be trying to execute SQL statements at the same time. This action ensures that internal data structures are not altered by different threads at the same time. APIs also use the latch used by run-time services; therefore, APIs have the same restrictions as run-time services routines within each context.

Contexts may be exchanged between threads in a process, but not exchanged between processes. One use of multiple contexts is to provide support for concurrent transactions.

In the default implementation of threaded applications against a DB2 database, serialization of access to the database is enforced by the database APIs. If one thread performs a database call, calls made by other threads will be blocked until the first call completes, even if the subsequent calls access database objects that are unrelated to the first call. In addition, all threads within a process share a commit scope. True concurrent access to a database can only be achieved through separate processes, or by using the APIs that are described in this topic.

DB2 database systems provide APIs that can be used to allocate and manipulate separate environments (contexts) for the use of database APIs and embedded SQL. Each context is a separate entity, and any connection or attachment using one context is independent of all other contexts (and thus all other connections or attachments within a process). In order for work to be done on a context, it must

first be associated with a thread. A thread must always have a context when making database API calls or when using embedded SQL.

All DB2 database system applications are multithreaded by default, and are capable of using multiple contexts. You can use the following DB2 APIs to use multiple contexts. Specifically, your application can create a context for a thread, attach to or detach from a separate context for each thread, and pass contexts between threads. If your application does not call *any* of these APIs, DB2 will automatically manage the multiple contexts for your application:

- `sqlAttachToCtx` - Attach to context
- `sqlBeginCtx` - Create and attach to an application context
- `sqlDetachFromCtx` - Detach from context
- `sqlEndCtx` - Detach and destroy application context
- `sqlGetCurrentCtx` - Get current context
- `sqlInterruptCtx` - Interrupt context

These APIs have no effect (that is, they are no-ops) on platforms that do not support application threading.

Contexts need not be associated with a given thread for the duration of a connection or attachment. One thread can attach to a context, connect to a database, detach from the context, and then a second thread can attach to the context and continue doing work using the already existing database connection. Contexts can be passed around among threads in a process, but not among processes.

Even if the new APIs are used, the following APIs continue to be serialized:

- `sqlabndx` - Bind
- `sqlaprep` - Precompile Program
- `sqluexpr` - Export
- `db2Import` and `sqluimpr` - Import

**Note:**

1. The DB2 CLI automatically uses multiple contexts to achieve thread-safe, concurrent database access on platforms that support multi-threading. While not recommended by DB2, users can explicitly disable this feature if required.
2. By default, AIX does not permit 32-bit applications to attach to more than 11 shared memory segments per process, of which a maximum of 10 can be used for DB2 connections.

When this limit is reached, DB2 returns SQLCODE -1224 on an SQL CONNECT. DB2 Connect also has the 10-connection limitation if local users are running two-phase commit with a TP Monitor (TCP/IP).

The AIX environment variable EXTSHM can be used to increase the maximum number of shared memory segments to which a process can attach.

To use EXTSHM with DB2, do the following:

In client sessions:

```
export EXTSHM=ON
```

When starting the DB2 server:

```
export EXTSHM=ON
db2set DB2ENVLIST=EXTSHM
db2start
```

On DPF, also add the following lines to your userprofile or usershrc files:

```
EXTSHM=ON
export EXTSHM
```

An alternative is to move the local database or DB2 Connect into another machine and to access it remotely, or to access the local database or the DB2 Connect database with TCP/IP loop-back by cataloging it as a remote node that has the TCP/IP address of the local machine.

---

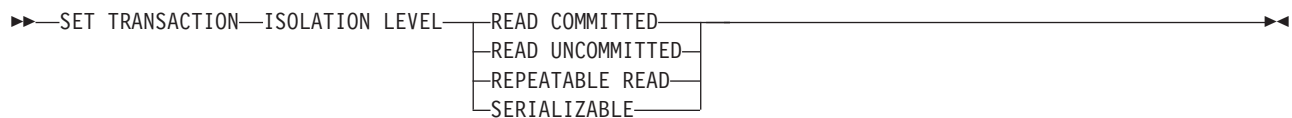
## Security and Java Applications

The sections that follow describe security considerations for SQLJ, JDBC, the Type 2 JDBC driver, and the Universal JDBC driver.

### SQLJ SET-TRANSACTION-clause

The SET TRANSACTION clause sets the isolation level for the current unit of work.

#### Syntax



#### Description

##### ISOLATION LEVEL

Specifies one of the following isolation levels:

##### READ COMMITTED

Specifies that the current DB2 isolation level is cursor stability.

##### READ UNCOMMITTED

Specifies that the current DB2 isolation level is uncommitted read.

##### REPEATABLE READ

Specifies that the current DB2 isolation level is read stability.

##### SERIALIZABLE

Specifies that the current DB2 isolation level is repeatable read.

#### Usage notes

You can execute SET TRANSACTION only at the beginning of a transaction.

### Setting the isolation level for an SQLJ transaction

To set the isolation level for a unit of work within an SQLJ program, use the SET TRANSACTION ISOLATION LEVEL clause.

The following table shows the values that you can specify in the SET TRANSACTION ISOLATION LEVEL clause and their DB2 equivalents.

Table 20. Equivalent SQLJ and DB2 isolation levels

| SET TRANSACTION value | DB2 isolation level |
|-----------------------|---------------------|
| SERIALIZABLE          | Repeatable read     |



Table 20. Equivalent SQLJ and DB2 isolation levels (continued)

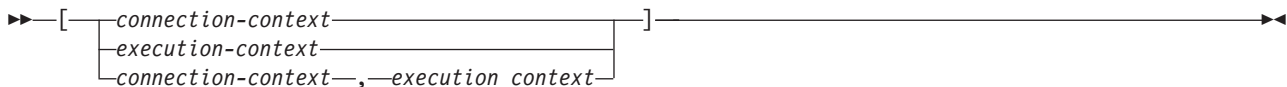
| SET TRANSACTION value | DB2 isolation level |
|-----------------------|---------------------|
| REPEATABLE READ       | Read stability      |
| READ COMMITTED        | Cursor stability    |
| READ UNCOMMITTED      | Uncommitted read    |

The isolation level affects the underlying JDBC connection as well as the SQLJ connection.

## SQLJ context-clause

A context clause specifies a connection context, an execution context, or both. You use a connection context to connect to a data source. You use an execution context to monitor and modify SQL statement execution.

### Syntax



### Description

#### connection-context

Specifies a valid Java identifier that is declared earlier in the SQLJ program. That identifier must be declared as an instance of the connection context class that SQLJ generates for a connection declaration clause.

#### execution-context

Specifies a valid Java identifier that is declared earlier in the SQLJ program. That identifier must be declared as an instance of class `sqlj.runtime.ExecutionContext`.

### Usage notes

- If you do not specify a connection context in an executable clause, SQLJ uses the default connection context.
- If you do not specify an execution context, SQLJ obtains the execution context from the connection context of the statement.

## Connecting to a data source using SQLJ

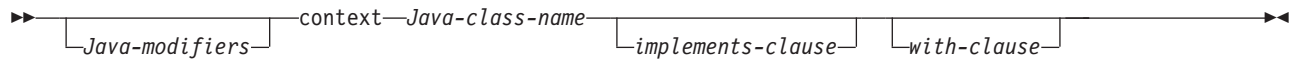
In an SQLJ application, as in any other DB2 application, you must be connected to a data source before you can execute SQL statements.

You can use one of six techniques to connect to a data source in an SQLJ program. Two use the JDBC DriverManager interface, two use the JDBC DataSource interface, one uses a previously created connection context, and one uses the default connection.

## SQLJ connection-declaration-clause

The connection declaration clause declares a connection to a data source in an SQLJ application program.

## Syntax



### Description

#### Java-modifiers

Specifies modifiers that are valid for Java class declarations, such as static, public, private, or protected.

#### Java-class-name

Specifies a valid Java identifier. During the program preparation process, SQLJ generates a connection context class whose name is this identifier.

#### implements-clause

See "SQLJ implements-clause" for a description of this clause. In a connection declaration clause, the interface class to which the implements clause refers must be a user-defined interface class.

#### with-clause

See "SQLJ with-clause" for a description of this clause.

### Usage notes

- SQLJ generates a connection class declaration for each connection declaration clause you specify. SQLJ data source connections are objects of those generated connection classes.
- You can specify a connection declaration clause anywhere that a Java class definition can appear in a Java program.

## Closing the connection to a data source in an SQLJ application

When you have finished with a connection to a data source, you need to close the connection to the data source. Doing so releases the connection context object's DB2 and SQLJ resources immediately.

To close the connection to the data source, use one of the `ConnectionContext.close` methods.

- If you execute `ConnectionContext.close()` or `ConnectionContext.close(ConnectionContext.CLOSE_CONNECTION)`, the connection context, as well as the connection to the data source, are closed.
- If you execute `ConnectionContext.close(ConnectionContext.KEEP_CONNECTION)` the connection context is closed, but the connection to the data source is not.

The following code closes the connection context, but does not close the connection to the data source.

```
...
ctx = new EzSqljctx(con0); // Create a connection context object
 // from JDBC connection con0
... // Perform various SQL operations
EzSqljctx.close(ConnectionContext.KEEP_CONNECTION);
 // Close the connection context but keep
 // the connection to the data source open
```

---

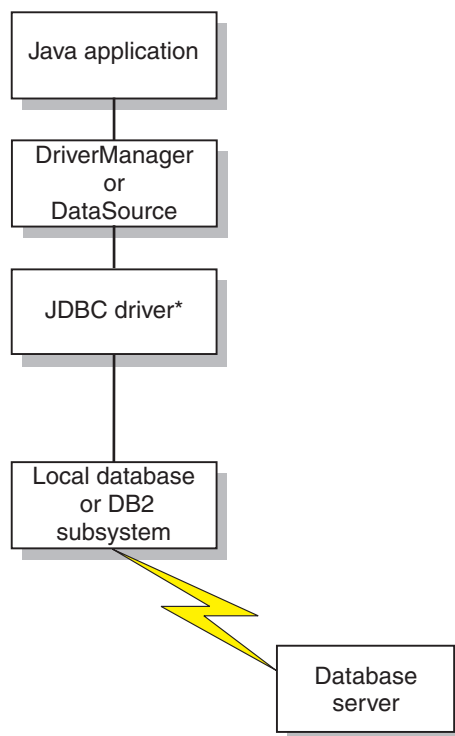
## JDBC Considerations

### How JDBC applications connect to a data source

Before you can execute SQL statements in any SQL program, you must be connected to a data source.

The IBM Data Server Driver for JDBC and SQLJ supports type 2 and type 4 connectivity. Connections to DB2 databases can use type 2 or type 4 connectivity. Connections to IBM Informix<sup>®</sup> Dynamic Server (IDS) databases can use type 4 connectivity.

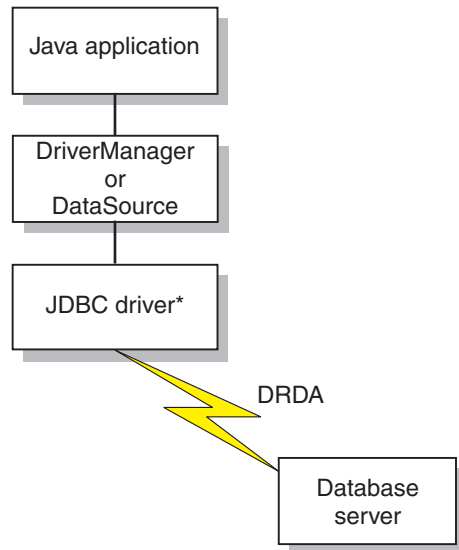
The following figure shows how a Java application connects to a data source using IBM Data Server Driver for JDBC and SQLJ type 2 connectivity.



\*Java byte code executed under JVM, and native code

*Figure 3. Java application flow for IBM Data Server Driver for JDBC and SQLJ type 2 connectivity*

The following figure shows how a Java application connects to a data source using IBM Data Server Driver for JDBC and SQLJ type 4 connectivity.



\*Java byte code executed under JVM

Figure 4. Java application flow for IBM Data Server Driver for JDBC and SQLJ type 4 connectivity

## Connecting to a data source using the DataSource interface

If your applications need to be portable among data sources, you should use the DataSource interface.

Using DriverManager to connect to a data source reduces portability because the application must identify a specific JDBC driver class name and driver URL. The driver class name and driver URL are specific to a JDBC vendor, driver implementation, and data source.

When you connect to a data source using the DataSource interface, you use a DataSource object.

The simplest way to use a DataSource object is to create and use the object in the same application, as you do with the DriverManager interface. However, this method does not provide portability.

The best way to use a DataSource object is for your system administrator to create and manage it separately, using WebSphere Application Server or some other tool. The program that creates and manages a DataSource object also uses the Java Naming and Directory Interface (JNDI) to assign a logical name to the DataSource object. The JDBC application that uses the DataSource object can then refer to the object by its logical name, and does not need any information about the underlying data source. In addition, your system administrator can modify the data source attributes, and you do not need to change your application program.

To learn more about using WebSphere to deploy DataSource objects, go to this URL on the Web:

<http://www.ibm.com/software/webservers/appserv/>

To learn about deploying DataSource objects yourself, see "Creating and deploying DataSource objects".

You can use the DataSource interface and the DriverManager interface in the same application, but for maximum portability, it is recommended that you use only the DataSource interface to obtain connections.

To obtain a connection using a DataSource object that the system administrator has already created and assigned a logical name to, follow these steps:

1. From your system administrator, obtain the logical name of the data source to which you need to connect.
2. Create a Context object to use in the next step. The Context interface is part of the Java Naming and Directory Interface (JNDI), not JDBC.
3. In your application program, use JNDI to get the DataSource object that is associated with the logical data source name.
4. Use the DataSource.getConnection method to obtain the connection.

You can use one of the following forms of the getConnection method:

```
getConnection();
getConnection(String user, String password);
```

Use the second form if you need to specify a user ID and password for the connection that are different from the ones that were specified when the DataSource was deployed.

*Example of obtaining a connection using a DataSource object that was created by the system administrator:* In this example, the logical name of the data source that you need to connect to is jdbc/sampledb. The numbers to the right of selected statements correspond to the previously-described steps.

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
Context ctx=new InitialContext();
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
Connection con=ds.getConnection();
```

2  
3  
4

Figure 5. Obtaining a connection using a DataSource object

*Example of creating and using a DataSource object in the same application:*

Figure 6. Creating and using a DataSource object in the same application

```
import java.sql.*; // JDBC base
import javax.sql.*; // Additional methods for JDBC
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC and SQLJ
 // interfaces
DB2SimpleDataSource dbds=new DB2SimpleDataSource();
dbds.setDatabaseName("dbloc1");
 // Assign the location name
dbds.setDescription("Our Sample Database");
 // Description for documentation
dbds.setUser("john");
 // Assign the user ID
dbds.setPassword("dbadm");
 // Assign the password
Connection con=dbds.getConnection();
 // Create a Connection object
```

1

2  
3

4

| Note | Description                                                                      |
|------|----------------------------------------------------------------------------------|
| 1    | Import the package that contains the implementation of the DataSource interface. |

| Note | Description                                                                                                                                                                                                                                                                                                                             |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2    | Creates a DB2SimpleDataSource object. DB2SimpleDataSource is one of the IBM Data Server Driver for JDBC and SQLJ implementations of the DataSource interface. See "Creating and deploying DataSource objects" for information on DB2's DataSource implementations.                                                                      |
| 3    | The setDatabaseName, setDescription, setUser, and setPassword methods assign attributes to the DB2SimpleDataSource object. See "Properties for the IBM Data Server Driver for JDBC and SQLJ" for information about the attributes that you can set for a DB2SimpleDataSource object under the IBM Data Server Driver for JDBC and SQLJ. |
| 4    | Establishes a connection to the data source that DB2SimpleDataSource object dbds represents.                                                                                                                                                                                                                                            |

## JDBC connection objects

When you connect to a data source by either connection method, you create a Connection object, which represents the connection to the data source.

You use this Connection object to do the following things:

- Create Statement, PreparedStatement, and CallableStatement objects for executing SQL statements. These are discussed in "Executing SQL statements in JDBC applications".
- Gather information about the data source to which you are connected. This process is discussed in "Learning about a data source using DatabaseMetaData methods".
- Commit or roll back transactions. You can commit transactions manually or automatically. These operations are discussed in "Commit or roll back a JDBC transaction".
- Close the connection to the data source. This operation is discussed in "Disconnecting from data sources in JDBC applications".

## Committing or rolling back JDBC transactions

In JDBC, to commit or roll back transactions explicitly, use the commit or rollback methods.

For example:

```
Connection con;
...
con.commit();
```

If autocommit mode is on, the database manager performs a commit operation after every SQL statement completes. To set autocommit mode on, invoke the Connection.setAutoCommit(true) method. To set autocommit mode off, invoke the Connection.setAutoCommit(false) method. To determine whether autocommit mode is on, invoke the Connection.getAutoCommit method.

Connections that participate in distributed transactions cannot invoke the setAutoCommit(true) method.

When you change the autocommit state, the database manager executes a commit operation, if the application is not already on a transaction boundary.

While a connection is participating in a distributed transaction, the associated application cannot issue the commit or rollback methods.

## Disconnecting from data sources in JDBC applications

When you have finished with a connection to a data source, it is *essential* that you close the connection to the data source. Doing this releases the Connection object's database and JDBC resources immediately.

To close the connection to the data source, use the close method. For example:

```
Connection con;
...
con.close();
```

For a connection to a DB2 data source, if autocommit mode is not on, the connection needs to be on a unit-of-work boundary before you close the connection.

For a connection to an IBM Informix Dynamic Server database, if the database supports logging, and autocommit mode is not on, the connection needs to be on a unit-of-work boundary before you close the connection.

---

## Type 2 JDBC Driver Considerations

### Security under the DB2 JDBC Type 2 Driver

The DB2 JDBC Type 2 Driver for Linux, UNIX and Windows (DB2 JDBC Type 2 Driver) supports user ID and password security.

You must set the user ID and the password, or set neither. If you do not set a user ID and password, the driver uses the user ID and password of the user who is currently logged on to the operating system.

To specify user ID and password security for a JDBC connection, use one of the following techniques.

*For the DriverManager interface:* you can specify the user ID and password directly in the DriverManager.getConnection invocation. For example:

```
import java.sql.*; // JDBC base
...
String id = "db2adm"; // Set user ID
String pw = "db2adm"; // Set password
String url = "jdbc:db2:toronto";
 // Set URL for the data source
Connection con = DriverManager.getConnection(url, id, pw);
 // Create connection
```

Alternatively, you can set the user ID and password by setting the user and password properties in a Properties object, and then invoking the form of the getConnection method that includes the Properties object as a parameter. For example:

```
import java.sql.*; // JDBC base
import COM.ibm.db2.jdbc.*; // DB2 JDBC Type 2 driver
 // implementation of JDBC
...
Properties properties = new java.util.Properties();
 // Create Properties object
properties.put("user", "db2adm"); // Set user ID for the connection
properties.put("password", "db2adm"); // Set password for the connection
String url = "jdbc:db2:toronto";
```



```

// Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
// Create connection

```

*For the DataSource interface:* you can specify the user ID and password directly in the DataSource.getConnection invocation. For example:

```

import java.sql.*; // JDBC base
import COM.ibm.db2.jdbc.*; // DB2 JDBC Type 2 driver
// implementation of JDBC
...
Context ctx=new InitialContext(); // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledbs");
// Get DataSource object
String id = "db2adm"; // Set user ID
String pw = "db2adm"; // Set password
Connection con = ds.getConnection(id, pw);
// Create connection

```

Alternatively, if you create and deploy the DataSource object, you can set the user ID and password by invoking the DataSource.setUser and DataSource.setPassword methods after you create the DataSource object. For example:

```

import java.sql.*; // JDBC base
import COM.ibm.db2.jdbc.*; // DB2 JDBC Type 2 driver
// implementation of JDBC
...
DB2DataSource db2ds = new DB2DataSource();
// Create DataSource object
db2ds.setDatabaseName("toronto"); // Set location
db2ds.setUser("db2adm"); // Set user ID
db2ds.setPassword("db2adm"); // Set password

```

## How DB2 applications connect to a data source using the DriverManager interface with the DB2 JDBC Type 2 Driver

A JDBC application can establish a connection to a data source using the JDBC DriverManager interface, which is part of the java.sql package.

The Java application first loads the JDBC driver by invoking the Class.forName method. After the application loads the driver, it connects to a data source by invoking the DriverManager.getConnection method.

For the DB2 JDBC Type 2 Driver for Linux, UNIX and Windows (DB2 JDBC Type 2 Driver), you load the driver by invoking the Class.forName method with the following argument:

```
COM.ibm.db2.jdbc.app.DB2Driver
```

The following code demonstrates loading the DB2 JDBC Type 2 Driver:

```

try {
 // Load the DB2 JDBC Type 2 Driver with DriverManager
 Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
} catch (ClassNotFoundException e) {
 e.printStackTrace();
}

```

The catch block is used to print an error if the driver is not found.

After you load the driver, you connect to the data source by invoking the DriverManager.getConnection method. You can use one of the following forms of getConnection:

```
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

The *url* argument represents a data source.

For the DB2 JDBC Type 2 Driver, specify a URL of the following form:

*Syntax for a URL for the DB2 JDBC Type 2 Driver:*

```
▶▶—jdbc—:—db2—:—database—————▶▶
```

The parts of the URL have the following meanings:

**jdbc:db2:**

jdbc:db2: indicates that the connection is to a DB2 data source.

**database**

A database alias. The alias refers to the DB2 database catalog entry on the DB2 client.

The *info* argument is an object of type `java.util.Properties` that contains a set of driver properties for the connection. Specifying the *info* argument is an alternative to specifying *property=value* strings in the URL.

*Specifying a user ID and password for a connection:* There are several ways to specify a user ID and password for a connection:

- Use the form of the `getConnection` method that specifies *user* and *password*.
- Use the form of the `getConnection` method that specifies *info*, after setting the user and password properties in a `java.util.Properties` object.

*Example: Setting the user ID and password in user and password parameters:*

```
String url = "jdbc:db2:toronto";
// Set URL for data source
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
// Create connection
```

*Example: Setting the user ID and password in a java.util.Properties object:*

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "db2adm"); // Set user ID for connection
properties.put("password", "db2adm"); // Set password for connection
String url = "jdbc:db2:toronto";
// Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
// Create connection
```

---

## Universal JDBC Driver Considerations

### User ID and password security under the IBM Data Server Driver for JDBC and SQLJ

With the IBM Data Server Driver for JDBC and SQLJ, one of the available security methods is user ID and password security.

To specify user ID and password security for a JDBC connection, use one of the following techniques.

**For the *DriverManager* interface:** You can specify the user ID and password directly in the `DriverManager.getConnection` invocation. For example:

```
import java.sql.*; // JDBC base
...
String id = "dbadm"; // Set user ID
String pw = "dbadm"; // Set password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
 // Set URL for the data source

Connection con = DriverManager.getConnection(url, id, pw);
 // Create connection
```

Another method is to set the user ID and password directly in the URL string. For example:

```
import java.sql.*; // JDBC base
...
String url =
 "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose:user=dbadm;password=dbadm;";

 // Set URL for the data source
Connection con = DriverManager.getConnection(url);
 // Create connection
```

Alternatively, you can set the user ID and password by setting the user and password properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. Optionally, you can set the `securityMechanism` property to indicate that you are using user ID and password security. For example:

```
import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC
 // and SQLJ implementation of JDBC
...
Properties properties = new java.util.Properties();
 // Create Properties object
properties.put("user", "dbadm"); // Set user ID for the connection
properties.put("password", "dbadm"); // Set password for the connection
properties.put("securityMechanism",
 new String("" + com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY +
 ""));
 // Set security mechanism to
 // user ID and password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
 // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
 // Create connection
```

**For the *DataSource* interface:** you can specify the user ID and password directly in the `DataSource.getConnection` invocation. For example:

```
import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC
 // and SQLJ implementation of JDBC
...
Context ctx=new InitialContext(); // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
 // Get DataSource object
String id = "dbadm"; // Set user ID
String pw = "dbadm"; // Set password
Connection con = ds.getConnection(id, pw);
 // Create connection
```

Alternatively, if you create and deploy the `DataSource` object, you can set the user ID and password by invoking the `DataSource.setUser` and `DataSource.setPassword` methods after you create the `DataSource` object. Optionally, you can invoke the `DataSource.setSecurityMechanism` method property to indicate that you are using user ID and password security. For example:

```
...
com.ibm.db2.jcc.DB2SimpleDataSource ds = // Create DB2SimpleDataSource object
 new com.ibm.db2.jcc.DB2SimpleDataSource();
ds.setDriverType(4); // Set driver type
ds.setDatabaseName("san_jose"); // Set location
ds.setServerName("mvs1.sj.ibm.com"); // Set server name
ds.setPortNumber(5021); // Set port number
ds.setUser("dbadm"); // Set user ID
ds.setPassword("dbadm"); // Set password
ds.setSecurityMechanism(
 com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY);
 // Set security mechanism to
 // user ID and password
```

## User ID-only security under the IBM Data Server Driver for JDBC and SQLJ

With the IBM Data Server Driver for JDBC and SQLJ, one of the available security methods is user-ID only security.

To specify user ID security for a JDBC connection, use one of the following techniques.

*For the `DriverManager` interface:* Set the user ID and security mechanism by setting the `user` and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example:

```
import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver
 // for JDBC and SQLJ
 // implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm"); // Set user ID for the connection
properties.put("securityMechanism",
 new String("" + com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY + ""));
 // Set security mechanism to
 // user ID only
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
 // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
 // Create the connection
```

*For the `DataSource` interface:* If you create and deploy the `DataSource` object, you can set the user ID and security mechanism by invoking the `DataSource.setUser` and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example:

```
import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver
 // for JDBC and SQLJ
 // implementation of JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
 // Create DB2SimpleDataSource object
```

```

db2ds.setDriverType(4); // Set the driver type
db2ds.setDatabaseName("san_jose"); // Set the location
db2ds.setServerName("mvs1.sj.ibm.com"); // Set the server name
db2ds.setPortNumber(5021); // Set the port number
db2ds.setUser("db2adm"); // Set the user ID
db2ds.setSecurityMechanism(
 com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY);
// Set security mechanism to
// user ID only

```

## Kerberos security under the IBM Data Server Driver for JDBC and SQLJ

JDBC support for Kerberos security is available for IBM Data Server Driver for JDBC and SQLJ type 4 connectivity only.

To enable JDBC support for Kerberos security, you also need to enable the following components of your software development kit (SDK) for Java:

- Java Cryptography Extension
- Java Generic Security Service (JGSS)
- Java Authentication and Authorization Service (JAAS)

See the documentation for your SDK for Java for information on how to enable these components.

There are three ways to specify Kerberos security for a connection:

- With a user ID and password
- Without a user ID or password
- With a delegated credential

### Kerberos security with a user ID and password

For this case, Kerberos uses the specified user ID and password to obtain a ticket-granting ticket (TGT) that lets you authenticate to the database server.

You need to set the user, password, `kerberosServerPrincipal`, and `securityMechanism` properties. Set the `securityMechanism` property to `com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY` (11). The `kerberosServerPrincipal` property specifies the principal name that the database server registers with a Kerberos Key Distribution Center (KDC).

*For the `DriverManager` interface:* Set the user ID, password, Kerberos server, and security mechanism by setting the user, password, `kerberosServerPrincipal`, and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example, use code like this to set the Kerberos security mechanism with a user ID and password:

```

import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC
 // and SQLJ implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm"); // Set user ID for the connection
properties.put("password", "db2adm"); // Set password for the connection
properties.put("kerberosServerPrincipal",
 "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
// Set the Kerberos server
properties.put("securityMechanism",

```

```

 new String("" +
com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + "");
// Set security mechanism to
// Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
// Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
// Create the connection

```

*For the DataSource interface:* If you create and deploy the DataSource object, set the Kerberos server and security mechanism by invoking the DataSource.setKerberosServerPrincipal and DataSource.setSecurityMechanism methods after you create the DataSource object. For example:

```

import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC
// and SQLJ implementation of JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
// Create the DataSource object
db2ds.setDriverType(4); // Set the driver type
db2ds.setDatabaseName("san_jose"); // Set the location
db2ds.setUser("db2adm"); // Set the user
db2ds.setPassword("db2adm"); // Set the password
db2ds.setServerName("mvs1.sj.ibm.com");
// Set the server name
db2ds.setPortNumber(5021); // Set the port number
db2ds.setKerberosServerPrincipal(
 "sample/srv1sj.ibm.com@SRVLSJ.SJ.IBM.COM");
// Set the Kerberos server
db2ds.setSecurityMechanism(
 com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
// Set security mechanism to
// Kerberos

```

## Kerberos security with no user ID or password

For this case, the Kerberos default credentials cache must contain a ticket-granting ticket (TGT) that lets you authenticate to the database server.

You need to set the kerberosServerPrincipal and securityMechanism properties. Set the securityMechanism property to com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS\_SECURITY (11).

*For the DriverManager interface:* Set the Kerberos server and security mechanism by setting the kerberosServerPrincipal and securityMechanism properties in a Properties object, and then invoking the form of the getConnection method that includes the Properties object as a parameter. For example, use code like this to set the Kerberos security mechanism without a user ID and password:

```

import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC
// and SQLJ implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal",
 "sample/srv1sj.ibm.com@SRVLSJ.SJ.IBM.COM");
// Set the Kerberos server
properties.put("securityMechanism",
 new String("" +
com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
// Set security mechanism to
// Kerberos

```

```
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
// Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
// Create the connection
```

*For the DataSource interface:* If you create and deploy the DataSource object, set the Kerberos server and security mechanism by invoking the DataSource.setKerberosServerPrincipal and DataSource.setSecurityMechanism methods after you create the DataSource object. For example:

```
import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC
// and SQLJ implementation of JDBC
...
DB2SimpleDataSource db2ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
// Create the DataSource object
db2ds.setDriverType(4); // Set the driver type
db2ds.setDatabaseName("san_jose"); // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");
// Set the server name
db2ds.setPortNumber(5021); // Set the port number
db2ds.setKerberosServerPrincipal(
 "sample/srv1sj.ibm.com@SRVLSJ.SJ.IBM.COM");
// Set the Kerberos server
db2ds.setSecurityMechanism(
 com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
// Set security mechanism to
// Kerberos
```

## Kerberos security with a delegated credential from another principal

For this case, you authenticate to the database server using a delegated credential that another principal passes to you.

You need to set the kerberosServerPrincipal, gssCredential, and securityMechanism properties. Set the securityMechanism property to com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS\_SECURITY (11).

*For the DriverManager interface:* Set the Kerberos server, delegated credential, and security mechanism by setting the kerberosServerPrincipal, and securityMechanism properties in a Properties object. Then invoke the form of the getConnection method that includes the Properties object as a parameter. For example, use code like this to set the Kerberos security mechanism without a user ID and password:

```
import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC
// and SQLJ implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal",
 "sample/srv1sj.ibm.com@SRVLSJ.SJ.IBM.COM");
// Set the Kerberos server
properties.put("gssCredential",delegatedCredential);
// Set the delegated credential
properties.put("securityMechanism",
 new String(" +
 com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + "));
// Set security mechanism to
// Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
```



```

// Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
// Create the connection

```

*For the DataSource interface:* If you create and deploy the DataSource object, set the Kerberos server, delegated credential, and security mechanism by invoking the DataSource.setKerberosServerPrincipal, DataSource.setGssCredential, and DataSource.setSecurityMechanism methods after you create the DataSource object. For example:

```

DB2SimpleDataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
// Create the DataSource object
db2ds.setDriverType(4); // Set the driver type
db2ds.setDatabaseName("san_jose"); // Set the location
db2ds.setServerName("mvs1.sj.ibm.com"); // Set the server name
db2ds.setPortNumber(5021); // Set the port number
db2ds.setKerberosServerPrincipal(
 "sample/srv1sj.ibm.com@SRVLSJ.SJ.IBM.COM");
// Set the Kerberos server
db2ds.setGssCredential(delegatedCredential);
// Set the delegated credential
db2ds.setSecurityMechanism(
 com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
// Set security mechanism to
// Kerberos

```

## Encrypted password, user ID, or user ID and password security under the IBM Data Server Driver for JDBC and SQLJ

IBM Data Server Driver for JDBC and SQLJ supports encrypted password security, encrypted user ID security, or encrypted user ID and encrypted password security for accessing data sources.

The IBM Data Server Driver for JDBC and SQLJ supports 56-bit DES (weak) encryption or 256-bit AES (strong) encryption. AES encryption is available with IBM Data Server Driver for JDBC and SQLJ type 4 connectivity only. You set the encryptionAlgorithm driver property to choose between 56-bit DES encryption (encryptionAlgorithm value of 1) and 256-bit AES encryption (encryptionAlgorithm value of 2). 256-bit AES encryption is used for a connection only if the database server supports it and is configured to use it.

If you use encrypted password security, encrypted user ID security, or encrypted user ID and encrypted password security, the IBM Java Cryptography Extension (JCE) needs to be enabled on your client. The IBM JCE is part of the IBM SDK for Java, Version 1.4.2 or later.

The IBM JCE needs to use 56-bit DES or 256-bit AES encrypted client/server communication from the IBM Data Server Driver for JDBC and SQLJ driver to DB2 Database for Linux, UNIX, and Windows servers.

For AES encryption, you need to get the unrestricted policy file for JCE. It is available at the following URL: <https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk>

Connections to DB2 for i V6R1 or later servers can use encrypted password security or encrypted user ID and encrypted password security. For encrypted password security or encrypted user ID and encrypted password security, the IBM Java Cryptography Extension (ibmjceprovider.jar) must be installed on your client. The IBM JCE is part of the IBM SDK for Java, Version 1.4.2 or later.

You can also use encrypted security-sensitive data in addition to encrypted user ID security or encrypted user ID and encrypted password security. You specify encryption of security-sensitive data through the `ENCRYPTED_USER_AND_DATA_SECURITY` or `ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY` `securityMechanism` value. `ENCRYPTED_USER_AND_DATA_SECURITY` is valid for connections to DB2 for z/OS<sup>®</sup> servers only, and only for DES encryption (`encryptionAlgorithm` value of 1).

DB2 for z/OS or DB2 Database for Linux, UNIX, and Windows database servers encrypt the following data when you specify encryption of security-sensitive data:

- SQL statements that are being prepared, executed, or bound into a package
- Input and output parameter information
- Result sets
- LOB data
- XML data
- Results of describe operations

Before you can use encrypted security-sensitive data, the z/OS Integrated Cryptographic Services Facility needs to be installed and enabled on the z/OS operating system.

To specify encrypted user ID or encrypted password security for a JDBC connection, use one of the following techniques.

***For the DriverManager interface:*** Set the user ID, password, and security mechanism by setting the `user`, `password`, and `securityMechanism` properties in a Properties object, and then invoking the form of the `getConnection` method that includes the Properties object as a parameter. For example, use code like this to set the user ID and encrypted password security mechanism, with AES encryption:

```
import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC
 // and SQLJ implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "dbadm"); // Set user ID for the connection
properties.put("password", "dbadm"); // Set password for the connection
properties.put("securityMechanism", "2");
 new String(" + com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY +
 "");
 // Set security mechanism to
 // user ID and encrypted password
properties.put("encryptionAlgorithm", "2");
 // Request AES security
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
 // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
 // Create the connection
```

***For the DataSource interface:*** If you create and deploy the `DataSource` object, you can set the user ID, password, and security mechanism by invoking the `DataSource.setUser`, `DataSource.setPassword`, and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example, use code like this to set the encrypted user ID and encrypted password security mechanism, with AES encryption:

```
import java.sql.*; // JDBC base
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC
 // and SQLJ implementation of JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource ds =
```

```

new com.ibm.db2.jcc.DB2SimpleDataSource();
// Create the DataSource object
ds.setDriverType(4); // Set the driver type
ds.setDatabaseName("san_jose"); // Set the location
ds.setServerName("mvs1.sj.ibm.com");
// Set the server name
ds.setPortNumber(5021); // Set the port number
ds.setUser("db2adm"); // Set the user ID
ds.setPassword("db2adm"); // Set the password
ds.setSecurityMechanism(
com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY);
// Set security mechanism to
// User ID and encrypted password
ds.setEncryptionAlgorithm(2); // Request AES encryption

```

## Security under the IBM Data Server Driver for JDBC and SQLJ

When you use the IBM Data Server Driver for JDBC and SQLJ, you choose a security mechanism by specifying a value for the `securityMechanism` property.

You can set this property in one of the following ways:

- If you use the `DriverManager` interface, set `securityMechanism` in a `java.util.Properties` object before you invoke the form of the `getConnection` method that includes the `java.util.Properties` parameter.
- If you use the `DataSource` interface, and you are creating and deploying your own `DataSource` objects, invoke the `DataSource.setSecurityMechanism` method after you create a `DataSource` object.

You can determine the security mechanism that is in effect for a connection by calling the `DB2Connection.getDB2SecurityMechanism` method.

The following table lists the security mechanisms that the IBM Data Server Driver for JDBC and SQLJ supports, and the data sources that support those security mechanisms.

Table 21. Database server support for IBM Data Server Driver for JDBC and SQLJ security mechanisms

| Security mechanism                                                           | Supported by                              |              |                             |                  |
|------------------------------------------------------------------------------|-------------------------------------------|--------------|-----------------------------|------------------|
|                                                                              | DB2 Database for Linux, UNIX, and Windows | DB2 for z/OS | IBM Informix Dynamic Server | DB2 for i        |
| User ID and password                                                         | Yes                                       | Yes          | Yes                         | Yes              |
| User ID only                                                                 | Yes                                       | Yes          | Yes                         | Yes              |
| User ID and encrypted password                                               | Yes                                       | Yes          | Yes                         | Yes <sup>2</sup> |
| Encrypted user ID                                                            | Yes                                       | Yes          | No                          | No               |
| Encrypted user ID and encrypted password                                     | Yes                                       | Yes          | Yes                         | Yes <sup>2</sup> |
| Encrypted user ID and encrypted security-sensitive data                      | No                                        | Yes          | No                          | No               |
| Encrypted user ID, encrypted password, and encrypted security-sensitive data | Yes                                       | Yes          | No                          | No               |
| Kerberos <sup>1</sup>                                                        | Yes                                       | Yes          | No                          | Yes              |

Table 21. Database server support for IBM Data Server Driver for JDBC and SQLJ security mechanisms (continued)

| Security mechanism  | Supported by |    |    |    |
|---------------------|--------------|----|----|----|
| Plugin <sup>1</sup> | Yes          | No | No | No |

**Note:**

1. Available for IBM Data Server Driver for JDBC and SQLJ type 4 connectivity only.
2. The version of the data source must be DB2 for i V6R1 or later.

The following table lists the security mechanisms that the IBM Data Server Driver for JDBC and SQLJ supports, and the value that you need to specify for the securityMechanism property to specify each security mechanism.

The default security mechanism is CLEAR\_TEXT\_PASSWORD\_SECURITY. If the server does not support CLEAR\_TEXT\_PASSWORD\_SECURITY but supports ENCRYPTED\_USER\_AND\_PASSWORD\_SECURITY, the IBM Data Server Driver for JDBC and SQLJ driver updates the security mechanism to ENCRYPTED\_USER\_AND\_PASSWORD\_SECURITY and attempts to connect to the server. Any other mismatch in security mechanism support between the requester and the server results in an error.

Table 22. Security mechanisms supported by the IBM Data Server Driver for JDBC and SQLJ

| Security mechanism                                                           | securityMechanism property value                            |
|------------------------------------------------------------------------------|-------------------------------------------------------------|
| User ID and password                                                         | DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY              |
| User ID only                                                                 | DB2BaseDataSource.USER_ONLY_SECURITY                        |
| User ID and encrypted password                                               | DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY               |
| Encrypted user ID                                                            | DB2BaseDataSource.ENCRYPTED_USER_ONLY_SECURITY              |
| Encrypted user ID and encrypted password                                     | DB2BaseDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY      |
| Encrypted user ID and encrypted security-sensitive data                      | DB2BaseDataSource.ENCRYPTED_USER_AND_DATA_SECURITY          |
| Encrypted user ID, encrypted password, and encrypted security-sensitive data | DB2BaseDataSource.ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY |
| Kerberos                                                                     | DB2BaseDataSource.KERBEROS_SECURITY                         |
| Plugin                                                                       | DB2BaseDataSource.PLUGIN_SECURITY                           |

The following table shows possible DB2 Database for Linux, UNIX, and Windows server authentication types and the compatible IBM Data Server Driver for JDBC and SQLJ securityMechanism property values.

Table 23. Compatible DB2 Database for Linux, UNIX, and Windows server authentication types and IBM Data Server Driver for JDBC and SQLJ securityMechanism values

| DB2 Database for Linux, UNIX, and Windows server authentication type | securityMechanism setting                                                                          |
|----------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| CLIENT                                                               | USER_ONLY_SECURITY                                                                                 |
| SERVER                                                               | CLEAR_TEXT_PASSWORD_SECURITY                                                                       |
| SERVER_ENCRYPT                                                       | CLEAR_TEXT_PASSWORD_SECURITY, ENCRYPTED_PASSWORD_SECURITY, or ENCRYPTED_USER_AND_PASSWORD_SECURITY |
| DATA_ENCRYPT                                                         | ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY                                                          |

Table 23. Compatible DB2 Database for Linux, UNIX, and Windows server authentication types and IBM Data Server Driver for JDBC and SQLJ securityMechanism values (continued)

| DB2 Database for Linux, UNIX, and Windows server authentication type                                                                                                                                                                                            | securityMechanism setting                                                                                                              |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| KERBEROS                                                                                                                                                                                                                                                        | KERBEROS_SECURITY or PLUGIN_SECURITY <sup>2</sup>                                                                                      |
| KRB_SERVER_ENCRYPT                                                                                                                                                                                                                                              | KERBEROS_SECURITY , PLUGIN_SECURITY <sup>1</sup> , ENCRYPTED_PASSWORD_SECURITY, or ENCRYPTED_USER_AND_PASSWORD_SECURITY                |
| GSSPLUGIN                                                                                                                                                                                                                                                       | PLUGIN_SECURITY <sup>1</sup> or KERBEROS_SECURITY                                                                                      |
| GSS_SERVER_ENCRYPT <sup>3</sup>                                                                                                                                                                                                                                 | CLEAR_TEXT_PASSWORD_SECURITY, ENCRYPTED_PASSWORD_SECURITY, ENCRYPTED_USER_AND_PASSWORD_SECURITY, PLUGIN_SECURITY, or KERBEROS_SECURITY |
| <b>Notes:</b><br>1. For PLUGIN_SECURITY, the plugin must be a Kerberos plugin.<br>2. For PLUGIN_SECURITY, one of the plugins at the server identifies itself as supporting Kerberos.<br>3. GSS_SERVER_ENCRYPT is a combination of GSSPLUGIN and SERVER_ENCRYPT. |                                                                                                                                        |

## Connecting to a data source using the DriverManager interface with the IBM Data Server Driver for JDBC and SQLJ

A JDBC application can establish a connection to a data source using the JDBC DriverManager interface, which is part of the java.sql package.

The steps for establishing a connection are:

1. Load the JDBC driver by invoking the Class.forName method.  
If you are using JDBC 4.0, you do not need to explicitly load the JDBC driver. For the IBM Data Server Driver for JDBC and SQLJ, you load the driver by invoking the Class.forName method with the following argument:

```
com.ibm.db2.jcc.DB2Driver
```

For compatibility with previous JDBC drivers, you can use the following argument instead:

```
COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver
```

The following code demonstrates loading the IBM Data Server Driver for JDBC and SQLJ:

```
try {
 // Load the IBM Data Server Driver for JDBC and SQLJ with DriverManager
 Class.forName("com.ibm.db2.jcc.DB2Driver");
} catch (ClassNotFoundException e) {
 e.printStackTrace();
}
```

The catch block is used to print an error if the driver is not found.

2. Connect to a data source by invoking the DriverManager.getConnection method.

You can use one of the following forms of getConnection:

```
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

For IBM Data Server Driver for JDBC and SQLJ type 4 connectivity, the `getConnection` method must specify a user ID and password, through parameters or through property values.

The `url` argument represents a data source, and indicates what type of JDBC connectivity you are using.

The `info` argument is an object of type `java.util.Properties` that contains a set of driver properties for the connection. Specifying the `info` argument is an alternative to specifying `property=value` strings in the URL. See "Properties for the IBM Data Server Driver for JDBC and SQLJ" for the properties that you can specify.

There are several ways to specify a user ID and password for a connection:

- Use the form of the `getConnection` method that specifies `url` with `property=value` clauses, and include the user and password properties in the URL.
- Use the form of the `getConnection` method that specifies `user` and `password`.
- Use the form of the `getConnection` method that specifies `info`, after setting the user and password properties in a `java.util.Properties` object.

*Example: Establishing a connection and setting the user ID and password in a URL:*

```
String url = "jdbc:db2://myhost:5021/mydb:" +
 "user=dbadm;password=dbadm;";

// Set URL for data source
Connection con = DriverManager.getConnection(url);
// Create connection
```

*Example: Establishing a connection and setting the user ID and password in user and password parameters:*

```
String url = "jdbc:db2://myhost:5021/mydb";
// Set URL for data source
String user = "dbadm";
String password = "dbadm";
Connection con = DriverManager.getConnection(url, user, password);
// Create connection
```

*Example: Establishing a connection and setting the user ID and password in a java.util.Properties object:*

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "dbadm"); // Set user ID for connection
properties.put("password", "dbadm"); // Set password for connection
String url = "jdbc:db2://myhost:5021/mydb";
// Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
// Create connection
```





---

## Chapter 5. Security and Routines

---

### Benefits of using routines

The following benefits can be gained by using routines:

#### **Encapsulate application logic that can be invoked from an SQL interface**

In an environment containing many different client applications that have common requirements, the effective use of routines can simplify code reuse, code standardization, and code maintenance. If a particular aspect of common application behavior needs to be changed in an environment where routines are used, only the affected routine that encapsulates the behavior requires modification. Without routines, application logic changes are required in each application.

#### **Enable controlled access to other database objects**

Routines can be used to control access to database objects. A user might not have permission to generally issue a particular SQL statement, such as CREATE TABLE; however the user can be given permission to invoke routines that contain one or more specific implementations of the statement, thus simplifying privilege management through encapsulation of privileges.

#### **Improve application performance by reducing network traffic**

When applications run on a client computer, each SQL statement is sent separately from the client computer to the database server computer to be executed and each result set is returned separately. This can result in high levels of network traffic. If a piece of work can be identified that requires extensive database interaction and little user interaction, it makes sense to install this piece of work on the server to minimize the quantity of network traffic and to allow the work to be done on the more powerful database servers.

#### **Allow for faster, more efficient SQL execution**

Because routines are database objects, they are more efficient at transmitting SQL requests and data than client applications. Therefore, SQL statements executed within routines can perform better than if executed in client applications. Routines that are created with the NOT FENCED clause run in the same process as the database manager, and can therefore use shared memory for communication, which can result in improved application performance.

#### **Allow the interoperability of logic implemented in different programming languages**

Because code modules might be implemented by different programmers in different programming languages, and because it is generally desirable to reuse code when possible, DB2 routines support a high degree of interoperability.

- Client applications in one programming language can invoke routines that are implemented in a different programming language. For example C client applications can invoke .NET common language runtime routines.
- Routines can invoke other routines regardless of the routine type or routine implementation. For example a Java procedure can invoke an embedded SQL scalar function.

- Routines created in a database server on one operating system can be invoked from a DB2 client running on a different operating system.

The benefits described above are just some of the many benefits of using routines. Using routines can be beneficial to a variety of users including database administrators, database architects, and database application developers. For this reason there are many useful applications of routines that you might want to explore.

There are various kinds of routines that address particular functional needs and various routine implementations. The choice of routine type and implementation can impact the degree to which the above benefits are exhibited. In general, routines are a powerful way of encapsulating logic so that you can extend your SQL, and improve the structure, maintenance, and potentially the performance of your applications.

---

## External scalar functions

External scalar functions are scalar functions that have their logic implemented in an external programming language.

These functions can be developed and used to extend the set of existing SQL functions and can be invoked in the same manner as DB2 built-in functions such as LENGTH and COUNT. That is, they can be referenced in SQL statements wherever an expression is valid.

The execution of external scalar function logic takes place on the DB2 database server, however unlike built-in or user-defined SQL scalar functions, the logic of external functions can access the database server filesystem, perform system calls or access a network.

External scalar functions can read SQL data, but cannot modify SQL data.

External scalar functions can be repeatedly invoked for a single reference of the function and can maintain state between these invocations by using a scratchpad, which is a memory buffer. This can be powerful if a function requires some initial, but expensive, setup logic. The setup logic can be done on a first invocation using the scratchpad to store some values that can be accessed or updated in subsequent invocations of the scalar function.

### Features of external scalar functions

- Can be referenced as part of an SQL statement anywhere an expression is supported.
- The output of a scalar function can be used directly by the invoking SQL statement.
- For external scalar user-defined functions, state can be maintained between the iterative invocations of the function by using a scratchpad.
- Can provide a performance advantage when used in predicates, because they are executed at the server. If a function can be applied to a candidate row at the server, it can often eliminate the row from consideration before transmitting it to the client machine, reducing the amount of data that must be passed from server to client.

### Limitations

- Cannot do transaction management within a scalar function. That is, you cannot issue a COMMIT or a ROLLBACK within a scalar function.

- Cannot return result sets.
- Scalar functions are intended to return a single scalar value per set of inputs.
- External scalar functions are not intended to be used for a single invocation. They are designed such that for a single reference to the function and a given set of inputs, that the function be invoked once per input, and return a single scalar value. On the first invocation, scalar functions can be designed to do some setup work, or store some information that can be accessed in subsequent invocations. SQL scalar functions are better suited to functionality that requires a single invocation.

- 

In a single partition database external scalar functions can contain SQL statements. These statements can read data from tables, but cannot modify data in tables. If the database has more than one partition then there must be no SQL statements in an external scalar function. SQL scalar functions can contain SQL statements that read or modify data.

#### Common uses

- Extend the set of DB2 built-in functions.
- Perform logic inside an SQL statement that SQL cannot natively perform.
- Encapsulate a scalar query that is commonly reused as a subquery in SQL statements. For example, given a postal code, search a table for the city where the postal code is found.

#### Supported languages

- C
- C++
- Java
- OLE
- .NET common language runtime languages

#### Note:

1. There is a limited capability for creating aggregate functions. Also known as column functions, these functions receive a set of like values (a column of data) and return a single answer. A user-defined aggregate function can only be created if it is sourced upon a built-in aggregate function. For example, if a distinct type SHOESIZE exists that is defined with base type INTEGER, you could define a function, `AVG(SHOESIZE)`, as an aggregate function sourced on the existing built-in aggregate function, `AVG(INTEGER)`.
2. You can also create function that return a row. These are known as row functions and can only be used as a transform function for structured types. The output of a row function is a single row.

---

## Methods

Methods enable you to define behaviors for structured types. They are like scalar UDFs, but can only be defined for structured types.

Methods share all the features of scalar UDFs, in addition to the following features:

#### Features

- Strongly associated with the structured type.

- Can be sensitive to the dynamic type of the subject type.

#### Limitations

- Can only return a scalar value.
- Can only be used with structured types.
- Cannot be invoked against typed tables.

#### Common uses

- Providing operations on structured types.
- Encapsulating the structured type.

#### Supported languages

- SQL
- C/C++
- Java™
- OLE

---

## Security considerations for routines

Developing and deploying routines provides you with an opportunity to greatly improve the performance and effectiveness of your database applications. There can, however, be security risks if the deployment of routines is not managed correctly by the database administrator. The following sections describe security risks and means by which you can mitigate these risks. The security risks are followed by a section on how to safely deploy routines whose security is unknown.

### Security risks

#### NOT FENCED routines can access database manager resources

NOT FENCED routines run in the same process as the database manager. Because of their close proximity to the database engine, NOT FENCED routines can accidentally or maliciously corrupt the database manager's shared memory, or damage the database control structures. Either form of damage will cause the database manager to fail. NOT FENCED routines can also corrupt databases and their tables.

To ensure the integrity of the database manager and its databases, you must thoroughly screen routines you intend to register as NOT FENCED. These routines must be fully tested, debugged, and exhibit no unexpected side-effects. In the examination of the routine, pay close attention to memory management and the use of static variables. The greatest potential for corruption arises when code does not properly manage memory or incorrectly uses static variables. These problems are prevalent in languages other than Java and .NET programming languages.

In order to register a NOT FENCED routine, the CREATE\_NOT\_FENCED\_ROUTINE authority is required. When granting the CREATE\_NOT\_FENCED\_ROUTINE authority, be aware that the recipient can potentially gain unrestricted access to the database manager and all its resources.

**Note:** NOT FENCED routines are not supported in Common Criteria compliant configurations.

### **FENCED THREADSAFE routines can access memory in other FENCED THREADSAFE routines**

FENCED THREADSAFE routines run as threads inside a shared process. Each of these routines are able to read the memory used by other routine threads in the same process. Therefore, it is possible for one threaded routine to collect sensitive data from other routines in the threaded process. Another risk inherent in the sharing of a single process, is that one routine thread with flawed memory management can corrupt other routine threads, or cause the entire threaded process to crash.

To ensure the integrity of other FENCED THREADSAFE routines, you must thoroughly screen routines you intend to register as FENCED THREADSAFE. These routines must be fully tested, debugged, and exhibit no unexpected side-effects. In the examination of the routine, pay close attention to memory management and the use of static variables. This is where the greatest potential for corruption exists, particularly in languages other than Java.

In order to register a FENCED THREADSAFE routine, the `CREATE_EXTERNAL_ROUTINE` authority is required. When granting the `CREATE_EXTERNAL_ROUTINE` authority, be aware that the recipient can potentially monitor or corrupt the memory of other FENCED THREADSAFE routines.

### **Write access to the database server by the owner of fenced processes can result in database manager corruption**

The user ID under which fenced processes run is defined by the `db2icrt` (create instance) or `db2iupdt` (update instance) system commands. This user ID must not have write access to the directory where routine libraries and classes are stored (in UNIX environments, `sqllib/function`; in Windows environments, `sqllib\function`). This user ID must also not have read or write access to any database, operating system, or otherwise critical files and directories on the database server.

If the owner of fenced processes does have write access to various critical resources on the database server, the potential for system corruption exists. For example, a database administrator registers a routine received from an unknown source as FENCED NOT THREADSAFE, thinking that any potential harm can be averted by isolating the routine in its own process. However, the user ID that owns fenced processes has write access to the `sqllib/function` directory. Users invoke this routine, and unbeknownst to them, it overwrites a library in `sqllib/function` with an alternate version of a routine body that is registered as NOT FENCED. This second routine has unrestricted access to the entire database manager, and can thereby distribute sensitive information from database tables, corrupt the databases, collect authentication information, or crash the database manager.

Ensure the user ID that owns fenced processes does not have write access to critical files or directories on the database server (especially `sqllib/function` and the database data directories).

### **Vulnerability of routine libraries and classes**

If access to the directory where routine libraries and classes are stored is not controlled, routine libraries and classes can be deleted or overwritten. As discussed in the previous item, the replacement of a NOT FENCED routine body with a malicious (or poorly coded) routine can severely compromise the stability, integrity, and privacy of the database server and its resources.

To protect the integrity of routines, you must manage access to the directory containing the routine libraries and classes. Ensure that the fewest possible number of users can access this directory and its files. When assigning write access to this directory, be aware that this privilege can provide the owner of the user ID unrestricted access to the database manager and all its resources.

## Deploying potentially insecure routines

If you happen to acquire a routine from an unknown source, be sure you know exactly what it does before you build, register, and invoke it. It is recommended that you register it as FENCED and NOT THREADSAFE unless you have tested it thoroughly, and it exhibits no unexpected side-effects.

If you need to deploy a routine that does not meet the criteria for secure routines, register the routine as FENCED and NOT THREADSAFE. To ensure that database integrity is maintained, FENCED and NOT THREADSAFE routines:

- Run in a separate DB2 process, shared with no other routines. If they abnormally terminate, the database manager will be unaffected.
- Use memory that is distinct from memory used by the database. An inadvertent mistake in a value assignment will not affect the database manager.

---

## Connection contexts in SQLJ routines

With the introduction of multithreaded routines in DB2 Universal Database, Version 8, it is important that SQLJ routines avoid the use of the default connection context. That is, each SQL statement must explicitly indicate the ConnectionContext object, and that context must be explicitly instantiated in the Java method. For instance, in previous releases of DB2, a SQLJ routine could be written as follows:

```
class myClass
{
 public static void myRoutine(short myInput)
 {
 DefaultContext ctx = DefaultContext.getDefaultContext();
 #sql { some SQL statement };
 }
}
```

This use of the default context causes all threads in a multithreaded environment to use the same connection context, which, in turn, will result in unexpected failures.

The SQLJ routine above must be changed as follows:

```
#context MyContext;

class myClass
{
 public static void myRoutine(short myInput)
 {
 MyContext ctx = new MyContext("jdbc:default:connection", false);
 #sql [ctx] { some SQL statement };
 ctx.close();
 }
}
```

This way, each invocation of the routine will create its own unique ConnectionContext (and underlying JDBC connection), which avoids unexpected interference by concurrent threads.

---

## External routine library and class management

To successfully develop and invoke external routines, external routine library and class files must be deployed and managed properly.

External routine library and class file management can be minimal if care is taken when external routines are first created and library and class files deployed.

The main external routine management considerations are the following:

- Deployment of external routine library and class files
- Security of external routine library and class files
- Resolution of external routine libraries and classes
- Modifications to external routine library and class files
- Backup and restore of external routine library and class files

System administrators, database administrators and database application developers should all take responsibility to ensure that external routine library and class files are secure and correctly preserved during routine development and database administration tasks.

---

## Rebuilding DB2 routine shared libraries

DB2 will cache the shared libraries used for stored procedures and user-defined functions once loaded. If you are developing a routine, you might want to test loading the same shared library a number of times, and this caching can prevent you from picking up the latest version of a shared library. The way to avoid caching problems depends on the type of routine:

1. **Fenced, not threadsafe routines.** The database manager configuration keyword `KEEPFENCED` has a default value of `YES`. This keeps the fenced mode process alive. This default setting can interfere with reloading the library. It is best to change the value of this keyword to `NO` while developing fenced, not threadsafe routines, and then change it back to `YES` when you are ready to load the final version of your shared library. For more information, see “Updating the database manager configuration file.”
2. **Trusted or threadsafe routines.** Except for SQL routines (including SQL procedures), the only way to ensure that an updated version of a DB2 routine library is picked up when that library is used for trusted, or threadsafe routines, is to recycle the DB2 instance by entering `db2stop` followed by `db2start` on the command line. This is not needed for an SQL routine because when it is recreated, the compiler uses a new unique library name to prevent possible conflicts.

For routines other than SQL routines, you can also avoid caching problems by creating the new version of the routine with a differently named library (for example `foo.a` becomes `foo.1.a`), and then using either the `ALTER PROCEDURE` or `ALTER FUNCTION SQL` statement with the new library.

---

## Updating the database manager configuration file

This file contains important settings for application development.



The keyword `KEEPFENCED` has the default value `YES`. For fenced, not threadsafe routines (stored procedures and UDFs), this keeps the routine process alive. It is best to change the value of this keyword to `NO` while developing these routines, and then change it back to `YES` when you are ready to load the final version of your shared library. For more information, see “Rebuilding DB2 routine shared libraries” on page 547.

**Note:** `KEEPFENCED` was known as `KEEPDARI` in previous versions of DB2.

For Java application development, you need to update the `JDK_PATH` keyword with the path where the Java Development Kit is installed.

**Note:** `JDK_PATH` was known as `JDK11_PATH` in previous versions of DB2.

To change these settings enter:

```
db2 update dbm cfg using <keyword> <value>
```

For example, to set the keyword `KEEPFENCED` to `NO`:

```
db2 update dbm cfg using KEEPFENCED NO
```

To set the `JDK_PATH` keyword to the directory `/home/db2inst/jdk13`:

```
db2 update dbm cfg using JDK_PATH /home/db2inst/jdk13
```

To view the current settings in the database manager configuration file, enter:

```
db2 get dbm cfg
```

**Note:** On Windows, you need to enter these commands in a DB2 command window.

---

## Chapter 6. SQLCA (SQL communications area)

An SQLCA is a collection of variables that is updated at the end of the execution of every SQL statement. A program that contains executable SQL statements and is precompiled with option LANGLEVEL SAA1 (the default) or MIA must provide exactly one SQLCA, though more than one SQLCA is possible by having one SQLCA per thread in a multi-threaded application.

When a program is precompiled with option LANGLEVEL SQL92E, an SQLCODE or SQLSTATE variable may be declared in the SQL declare section or an SQLCODE variable can be declared somewhere in the program.

An SQLCA should not be provided when using LANGLEVEL SQL92E. The SQL INCLUDE statement can be used to provide the declaration of the SQLCA in all languages but REXX. The SQLCA is automatically provided in REXX.

To display the SQLCA after each command executed through the command line processor, issue the command `db2 -a`. The SQLCA is then provided as part of the output for subsequent commands. The SQLCA is also dumped in the `db2diag` log file.

### SQLCA field descriptions

*Table 24. Fields of the SQLCA.* The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

| Name     | Data Type | Field Values                                                                                                                                                                                                                                                                      |
|----------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqlcaid  | CHAR(8)   | An "eye catcher" for storage dumps containing 'SQLCA'. The sixth byte is 'L' if line number information is returned from parsing an SQL procedure body.                                                                                                                           |
| sqlcab   | INTEGER   | Contains the length of the SQLCA, 136.                                                                                                                                                                                                                                            |
| sqlcode  | INTEGER   | Contains the SQL return code.<br><br><b>Code</b> <b>Means</b><br><br><b>0</b> Successful execution (although one or more SQLWARN indicators may be set).<br><br><b>positive</b><br>Successful execution, but with a warning condition.<br><br><b>negative</b><br>Error condition. |
| sqlerrml | SMALLINT  | Length indicator for <i>sqlerrmc</i> , in the range 0 through 70. 0 means that the value of <i>sqlerrmc</i> is not relevant.                                                                                                                                                      |

Table 24. Fields of the SQLCA (continued). The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

| Name       | Data Type       | Field Values                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqlerrmc   | VARCHAR<br>(70) | <p>Contains one or more tokens, separated by X'FF', which are substituted for variables in the descriptions of error conditions.</p> <p>This field is also used when a successful connection is completed.</p> <p>When a NOT ATOMIC compound SQL statement is issued, it may contain information on up to seven errors.</p> <p>The last token might be followed by X'FF'. The <i>sqlerrml</i> value will include any trailing X'FF'.</p>                                                                                                                                                                 |
| sqlerrp    | CHAR(8)         | <p>Begins with a three-letter identifier indicating the product, followed by five alphanumeric characters indicating the version, release, and modification level of the product. The characters A-Z indicate a modification level higher than 9. A indicates modification level 10, B indicates modification level 11, and so on. For example, SQL0907C means DB2 Version 9, release 7, modification level 12).</p> <p>If SQLCODE indicates an error condition, this field identifies the module that returned the error.</p> <p>This field is also used when a successful connection is completed.</p> |
| sqlerrd    | ARRAY           | <p>Six INTEGER variables that provide diagnostic information. These values are generally empty if there are no errors, except for sqlerrd(6) from a partitioned database.</p>                                                                                                                                                                                                                                                                                                                                                                                                                            |
| sqlerrd(1) | INTEGER         | <p>If connection is invoked and successful, contains the maximum expected difference in length of mixed character data (CHAR data types) when converted to the database code page from the application code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction.</p> <p>On successful return from an SQL procedure, contains the return status value from the SQL procedure.</p>                                                                                                          |
| sqlerrd(2) | INTEGER         | <p>If connection is invoked and successful, contains the maximum expected difference in length of mixed character data (CHAR data types) when converted to the application code page from the database code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction. If the SQLCA results from a NOT ATOMIC compound SQL statement that encountered one or more errors, the value is set to the number of statements that failed.</p>                                                         |

Table 24. Fields of the SQLCA (continued). The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

| Name       | Data Type | Field Values                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqlerrd(3) | INTEGER   | <p>If PREPARE is invoked and successful, contains an estimate of the number of rows that will be returned. After INSERT, UPDATE, DELETE, or MERGE, contains the actual number of rows that qualified for the operation. For a TRUNCATE statement, the value will be -1. If compound SQL is invoked, contains an accumulation of all sub-statement rows. If CONNECT is invoked, contains 1 if the database can be updated, or 2 if the database is read only.</p> <p>If the OPEN statement is invoked, and the cursor contains SQL data change statements, this field contains the sum of the number of rows that qualified for the embedded insert, update, delete, or merge operations.</p> <p>If CREATE PROCEDURE for an SQL procedure is invoked, and an error is encountered when parsing the SQL procedure body, contains the line number where the error was encountered. The sixth byte of sqlcaid must be 'L' for this to be a valid line number.</p> |
| sqlerrd(4) | INTEGER   | <p>If PREPARE is invoked and successful, contains a relative cost estimate of the resources required to process the statement. If compound SQL is invoked, contains a count of the number of successful sub-statements. If CONNECT is invoked, contains 0 for a one-phase commit from a down-level client; 1 for a one-phase commit; 2 for a one-phase, read-only commit; and 3 for a two-phase commit.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| sqlerrd(5) | INTEGER   | <p>Contains the total number of rows deleted, inserted, or updated as a result of both:</p> <ul style="list-style-type: none"> <li>• The enforcement of constraints after a successful delete operation</li> <li>• The processing of triggered SQL statements from activated triggers</li> </ul> <p>If compound SQL is invoked, contains an accumulation of the number of such rows for all sub-statements. In some cases, when an error is encountered, this field contains a negative value that is an internal error pointer. If CONNECT is invoked, contains an authentication type value of 0 for server authentication; 1 for client authentication; 2 for authentication using DB2 Connect; 4 for SERVER_ENCRYPT authentication; 5 for authentication using DB2 Connect with encryption; 7 for KERBEROS authentication; 9 for GSSPLUGIN authentication; 11 for DATA_ENCRYPT authentication; and 255 for unspecified authentication.</p>                |
| sqlerrd(6) | INTEGER   | <p>For a partitioned database, contains the partition number of the database partition that encountered the error or warning. If no errors or warnings were encountered, this field contains the partition number of the coordinator partition. The number in this field is the same as that specified for the database partition in the db2nodes.cfg file.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

Table 24. Fields of the SQLCA (continued). The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

| Name      | Data Type | Field Values                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqlwarn   | Array     | A set of warning indicators, each containing a blank or W. If compound SQL is invoked, contains an accumulation of the warning indicators set for all sub-statements.                                                                                                                                                                                                                                                                                                                                                                                        |
| sqlwarn0  | CHAR(1)   | Blank if all other indicators are blank; contains 'W' if at least one other indicator is not blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| sqlwarn1  | CHAR(1)   | Contains 'W' if the value of a string column was truncated when assigned to a host variable. Contains 'N' if the null terminator was truncated. Contains 'A' if the CONNECT or ATTACH is successful, and the authorization name for the connection is longer than 8 bytes. Contains 'P' if the PREPARE statement relative cost estimate stored in sqlerrd(4) exceeded the value that could be stored in an INTEGER or was less than 1, and either the CURRENT EXPLAIN MODE or the CURRENT EXPLAIN SNAPSHOT special register is set to a value other than NO. |
| sqlwarn2  | CHAR(1)   | Contains 'W' if null values were eliminated from the argument of an aggregate function. <sup>a</sup><br><br>If CONNECT is invoked and successful, contains 'D' if the database is in quiesce state, or 'I' if the instance is in quiesce state.                                                                                                                                                                                                                                                                                                              |
| sqlwarn3  | CHAR(1)   | Contains 'W' if the number of columns is not equal to the number of host variables. Contains 'Z' if the number of result set locators specified on the ASSOCIATE LOCATORS statement is less than the number of result sets returned by a procedure.                                                                                                                                                                                                                                                                                                          |
| sqlwarn4  | CHAR(1)   | Contains 'W' if a prepared UPDATE or DELETE statement does not include a WHERE clause.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| sqlwarn5  | CHAR(1)   | Contains 'E' if an error was tolerated during SQL statement execution.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| sqlwarn6  | CHAR(1)   | Contains 'W' if the result of a date calculation was adjusted to avoid an impossible date.                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| sqlwarn7  | CHAR(1)   | Reserved for future use.<br><br>If CONNECT is invoked and successful, contains 'E' if the <b>dyn_query_mgmt</b> database configuration parameter is enabled.                                                                                                                                                                                                                                                                                                                                                                                                 |
| sqlwarn8  | CHAR(1)   | Contains 'W' if a character that could not be converted was replaced with a substitution character. Contains 'Y' if there was an unsuccessful attempt to establish a trusted connection.                                                                                                                                                                                                                                                                                                                                                                     |
| sqlwarn9  | CHAR(1)   | Contains 'W' if arithmetic expressions with errors were ignored during aggregate function processing.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| sqlwarn10 | CHAR(1)   | Contains 'W' if there was a conversion error when converting a character data value in one of the fields in the SQLCA.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| sqlstate  | CHAR(5)   | A return code that indicates the outcome of the most recently executed SQL statement.                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

<sup>a</sup> Some functions may not set SQLWARN2 to W, even though null values were eliminated, because the result was not dependent on the elimination of null values.

## Error reporting

The order of error reporting is as follows:

1. Severe error conditions are always reported. When a severe error is reported, there are no additions to the SQLCA.
2. If no severe error occurs, a deadlock error takes precedence over other errors.
3. For all other errors, the SQLCA for the first negative SQL code is returned.
4. If no negative SQL codes are detected, the SQLCA for the first warning (that is, positive SQL code) is returned.

In a partitioned database system, the exception to this rule occurs if a data manipulation operation is invoked against a table that is empty on one database partition, but has data on other database partitions. SQLCODE +100 is only returned to the application if agents from all database partitions return SQL0100W, either because the table is empty on all database partitions, or there are no more rows that satisfy the WHERE clause in an UPDATE statement.

## SQLCA usage in partitioned database systems

In partitioned database systems, one SQL statement may be executed by a number of agents on different database partitions, and each agent may return a different SQLCA for different errors or warnings. The coordinator agent also has its own SQLCA.

To provide a consistent view for applications, all SQLCA values are merged into one structure, and SQLCA fields indicate global counts, such that:

- For all errors and warnings, the *sqlwarn* field contains the warning flags received from all agents.
- Values in the *sqlerrd* fields indicating row counts are accumulations from all agents.

Note that SQLSTATE 09000 may not be returned every time an error occurs during the processing of a triggered SQL statement.





---

## Chapter 7. SQLDA (SQL descriptor area)

An SQLDA is a collection of variables that is required for execution of the SQL DESCRIBE statement. The SQLDA variables are options that can be used by the PREPARE, OPEN, FETCH, and EXECUTE statements. An SQLDA communicates with dynamic SQL; it can be used in a DESCRIBE statement, modified with the addresses of host variables, and then reused in a FETCH or EXECUTE statement.

SQLDAs are supported for all languages, but predefined declarations are provided only for C, REXX, FORTRAN, and COBOL.

The meaning of the information in an SQLDA depends on its use. In PREPARE and DESCRIBE, an SQLDA provides information to an application program about a prepared statement. In OPEN, EXECUTE, and FETCH, an SQLDA describes host variables.

In DESCRIBE and PREPARE, if any one of the columns being described is either a LOB type (LOB locators and file reference variables do not require doubled SQLDAs), reference type, or a user-defined type, the number of SQLVAR entries for the entire SQLDA will be doubled. For example:

- When describing a table with 3 VARCHAR columns and 1 INTEGER column, there will be 4 SQLVAR entries
- When describing a table with 2 VARCHAR columns, 1 CLOB column, and 1 integer column, there will be 8 SQLVAR entries

In EXECUTE, FETCH, and OPEN, if any one of the variables being described is a LOB type (LOB locators and file reference variables do not require doubled SQLDAs) or a structured type, the number of SQLVAR entries for the entire SQLDA must be doubled. (Distinct types and reference types are not relevant in these cases, because the additional information in the double entries is not required by the database. Array, cursor and row types are not supported as SQLDA variables in EXECUTE, FETCH and OPEN statements.)

### SQLDA field descriptions

An SQLDA consists of four variables followed by an arbitrary number of occurrences of a sequence of variables collectively named SQLVAR. In OPEN, FETCH, and EXECUTE, each occurrence of SQLVAR describes a host variable. In DESCRIBE and PREPARE, each occurrence of SQLVAR describes a column of a result table or a parameter marker. There are two types of SQLVAR entries:

- **Base SQLVARs:** These entries are always present. They contain the base information about the column, parameter marker, or host variable such as data type code, length attribute, column name, host variable address, and indicator variable address.
- **Secondary SQLVARs:** These entries are only present if the number of SQLVAR entries is doubled as per the rules outlined above. For user-defined types (excluding reference types), they contain the user-defined type name. For reference types, they contain the target type of the reference. For LOBs, they contain the length attribute of the host variable and a pointer to the buffer that contains the actual length. (The distinct type and LOB information does not overlap, so distinct types can be based on LOBs without forcing the number of

SQLVAR entries on a DESCRIBE to be tripled.) If locators or file reference variables are used to represent LOBs, these entries are not necessary.

In SQLDAs that contain both types of entries, the base SQLVARs are in a block before the block of secondary SQLVARs. In each, the number of entries is equal to the value in SQLD (even though many of the secondary SQLVAR entries may be unused).

The circumstances under which the SQLVAR entries are set by DESCRIBE is detailed in "Effect of DESCRIBE on the SQLDA" on page 560.

## Fields in the SQLDA header

Table 25. Fields in the SQLDA Header

| C Name  | SQL Data Type | Usage in DESCRIBE and PREPARE (set by the database manager except for SQLN)                                                                                                                                                                                                                                                                   | Usage in FETCH, OPEN, and EXECUTE (set by the application prior to executing the statement)                                                                                                                                                                                                                                        |
|---------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqldaid | CHAR(8)       | The seventh byte of this field is a flag byte named SQLDOUBLED. The database manager sets SQLDOUBLED to the character '2' if two SQLVAR entries have been created for each column; otherwise it is set to a blank (X'20' in ASCII, X'40' in EBCDIC). See "Effect of DESCRIBE on the SQLDA" on page 560 for details on when SQLDOUBLED is set. | The seventh byte of this field is used when the number of SQLVARs is doubled. It is named SQLDOUBLED. If any of the host variables being described is a structured type, BLOB, CLOB, or DBCLOB, the seventh byte must be set to the character '2'; otherwise it can be set to any character but the use of a blank is recommended. |
| sqldabc | INTEGER       | For 32 bit, the length of the SQLDA, equal to SQLN*44+16. For 64 bit, the length of the SQLDA, equal to SQLN*56+16                                                                                                                                                                                                                            | For 32 bit, the length of the SQLDA, >= to SQLN*44+16. For 64 bit, the length of the SQLDA, >= to SQLN*56+16.                                                                                                                                                                                                                      |
| sqln    | SMALLINT      | Unchanged by the database manager. Must be set to a value greater than or equal to zero before the DESCRIBE statement is executed. Indicates the total number of occurrences of SQLVAR.                                                                                                                                                       | Total number of occurrences of SQLVAR provided in the SQLDA. SQLN must be set to a value greater than or equal to zero.                                                                                                                                                                                                            |
| sqld    | SMALLINT      | Set by the database manager to the number of columns in the result table or to the number of parameter markers.                                                                                                                                                                                                                               | The number of host variables described by occurrences of SQLVAR.                                                                                                                                                                                                                                                                   |

## Fields in an occurrence of a base SQLVAR

Table 26. Fields in a Base SQLVAR

| Name    | Data Type | Usage in DESCRIBE and PREPARE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Usage in FETCH, OPEN, and EXECUTE                                                                                                                                                                                                        |
|---------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqltype | SMALLINT  | Indicates the data type of the column or parameter marker, and whether it can contain nulls. (Parameter markers are always considered nullable.) Table 28 on page 561 lists the allowable values and their meanings.                                                                                                                                                                                                                                                                                                                                | Same for host variable. Host variables for datetime values must be character string variables. For FETCH, a datetime type code means a fixed-length character string. If sqltype is an even number value, the sqlind field is ignored.   |
|         |           | Note that for a distinct, array, cursor, row, or reference type, the data type of the base type is placed into this field. For a structured type, the data type of the result of the FROM SQL transform function of the transform group (based on the CURRENT DEFAULT TRANSFORM GROUP special register) for the type is placed into this field. There is no indication in the base SQLVAR that it is part of the description of a user-defined type or reference type.                                                                              |                                                                                                                                                                                                                                          |
| sqllen  | SMALLINT  | The length attribute of the column or parameter marker. For datetime columns and parameter markers, the length of the string representation of the values. See Table 28 on page 561.<br><br>Note that the value is set to 0 for large object strings (even for those whose length attribute is small enough to fit into a two byte integer).                                                                                                                                                                                                        | The length attribute of the host variable. See Table 28 on page 561.<br><br>Note that the value is ignored by the database manager for CLOB, DBCLOB, and BLOB columns. The len.sqllonglen field in the Secondary SQLVAR is used instead. |
| sqldata | pointer   | For string SQLVARs, sqldata contains the code page. For character-string SQLVARs where the column is defined with the FOR BIT DATA attribute, sqldata contains 0. For other character-string SQLVARs, sqldata contains either the SBCS code page for SBCS data, or the SBCS code page associated with the composite MBCS code page for MBCS data. For Japanese EUC, Traditional Chinese EUC, and Unicode UTF-8 character-string SQLVARs, sqldata contains 954, 964, and 1208 respectively.<br><br>For all other column types, sqldata is undefined. | Contains the address of the host variable (where the fetched data will be stored).                                                                                                                                                       |
| sqlind  | pointer   | For character-string SQLVARs, sqlind contains 0, except for MBCS data, when sqlind contains the DBCS code page associated with the composite MBCS code page.<br><br>For all other types, sqlind is undefined.                                                                                                                                                                                                                                                                                                                                       | Contains the address of an associated indicator variable, if there is one; otherwise, not used. If sqltype is an even number value, the sqlind field is ignored.                                                                         |

Table 26. Fields in a Base SQLVAR (continued)

| Name    | Data Type       | Usage in DESCRIBE and PREPARE                                                                                                                                                                                                                            | Usage in FETCH, OPEN, and EXECUTE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqlname | VARCHAR<br>(30) | <p>Contains the unqualified name of the column or parameter marker.</p> <p>For columns and parameter markers that have a system-generated name, the thirtieth byte is set to X'FF'. For column names specified by the AS clause, this byte is X'00'.</p> | <p>When connecting to a host database, sqlname can be set to indicate a FOR BIT DATA string as follows:</p> <ul style="list-style-type: none"> <li>• The sixth byte of the SQLDAID in the SQLDA header is set to '+'</li> <li>• The length of sqlname is 8</li> <li>• The first two bytes of sqlname are X'0000'</li> <li>• The third and fourth bytes of sqlname are X'0000'</li> <li>• The remaining four bytes of sqlname are reserved and should be set to X'00000000'</li> </ul> <p>When working with XML data, sqlname can be set to indicate an XML subtype as follows:</p> <ul style="list-style-type: none"> <li>• The length of sqlname is 8</li> <li>• The first two bytes of sqlname are X'0000'</li> <li>• The third and fourth bytes of sqlname are X'0000'</li> <li>• The fifth byte of sqlname is X'01'</li> <li>• The remaining three bytes of sqlname are reserved and should be set to X'000000'</li> </ul> |

### Fields in an occurrence of a secondary SQLVAR

Table 27. Fields in a Secondary SQLVAR

| Name           | Data Type                                    | Usage in DESCRIBE and PREPARE                                               | Usage in FETCH, OPEN, and EXECUTE                                                                                                                                                                                                                                              |
|----------------|----------------------------------------------|-----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| len.sqllonglen | INTEGER                                      | The length attribute of a BLOB, CLOB, or DBCLOB column or parameter marker. | The length attribute of a BLOB, CLOB, or DBCLOB host variable. The database manager ignores the SQLLEN field in the Base SQLVAR for the data types. The length attribute stores the number of bytes for a BLOB or CLOB, and the number of double-byte characters for a DBCLOB. |
| reserve2       | CHAR(3) for 32 bit, and CHAR(11) for 64 bit. | Not used.                                                                   | Not used.                                                                                                                                                                                                                                                                      |

Table 27. Fields in a Secondary SQLVAR (continued)

| Name             | Data Type   | Usage in DESCRIBE and PREPARE                                                                                                                                                                                                                                 | Usage in FETCH, OPEN, and EXECUTE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqlflag4         | CHAR(1)     | The value is X'01' if the SQLVAR represents a reference type with a target type named in sqldatatype_name. The value is X'12' if the SQLVAR represents a structured type, with the user-defined type name in sqldatatype_name. Otherwise, the value is X'00'. | Set to X'01' if the SQLVAR represents a reference type with a target type named in sqldatatype_name. Set to X'12' if the SQLVAR represents a structured type, with the user-defined type name in sqldatatype_name. Otherwise, the value is X'00'.                                                                                                                                                                                                                                                                                                                                                                    |
| sqldatalen       | pointer     | Not used.                                                                                                                                                                                                                                                     | Used for BLOB, CLOB, and DBCLOB host variables only.<br><br>If this field is NULL, then the actual length (in double-byte characters) should be stored in the 4 bytes immediately before the start of the data and SQLDATA should point to the first byte of the field length.<br><br>If this field is not NULL, it contains a pointer to a 4 byte long buffer that contains the actual length <i>in bytes</i> (even for DBCLOB) of the data in the buffer pointed to from the SQLDATA field in the matching base SQLVAR.<br><br>Note that, whether or not this field is used, the len.sqllonglen field must be set. |
| sqldatatype_name | VARCHAR(27) | For a user-defined type, the database manager sets this to the fully qualified user-defined type name. <sup>1</sup> For a reference type, the database manager sets this to the fully qualified type name of the target type of the reference.                | For structured types, set to the fully qualified user-defined type name in the format indicated in the table note. <sup>1</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| reserved         | CHAR(3)     | Not used.                                                                                                                                                                                                                                                     | Not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

<sup>1</sup> The first 8 bytes contain the schema name of the type (extended to the right with spaces, if necessary). Byte 9 contains a dot (.). Bytes 10 to 27 contain the low order portion of the type name, which is *not* extended to the right with spaces.

Note that, although the prime purpose of this field is for the name of user-defined types, the field is also set for IBM predefined data types. In this case, the schema name is SYSIBM, and the low order portion of the name is the name stored in the TYPENAME column of the DATATYPES catalog view. For example:

| type name       | length | sqldatatype_name |
|-----------------|--------|------------------|
| -----           | -----  | -----            |
| A.B             | 10     | A .B             |
| INTEGER         | 16     | SYSIBM .INTEGER  |
| "Frank's".SMINT | 13     | Frank's .SMINT   |
| MY."type "      | 15     | MY .type         |

## Effect of DESCRIBE on the SQLDA

For a DESCRIBE OUTPUT or PREPARE OUTPUT INTO statement, the database manager always sets SQLD to the number of columns in the result set, or the number of output parameter markers. For a DESCRIBE INPUT or PREPARE INPUT INTO statement, the database manager always sets SQLD to the number of input parameter markers in the statement. Note that a parameter marker that corresponds to an INOUT parameter in a CALL statement is described in both the input and output descriptors.

The SQLVARs in the SQLDA are set in the following cases:

- $SQLN \geq SQLD$  and no entry is either a LOB, user-defined type or reference type  
The first SQLD SQLVAR entries are set and SQLDOUBLED is set to blank.
- $SQLN \geq 2 * SQLD$  and at least one entry is a LOB, user-defined type or reference type  
Two times SQLD SQLVAR entries are set, and SQLDOUBLED is set to '2'.
- $SQLD \leq SQLN < 2 * SQLD$  and at least one entry is a distinct, array, cursor, row, or reference type, but there are no LOB entries or structured type entries  
The first SQLD SQLVAR entries are set and SQLDOUBLED is set to blank. If the SQLWARN bind option is YES, a warning SQLCODE +237 (SQLSTATE 01594) is issued.

The SQLVARs in the SQLDA are NOT set (requiring allocation of additional space and another DESCRIBE) in the following cases:

- $SQLN < SQLD$  and no entry is either a LOB, user-defined type or reference type  
No SQLVAR entries are set and SQLDOUBLED is set to blank. If the SQLWARN bind option is YES, a warning SQLCODE +236 (SQLSTATE 01005) is issued.  
Allocate SQLD SQLVARs for a successful DESCRIBE.
- $SQLN < SQLD$  and at least one entry is a distinct, array, cursor, row, or reference type, but there are no LOB entries or structured type entries  
No SQLVAR entries are set and SQLDOUBLED is set to blank. If the SQLWARN bind option is YES, a warning SQLCODE +239 (SQLSTATE 01005) is issued.  
Allocate  $2 * SQLD$  SQLVARs for a successful DESCRIBE including the names of the distinct, array, cursor, and row types and target types of reference types.
- $SQLN < 2 * SQLD$  and at least one entry is a LOB or a structured type  
No SQLVAR entries are set and SQLDOUBLED is set to blank. A warning SQLCODE +238 (SQLSTATE 01005) is issued (regardless of the setting of the SQLWARN bind option).  
Allocate  $2 * SQLD$  SQLVARs for a successful DESCRIBE.

References in the above lists to LOB entries include distinct type entries whose source type is a LOB type.

The SQLWARN option of the BIND or PREP command is used to control whether the DESCRIBE (or PREPARE INTO) will return the warning SQLCODEs +236, +237, +239. It is recommended that your application code always consider that these SQLCODEs could be returned. The warning SQLCODE +238 is always returned when there are LOB or structured type entries in the select list and there are insufficient SQLVARs in the SQLDA. This is the only way the application can know that the number of SQLVARs must be doubled because of a LOB or structured type entry in the result set.

If a structured type entry is being described, but no FROM SQL transform is defined (either because no TRANSFORM GROUP was specified using the CURRENT DEFAULT TRANSFORM GROUP special register (SQLSTATE 42741) or because the name group does not have a FROM SQL transform function defined (SQLSTATE 42744)), the DESCRIBE will return an error. This error is the same error returned for a DESCRIBE of a table with a structured type entry.

If the database manager returns identifiers that are longer than those that can be stored in the SQLDA, the identifier is truncated and a warning is returned (SQLSTATE 01665); however, when the name of a structured type is truncated, an error is returned (SQLSTATE 42622). For details on identifier length limitations, see “SQL and XQuery limits” .

## SQLTYPE and SQLLEN

Table 28 shows the values that may appear in the SQLTYPE and SQLLEN fields of the SQLDA. In DESCRIBE and PREPARE INTO, an even value of SQLTYPE means that the column does not allow nulls, and an odd value means the column does allow nulls. In FETCH, OPEN, and EXECUTE, an even value of SQLTYPE means that no indicator variable is provided, and an odd value means that SQLIND contains the address of an indicator variable.

Table 28. SQLTYPE and SQLLEN values for DESCRIBE, FETCH, OPEN, and EXECUTE

| SQLTYPE | For DESCRIBE and PREPARE INTO        |                                                     | For FETCH, OPEN, and EXECUTE                                |                                       |
|---------|--------------------------------------|-----------------------------------------------------|-------------------------------------------------------------|---------------------------------------|
|         | Column Data Type                     | SQLLEN                                              | Host Variable Data Type                                     | SQLLEN                                |
| 384/385 | date                                 | 10                                                  | fixed-length character string representation of a date      | length attribute of the host variable |
| 388/389 | time                                 | 8                                                   | fixed-length character string representation of a time      | length attribute of the host variable |
| 392/393 | timestamp                            | 19 for TIMESTAMP(0) otherwise 20+p for TIMESTAMP(p) | fixed-length character string representation of a timestamp | length attribute of the host variable |
| 400/401 | N/A                                  | N/A                                                 | NULL-terminated graphic string                              | length attribute of the host variable |
| 404/405 | BLOB                                 | 0 *                                                 | BLOB                                                        | Not used. *                           |
| 408/409 | CLOB                                 | 0 *                                                 | CLOB                                                        | Not used. *                           |
| 412/413 | DBCLOB                               | 0 *                                                 | DBCLOB                                                      | Not used. *                           |
| 448/449 | varying-length character string      | length attribute of the column                      | varying-length character string                             | length attribute of the host variable |
| 452/453 | fixed-length character string        | length attribute of the column                      | fixed-length character string                               | length attribute of the host variable |
| 456/457 | long varying-length character string | length attribute of the column                      | long varying-length character string                        | length attribute of the host variable |
| 460/461 | not applicable                       | not applicable                                      | NULL-terminated character string                            | length attribute of the host variable |
| 464/465 | varying-length graphic string        | length attribute of the column                      | varying-length graphic string                               | length attribute of the host variable |
| 468/469 | fixed-length graphic string          | length attribute of the column                      | fixed-length graphic string                                 | length attribute of the host variable |



Table 28. *SQLTYPE and SQLLEN values for DESCRIBE, FETCH, OPEN, and EXECUTE (continued)*

| SQLTYPE     | For DESCRIBE and PREPARE INTO      |                                                | For FETCH, OPEN, and EXECUTE                                                    |                                                |
|-------------|------------------------------------|------------------------------------------------|---------------------------------------------------------------------------------|------------------------------------------------|
|             | Column Data Type                   | SQLLEN                                         | Host Variable Data Type                                                         | SQLLEN                                         |
| 472/473     | long varying-length graphic string | length attribute of the column                 | long graphic string                                                             | length attribute of the host variable          |
| 480/481     | floating-point                     | 8 for double precision, 4 for single precision | floating-point                                                                  | 8 for double precision, 4 for single precision |
| 484/485     | packed decimal                     | precision in byte 1; scale in byte 2           | packed decimal                                                                  | precision in byte 1; scale in byte 2           |
| 492/493     | big integer                        | 8                                              | big integer                                                                     | 8                                              |
| 496/497     | large integer                      | 4                                              | large integer                                                                   | 4                                              |
| 500/501     | small integer                      | 2                                              | small integer                                                                   | 2                                              |
| 916/917     | not applicable                     | not applicable                                 | BLOB file reference variable                                                    | 267                                            |
| 920/921     | not applicable                     | not applicable                                 | CLOB file reference variable                                                    | 267                                            |
| 924/925     | not applicable                     | not applicable                                 | DBCLOB file reference variable.                                                 | 267                                            |
| 960/961     | not applicable                     | not applicable                                 | BLOB locator                                                                    | 4                                              |
| 964/965     | not applicable                     | not applicable                                 | CLOB locator                                                                    | 4                                              |
| 968/969     | not applicable                     | not applicable                                 | DBCLOB locator                                                                  | 4                                              |
| 988/989     | XML                                | 0                                              | not applicable; use an XML AS <string or binary LOB type> host variable instead | not used                                       |
| 996         | decimal floating-point             | 8 for DECFLOAT(16), 16 for DECFLOAT(34)        | decimal floating-point                                                          | 8 for DECFLOAT(16), 16 for DECFLOAT(34)        |
| I 2440/2441 | row                                | not applicable                                 | row                                                                             | not used                                       |
| I 2440/2441 | cursor                             | not applicable                                 | row                                                                             | not used                                       |

**Note:**

- The len.sqllonglen field in the secondary SQLVAR contains the length attribute of the column.
- The SQLTYPE has changed from the previous version for portability in DB2. The values from the previous version (see previous version SQL Reference) continue to be supported.

## Unrecognized and unsupported SQLTYPES

The values that appear in the SQLTYPE field of the SQLDA are dependent on the level of data type support available at the sender as well as at the receiver of the data. This is particularly important as new data types are added to the product.

New data types may or may not be supported by the sender or receiver of the data and may or may not even be recognized by the sender or receiver of the data. Depending on the situation, the new data type may be returned, or a compatible data type agreed upon by both the sender and receiver of the data may be returned or an error may result.

When the sender and receiver agree to use a compatible data type, the following indicates the mapping that will take place. This mapping will take place when at least one of the sender or the receiver does not support the data type provided. The unsupported data type can be provided by either the application or the database manager.

| Data Type          | Compatible Data Type     |
|--------------------|--------------------------|
| BIGINT             | DECIMAL(19, 0)           |
| ROWID <sup>1</sup> | VARCHAR(40) FOR BIT DATA |

<sup>1</sup> ROWID is supported by DB2 Universal Database for z/OS Version 8.

Note that no indication is given in the SQLDA that the data type is substituted.

## Packed decimal numbers

Packed decimal numbers are stored in a variation of Binary Coded Decimal (BCD) notation. In BCD, each nybble (four bits) represents one decimal digit. For example, 0001 0111 1001 represents 179. Therefore, read a packed decimal value nybble by nybble. Store the value in bytes and then read those bytes in hexadecimal representation to return to decimal. For example, 0001 0111 1001 becomes 00000001 01111001 in binary representation. By reading this number as hexadecimal, it becomes 0179.

The decimal point is determined by the scale. In the case of a DEC(12,5) column, for example, the rightmost 5 digits are to the right of the decimal point.

Sign is indicated by a nybble to the right of the nybbles representing the digits. A positive or negative sign is indicated as follows:

Table 29. Values for Sign Indicator of a Packed Decimal Number

| Sign         | Representation |         |             |
|--------------|----------------|---------|-------------|
|              | Binary         | Decimal | Hexadecimal |
| Positive (+) | 1100           | 12      | C           |
| Negative (-) | 1101           | 13      | D           |

In summary:

- To store any value, allocate  $p/2+1$  bytes, where  $p$  is precision.
- Assign the nybbles from left to right to represent the value. If a number has an even precision, a leading zero nybble is added. This assignment includes leading (insignificant) and trailing (significant) zero digits.
- The sign nybble will be the second nybble of the last byte.

For example:

| Column   | Value   | Nybbles in Hexadecimal Grouped by Bytes |
|----------|---------|-----------------------------------------|
| DEC(8,3) | 6574.23 | 00 65 74 23 0C                          |
| DEC(6,2) | -334.02 | 00 33 40 2D                             |
| DEC(7,5) | 5.2323  | 05 23 23 0C                             |
| DEC(5,2) | -23.5   | 02 35 0D                                |

## SQLLEN field for decimal

The SQLLEN field contains the precision (first byte) and scale (second byte) of the decimal column. If writing a portable application, the precision and scale bytes should be set individually, versus setting them together as a short integer. This will avoid integer byte reversal problems.

For example, in C:

```
((char *)&(sqlda->sqlvar[i].sqlen))[0] = precision;
((char *)&(sqlda->sqlvar[i].sqlen))[1] = scale;
```

---

## Chapter 8. Identifiers

An *identifier* is a token that is used to form a name. An identifier in an SQL statement is either an SQL identifier or a host identifier.

- SQL identifiers

There are two types of *SQL identifiers*: ordinary and delimited.

- An *ordinary identifier* is an uppercase letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. Note that ordinary identifiers are converted to uppercase. An ordinary identifier should not be a reserved word.

*Examples*

WKLYSAL      WKLY\_SAL

- A *delimited identifier* is a sequence of one or more characters enclosed by double quotation marks. Two consecutive quotation marks are used to represent one quotation mark within the delimited identifier. In this way an identifier can include lowercase letters.

*Examples*

"WKLY\_SAL"      "WKLY SAL"      "UNION"      "wkly\_sal"

Character conversion of identifiers created on a double-byte code page, but used by an application or database on a multi-byte code page, may require special consideration: After conversion, such identifiers may exceed the length limit for an identifier.

- Host identifiers

A *host identifier* is a name declared in the host program. The rules for forming a host identifier are the rules of the host language. A host identifier should not be greater than 255 bytes in length and should not begin with SQL or DB2 (in uppercase or lowercase characters).

### Naming conventions and implicit object name qualifications

The rules for forming a database object name depend on the type of the object designated by the name. A name may consist of a single SQL identifier or it may be qualified with one or more identifiers that more specifically identify the object. A period must separate each identifier.

The following object names, when used in the context of an SQL procedure, are permitted to use only the characters allowed in an ordinary identifier, even if the names are delimited:

- condition-name
- label
- parameter-name
- procedure-name
- SQL-variable-name
- statement-name

The syntax diagrams use different terms for different types of names. The following list defines these terms.

#### **alias-name**

A schema-qualified name that designates an alias.

**attribute-name**

An identifier that designates an attribute of a structured data type.

**authorization-name**

An identifier that designates a user, group, or role. For a user or a group:

- Valid characters are: 'A' through 'Z'; 'a' through 'z'; '0' through '9'; '#'; '@'; '\$'; '\_'; '!'; '('; ')'; '{'; '}'; '-'; '.'; and '^'.
- The following characters must be delimited with quotation marks when entered through the command line processor: '!'; '('; ')'; '{'; '}'; '-'; '.'; and '^'.
- The name must not begin with the characters 'SYS', 'IBM', or 'SQL'.
- The name must not be: 'ADMINS', 'GUESTS', 'LOCAL', 'PUBLIC', or 'USERS'.
- A delimited authorization ID must not contain lowercase letters.

**bufferpool-name**

An identifier that designates a buffer pool.

**column-name**

A qualified or unqualified name that designates a column of a table or view. The qualifier is a table name, a view name, a nickname, or a correlation name.

**component-name**

An identifier that designates a security label component.

**condition-name**

A qualified or unqualified name that designates a condition. An unqualified condition name in an SQL statement is implicitly qualified, depending on its context. If the condition is defined in a module and used outside of the same module, it must be qualified by the module-name.

**constraint-name**

An identifier that designates a referential constraint, primary key constraint, unique constraint, or a table check constraint.

**correlation-name**

An identifier that designates a result table.

**cursor-name**

An identifier that designates an SQL cursor. For host compatibility, a hyphen character may be used in the name.

**cursor-type-name**

A qualified or unqualified name that designates a user-defined cursor type. An unqualified cursor-type-name in an SQL statement is implicitly qualified, depending on context.

**cursor-variable-name**

A qualified or unqualified name that designates a global variable, local variable or an SQL parameter of a cursor type. An unqualified cursor variable name in an SQL statement is implicitly qualified, depending on context.

**data-source-name**

An identifier that designates a data source. This identifier is the first part of a three-part remote object name.

**db-partition-group-name**

An identifier that designates a database partition group.

**descriptor-name**

A colon followed by a host identifier that designates an SQL descriptor area (SQLDA). For the description of a host identifier, see "References to host variables" on page 582. Note that a descriptor name never includes an indicator variable.

**distinct-type-name**

A qualified or unqualified name that designates a distinct type. The unqualified form of distinct-type-name is an SQL identifier. An unqualified distinct type name in an SQL statement is implicitly qualified. The implicit qualifier is a schema name or a module name, which is determined by the context in which distinct-type-name appears. The qualified form is a schema-name followed by a period and an SQL identifier or a module-name (which can also be qualified by a schema-name) followed by a period and an SQL identifier. If the distinct type is defined in a module and used outside of the same module, it must be qualified by the module-name.

**event-monitor-name**

An identifier that designates an event monitor.

**function-mapping-name**

An identifier that designates a function mapping.

**function-name**

A qualified or unqualified name that designates a function. The unqualified form of function-name is an SQL identifier. An unqualified function name in an SQL statement is implicitly qualified. The implicit qualifier is a schema name, which is determined by the context in which the function appears. The qualified form could be is a schema-name followed by a period and an SQL identifier or a module-name followed by a period and an SQL identifier. If the function is published in a module and used outside of the same module, it must be qualified by the module-name.

**global-variable-name**

A qualified or unqualified name that designates a global variable. An unqualified global variable name in an SQL statement is implicitly qualified, depending on context. If the global variable is defined in a module and used outside of the same module, it must be qualified by the module-name.

**group-name**

An unqualified identifier that designates a transform group defined for a structured type.

**host-variable**

A sequence of tokens that designates a host variable. A host variable includes at least one host identifier, explained in "References to host variables" on page 582.

**index-name**

A schema-qualified name that designates an index or an index specification.

**label** An identifier that designates a label in an SQL procedure.

**method-name**

An identifier that designates a method. The schema context for a method is determined by the schema of the subject type (or a supertype of the subject type) of the method.

| **module-name**

| A qualified or unqualified name that designates a module. An unqualified  
| module-name in an SQL statement is implicitly qualified. The implicit  
| qualifier is a schema name, which is determined by the context in which  
| the module-name appears. The qualified form is a schema-name followed  
| by a period and an SQL identifier.

| **nickname**

| A schema-qualified name that designates a federated server reference to a  
| table or a view.

| **package-name**

| A schema-qualified name that designates a package. If a package has a  
| version ID that is not the empty string, the package name also includes the  
| version ID at the end of the name, in the form: schema-id.package-  
| id.version-id.

| **parameter-name**

| An identifier that designates a parameter that can be referenced in a  
| procedure, user-defined function, method, or index extension.

| **partition-name**

| An identifier that designates a data partition in a partitioned table.

| **procedure-name**

| A qualified or unqualified name that designates a procedure. The  
| unqualified form of procedure-name is an SQL identifier. An unqualified  
| procedure name in an SQL statement is implicitly qualified. The implicit  
| qualifier is a schema name, which is determined by the context in which  
| the procedure appears. The qualified form is a schema-name followed by a  
| period and an SQL identifier or a module-name followed by a period and  
| an SQL identifier. If the procedure is defined in a module and used outside  
| of the same module, it must be qualified by the module-name.

| **remote-authorization-name**

| An identifier that designates a data source user. The rules for authorization  
| names vary from data source to data source.

| **remote-function-name**

| A name that designates a function registered to a data source database.

| **remote-object-name**

| A three-part name that designates a data source table or view, and that  
| identifies the data source in which the table or view resides. The parts of  
| this name are data-source-name, remote-schema-name, and  
| remote-table-name.

| **remote-schema-name**

| A name that designates the schema to which a data source table or view  
| belongs. This name is the second part of a three-part remote object name.

| **remote-table-name**

| A name that designates a table or view at a data source. This name is the  
| third part of a three-part remote object name.

| **remote-type-name**

| A data type supported by a data source database. Do not use the long  
| form for built-in types (use CHAR instead of CHARACTER, for example).

| **role-name**

| An identifier that designates a role.



|

|

|

|

|

|

|

|

|

**row-type-name**

A qualified or unqualified name that designates a row type. The unqualified form of row-type-name is an SQL identifier. An unqualified row-type-name in an SQL statement is implicitly qualified. The implicit qualifier is a schema name, which is determined by the context in which the row-type-name appears as described by the rules in “Unqualified user-defined type, function, procedure, and specific names”. The qualified form is a schema-name followed by a period and an SQL identifier.

**savepoint-name**

An identifier that designates a savepoint.

**schema-name**

An identifier that provides a logical grouping for SQL objects. A schema name used as a qualifier for the name of an object may be implicitly determined:

- from the value of the CURRENT SCHEMA special register
- from the value of the QUALIFIER precompile/bind option
- on the basis of a resolution algorithm that uses the CURRENT PATH special register
- on the basis of the schema name for another object in the same SQL statement.

To avoid complications, it is recommended that the name SESSION not be used as a schema, except as the schema for declared global temporary tables (which *must* use the schema name SESSION).

**security-label-name**

A qualified or unqualified name that designates a security label. An unqualified security label name in an SQL statement is implicitly qualified by the applicable security-policy-name, when one applies. If no security-policy-name is implicitly applicable, the name must be qualified.

**security-policy-name**

An identifier that designates a security policy.

**sequence-name**

An identifier that designates a sequence.

**server-name**

An identifier that designates an application server. In a federated system, the server name also designates the local name of a data source.

**specific-name**

|

|

|

A qualified or unqualified name that designates a specific name. An unqualified specific name in an SQL statement is implicitly qualified, depending on context.

**SQL-variable-name**

The name of a local variable in an SQL procedure statement. SQL variable names can be used in other SQL statements where a host variable name is allowed. The name can be qualified by the label of the compound statement that declared the SQL variable.

**statement-name**

An identifier that designates a prepared SQL statement.

**supertype-name**

A qualified or unqualified name that designates the supertype of a type. An unqualified supertype name in an SQL statement is implicitly qualified, depending on context.

**table-name**

A schema-qualified name that designates a table.

**table-reference**

A qualified or unqualified name that designates a table. An unqualified table reference in a common table expression is implicitly qualified by the default schema.

**tablespace-name**

An identifier that designates a table space.

**trigger-name**

A schema-qualified name that designates a trigger.

**type-mapping-name**

An identifier that designates a data type mapping.

**type-name**

A qualified or unqualified name that designates a type. An unqualified type name in an SQL statement is implicitly qualified, depending on context.

**typed-table-name**

A schema-qualified name that designates a typed table.

**typed-view-name**

A schema-qualified name that designates a typed view.

**user-defined-type-name**

A qualified or unqualified name that designates a user-defined data type. The unqualified form of user-defined-type-name is an SQL identifier. An unqualified user-defined-type-name in an SQL statement is implicitly qualified. The implicit qualifier is a schema name or a module name, which is determined by the context in which user-defined-type-name appears. The qualified form is a schema-name followed by a period and an SQL identifier or a module-name (which can also be qualified by a schema-name) followed by a period and an SQL identifier. If the user-defined data type is defined in a module and used outside of the same module, it must be qualified by the module-name.

**view-name**

A schema-qualified name that designates a view.

**wrapper-name**

An identifier that designates a wrapper.

**XML-schema-name**

A qualified or unqualified name that designates an XML schema.

**xsobject-name**

A qualified or unqualified name that designates an object in the XML schema repository.

**Aliases for database objects**

An alias can be thought of as an alternative name for an SQL object. An SQL object, therefore, can be referred to in an SQL statement by its name or by an alias.

A public alias is an alias which can always be referenced without qualifying its name with a schema name. The implicit qualifier of a public alias is SYSPUBLIC, which can also be specified explicitly.

Aliases are also known as synonyms.

An alias can be used wherever the object it is based on can be used. An alias can be created even if the object does not exist (although it must exist by the time a statement referring to it is compiled). It can refer to another alias if no circular or repetitive references are made along the chain of aliases. An alias can only refer to a module, nickname, sequence, table, view, or another alias within the same database. An alias name cannot be used where a new object name is expected, such as in the CREATE TABLE or CREATE VIEW statements; for example, if the table alias name PERSONNEL has been created, subsequent statements such as CREATE TABLE PERSONNEL... will return an error.

The option of referring to an object by an alias is not explicitly shown in the syntax diagrams, or mentioned in the descriptions of SQL statements.

A new unqualified alias of a given object type, say for a sequence, cannot have the same fully-qualified name as an existing object of that object type. For example, a sequence alias named ORDERID cannot be defined in the KANDIL schema for the sequence named KANDIL.ORDERID.

The effect of using an alias in an SQL statement is similar to that of text substitution. The alias, which must be defined by the time that the SQL statement is compiled, is replaced at statement compilation time by the qualified object name. For example, if PBIRD.SALES is an alias for DSPN014.DIST4\_SALES\_148, then at compilation time:

```
SELECT * FROM PBIRD.SALES
```

effectively becomes

```
SELECT * FROM DSPN014.DIST4_SALES_148
```

## Authorization IDs and authorization names

An *authorization ID* is a character string that is obtained by the database manager when a connection is established between the database manager and either an application process or a program preparation process. It designates a set of privileges. It may also designate a user or a group of users, but this property is not controlled by the database manager.

Authorization IDs are used by the database manager to provide:

- Authorization checking of SQL statements
- A default value for the QUALIFIER precompile/bind option and the CURRENT SCHEMA special register. The authorization ID is also included in the default CURRENT PATH special register and the FUNCPATH precompile/bind option.

An authorization ID applies to every SQL statement. The authorization ID that applies to a static SQL statement is the authorization ID that is used during program binding. The authorization ID that applies to a dynamic SQL statement is based on the DYNAMICRULES option supplied at bind time, and on the current runtime environment for the package issuing the dynamic SQL statement:

- In a package that has bind behavior, the authorization ID used is the authorization ID of the package owner.

- In a package that has define behavior, the authorization ID used is the authorization ID of the corresponding routine's definer.
- In a package that has run behavior, the authorization ID used is the current authorization ID of the user executing the package.
- In a package that has invoke behavior, the authorization ID used is the authorization ID currently in effect when the routine is invoked. This is called the runtime authorization ID.

For more information, see "Dynamic SQL characteristics at run time" on page 573.

An *authorization name* specified in an SQL statement should not be confused with the authorization ID of the statement. An authorization name is an identifier that is used within various SQL statements. An authorization name is used in the CREATE SCHEMA statement to designate the owner of the schema. An authorization name is used in the GRANT and REVOKE statements to designate a target of the grant or revoke operation. Granting privileges to *X* means that *X* (or a member of the group or role *X*) will subsequently be the authorization ID of statements that require those privileges.

*Examples:*

- Assume that SMITH is the user ID and the authorization ID that the database manager obtained when a connection was established with the application process. The following statement is executed interactively:

```
GRANT SELECT ON TDEPT TO KEENE
```

SMITH is the authorization ID of the statement. Therefore, in a dynamic SQL statement, the default value of the CURRENT SCHEMA special register is SMITH, and in static SQL, the default value of the QUALIFIER precompile/bind option is SMITH. The authority to execute the statement is checked against SMITH, and SMITH is the *table-name* implicit qualifier based on qualification rules described in "Naming conventions and implicit object name qualifications" on page 565.

KEENE is an authorization name specified in the statement. KEENE is given the SELECT privilege on SMITH.TDEPT.

- Assume that SMITH has administrative authority and is the authorization ID of the following dynamic SQL statements, with no SET SCHEMA statement issued during the session:

```
DROP TABLE TDEPT
```

Removes the SMITH.TDEPT table.

```
DROP TABLE SMITH.TDEPT
```

Removes the SMITH.TDEPT table.

```
DROP TABLE KEENE.TDEPT
```

Removes the KEENE.TDEPT table. Note that KEENE.TDEPT and SMITH.TDEPT are different tables.

```
CREATE SCHEMA PAYROLL AUTHORIZATION KEENE
```

KEENE is the authorization name specified in the statement that creates a schema called PAYROLL. KEENE is the owner of the schema PAYROLL and is given CREATEIN, ALTERIN, and DROPIN privileges, with the ability to grant them to others.

## Dynamic SQL characteristics at run time

The BIND option DYNAMICRULES determines the authorization ID that is used for checking authorization when dynamic SQL statements are processed. In addition, the option also controls other dynamic SQL attributes, such as the implicit qualifier that is used for unqualified object references, and whether certain SQL statements can be invoked dynamically.

The set of values for the authorization ID and other dynamic SQL attributes is called the dynamic SQL statement behavior. The four possible behaviors are run, bind, define, and invoke. As the following table shows, the combination of the value of the DYNAMICRULES BIND option and the runtime environment determines which of the behaviors is used. DYNAMICRULES RUN, which implies run behavior, is the default.

*Table 30. How DYNAMICRULES and the runtime environment determine dynamic SQL statement behavior*

| DYNAMICRULES value | Behavior of dynamic SQL statements |                     |
|--------------------|------------------------------------|---------------------|
|                    | Standalone program environment     | Routine 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     |

### Run behavior

DB2 uses the authorization ID of the user (the ID that initially connected to DB2) executing the package as the value to be used for authorization checking of dynamic SQL statements and for the initial value used for implicit qualification of unqualified object references within dynamic SQL statements.

### Bind behavior

At run time, DB2 uses all the rules that apply to static SQL for authorization and qualification. It takes the authorization ID of the package owner as the value to be used for authorization checking of dynamic SQL statements, and the package default qualifier for implicit qualification of unqualified object references within dynamic SQL statements.

### Define behavior

Define behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES DEFINEBIND or DYNAMICRULES DEFINERUN. DB2 uses the authorization ID of the routine definer (not the routine's package binder) as the value to be used for authorization checking of dynamic SQL statements, and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

### Invoke behavior

Invoke behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES INVOKEBIND or DYNAMICRULES INVOKERUN. DB2 uses the statement authorization ID in effect when the routine is invoked

as the value to be used for authorization checking of dynamic SQL, and for implicit qualification of unqualified object references within dynamic SQL statements within that routine. This is summarized by the following table.

| Invoking Environment                        | ID Used                                                                                       |
|---------------------------------------------|-----------------------------------------------------------------------------------------------|
| any static SQL                              | implicit or explicit value of the OWNER of the package the SQL invoking the routine came from |
| used in definition of view or trigger       | definer of the view or trigger                                                                |
| dynamic SQL from a bind behavior package    | implicit or explicit value of the OWNER of the package the SQL invoking the routine came from |
| dynamic SQL from a run behavior package     | ID used to make the initial connection to DB2                                                 |
| dynamic SQL from a define behavior package  | definer of the routine that uses the package that the SQL invoking the routine came from      |
| dynamic SQL from an invoke behavior package | the current authorization ID invoking the routine                                             |

### *Restricted statements when run behavior does not apply*

When bind, define, or invoke behavior is in effect, you cannot use the following dynamic SQL statements: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT, RENAME, SET INTEGRITY, SET EVENT MONITOR STATE; or queries that reference a nickname.

### *Considerations regarding the DYNAMICRULES option*

The CURRENT SCHEMA special register cannot be used to qualify unqualified object references within dynamic SQL statements executed from bind, define or invoke behavior packages. This is true even after you issue the SET CURRENT SCHEMA statement to change the CURRENT SCHEMA special register; the register value is changed but not used.

In the event that multiple packages are referenced during a single connection, all dynamic SQL statements prepared by those packages will exhibit the behavior specified by the DYNAMICRULES option for that specific package and the environment in which they are used.

It is important to keep in mind that when a package exhibits bind behavior, the binder of the package should not have any authorities granted that the user of the package should not receive, because a dynamic statement will be using the authorization ID of the package owner. Similarly, when a package exhibits define behavior, the definer of the routine should not have any authorities granted that the user of the package should not receive.

## **Authorization IDs and statement preparation**

If the VALIDATE BIND option is specified at bind time, the privileges required to manipulate tables and views must also exist at bind time. If these privileges or the referenced objects do not exist, and the SQLERROR NOPACKAGE option is in effect, the bind operation will be unsuccessful. If the SQLERROR CONTINUE

option is specified, the bind operation will be successful, and any statements in error will be flagged. Any attempt to execute such a statement will result in an error.

If a package is bound with the VALIDATE RUN option, all normal bind processing is completed, but the privileges required to use the tables and views that are referenced in the application need not exist yet. If a required privilege does not exist at bind time, an incremental bind operation is performed whenever the statement is first executed in an application, and all privileges required for the statement must exist. If a required privilege does not exist, execution of the statement is unsuccessful.

Authorization checking at run time is performed using the authorization ID of the package owner.

## Column names

The meaning of a *column name* depends on its context. A column name can be used to:

- Declare the name of a column, as in a CREATE TABLE statement.
- Identify a column, as in a CREATE INDEX statement.
- Specify values of the column, as in the following contexts:
  - In an aggregate function, a column name specifies all values of the column in the group or intermediate result table to which the function is applied. For example, MAX(SALARY) applies the function MAX to all values of the column SALARY in a group.
  - In a GROUP BY or ORDER BY clause, a column name specifies all values in the intermediate result table to which the clause is applied. For example, ORDER BY DEPT orders an intermediate result table by the values of the column DEPT.
  - In an expression, a search condition, or a scalar function, a column name specifies a value for each row or group to which the construct is applied. For example, when the search condition CODE = 20 is applied to some row, the value specified by the column name CODE is the value of the column CODE in that row.
- Temporarily rename a column, as in the *correlation-clause* of a *table-reference* in a FROM clause.

## Qualified column names

A qualifier for a column name may be a table, view, nickname, alias, or correlation name.

Whether a column name may be qualified depends on its context:

- Depending on the form of the COMMENT ON statement, a single column name may need to be qualified. Multiple column names must be unqualified.
- Where the column name specifies values of the column, it may be qualified at the user's option.
- In the assignment-clause of an UPDATE statement, it may be qualified at the user's option.
- In all other contexts, a column name must not be qualified.



Where a qualifier is optional, it can serve two purposes. They are described under “Column name qualifiers to avoid ambiguity” on page 577 and “Column name qualifiers in correlated references” on page 579.

## Correlation names

A *correlation name* can be defined in the FROM clause of a query and in the first clause of an UPDATE or DELETE statement. For example, the clause FROM X.MYTABLE Z establishes Z as a correlation name for X.MYTABLE.

```
FROM X.MYTABLE Z
```

With Z defined as a correlation name for X.MYTABLE, only Z can be used to qualify a reference to a column of that instance of X.MYTABLE in that SELECT statement.

A correlation name is associated with a table, view, nickname, alias, nested table expression, table function, or data change table reference only within the context in which it is defined. Hence, the same correlation name can be defined for different purposes in different statements, or in different clauses of the same statement.

As a qualifier, a correlation name can be used to avoid ambiguity or to establish a correlated reference. It can also be used merely as a shorter name for a table reference. In the example, Z might have been used merely to avoid having to enter X.MYTABLE more than once.

If a correlation name is specified for a table, view, nickname, or alias name, any qualified reference to a column of that instance of the table, view, nickname, or alias must use the correlation name, rather than the table, view, nickname, or alias name. For example, the reference to EMPLOYEE.PROJECT in the following example is incorrect, because a correlation name has been specified for EMPLOYEE:

Example

```
FROM EMPLOYEE E
WHERE EMPLOYEE.PROJECT='ABC' * incorrect*
```

The qualified reference to PROJECT should instead use the correlation name, “E”, as shown below:

```
FROM EMPLOYEE E
WHERE E.PROJECT='ABC'
```

Names specified in a FROM clause are either *exposed* or *non-exposed*. A table, view, nickname, or alias name is said to be exposed in the FROM clause if a correlation name is not specified. A correlation name is always an exposed name. For example, in the following FROM clause, a correlation name is specified for EMPLOYEE but not for DEPARTMENT, so DEPARTMENT is an exposed name, and EMPLOYEE is not:

```
FROM EMPLOYEE E, DEPARTMENT
```

A table, view, nickname, or alias name that is exposed in a FROM clause may be the same as any other table name, view name or nickname exposed in that FROM clause or any correlation name in the FROM clause. This may result in ambiguous column name references which returns an error (SQLSTATE 42702).

The first two FROM clauses shown below are correct, because each one contains no more than one reference to EMPLOYEE that is exposed:

1. Given the FROM clause:

```
FROM EMPLOYEE E1, EMPLOYEE
```

a qualified reference such as EMPLOYEE.PROJECT denotes a column of the second instance of EMPLOYEE in the FROM clause. A qualified reference to the first instance of EMPLOYEE must use the correlation name "E1" (E1.PROJECT).

2. Given the FROM clause:

```
FROM EMPLOYEE, EMPLOYEE E2
```

a qualified reference such as EMPLOYEE.PROJECT denotes a column of the first instance of EMPLOYEE in the FROM clause. A qualified reference to the second instance of EMPLOYEE must use the correlation name "E2" (E2.PROJECT).

3. Given the FROM clause:

```
FROM EMPLOYEE, EMPLOYEE
```

the two exposed table names included in this clause (EMPLOYEE and EMPLOYEE) are the same. This is allowed, but references to specific column names would be ambiguous (SQLSTATE 42702).

4. Given the following statement:

```
SELECT *
FROM EMPLOYEE E1, EMPLOYEE E2 * incorrect *
WHERE EMPLOYEE.PROJECT = 'ABC'
```

the qualified reference EMPLOYEE.PROJECT is incorrect, because both instances of EMPLOYEE in the FROM clause have correlation names. Instead, references to PROJECT must be qualified with either correlation name (E1.PROJECT or E2.PROJECT).

5. Given the FROM clause:

```
FROM EMPLOYEE, X.EMPLOYEE
```

a reference to a column in the second instance of EMPLOYEE must use X.EMPLOYEE (X.EMPLOYEE.PROJECT). If X is the CURRENT SCHEMA special register value in dynamic SQL or the QUALIFIER precompile/bind option in static SQL, then the columns cannot be referenced since any such reference would be ambiguous.

The use of a correlation name in the FROM clause also allows the option of specifying a list of column names to be associated with the columns of the result table. As with a correlation name, these listed column names become the *exposed* names of the columns that must be used for references to the columns throughout the query. If a column name list is specified, then the column names of the underlying table become *non-exposed*.

Given the FROM clause:

```
FROM DEPARTMENT D (NUM,NAME,MGR,ANUM,LOC)
```

a qualified reference such as D.NUM denotes the first column of the DEPARTMENT table that is defined in the table as DEPTNO. A reference to D.DEPTNO using this FROM clause is incorrect since the column name DEPTNO is a non-exposed column name.

## Column name qualifiers to avoid ambiguity

In the context of a function, a GROUP BY clause, ORDER BY clause, an expression, or a search condition, a column name refers to values of a column in some table,

view, nickname, nested table expression or table function. The tables, views, nicknames, nested table expressions and table functions that might contain the column are called the *object tables* of the context. Two or more object tables might contain columns with the same name; one reason for qualifying a column name is to designate the table from which the column comes. Qualifiers for column names are also useful in SQL procedures to distinguish column names from SQL variable names used in SQL statements.

A nested table expression or table function will consider *table-references* that precede it in the FROM clause as object tables. The *table-references* that follow are not considered as object tables.

## Table designators

A qualifier that designates a specific object table is called a *table designator*. The clause that identifies the object tables also establishes the table designators for them. For example, the object tables of an expression in a SELECT clause are named in the FROM clause that follows it:

```
SELECT CORZ.COLA, OWNY.MYTABLE.COLA
FROM OWNX.MYTABLE CORZ, OWNY.MYTABLE
```

Table designators in the FROM clause are established as follows:

- A name that follows a table, view, nickname, alias, nested table expression or table function is both a correlation name and a table designator. Thus, CORZ is a table designator. CORZ is used to qualify the first column name in the select list.
- An exposed table, view name, nickname or alias is a table designator. Thus, OWNY.MYTABLE is a table designator. OWNY.MYTABLE is used to qualify the second column name in the select list.

When qualifying a column with the exposed table name form of a table designator, either the qualified or unqualified form of the exposed table name can be used. If the qualified form is used, the qualifier must be the same as the default qualifier for the exposed table name.

For example, assume that the current schema is CORPDATA.

```
SELECT CORPDATA.EMPLOYEE.WORKDEPT FROM EMPLOYEE
```

is valid because the EMPLOYEE table referenced in the FROM clause fully qualifies to CORPDATA.EMPLOYEE, which matches the qualifier for the WORKDEPT column.

```
SELECT EMPLOYEE.WORKDEPT, REGEMP.WORKDEPT
FROM CORPDATA.EMPLOYEE, REGION.EMPLOYEE REGEMP
```

is also valid, because the first select list column references the unqualified exposed table designator CORPDATA.EMPLOYEE, which is in the FROM clause, and the second select list column references the correlation name REGEMP of the table object REGION.EMPLOYEE, which is also in the FROM clause.

Now assume that the current schema is REGION.

```
SELECT CORPDATA.EMPLOYEE.WORKDEPT FROM EMPLOYEE
```

is not valid because the EMPLOYEE table referenced in the FROM clause fully qualifies to REGION.EMPLOYEE, and the qualifier for the WORKDEPT column represents the CORPDATA.EMPLOYEE table.

Each table designator should be unique within a particular FROM clause to avoid the possibility of ambiguous references to columns.

## Avoiding undefined or ambiguous references

When a column name refers to values of a column, exactly one object table must include a column with that name. The following situations are considered errors:

- No object table contains a column with the specified name. The reference is undefined.
- The column name is qualified by a table designator, but the table designated does not include a column with the specified name. Again the reference is undefined.
- The name is unqualified, and more than one object table includes a column with that name. The reference is ambiguous.
- The column name is qualified by a table designator, but the table designated is not unique in the FROM clause and both occurrences of the designated table include the column. The reference is ambiguous.
- The column name is in a nested table expression which is not preceded by the TABLE keyword or in a table function or nested table expression that is the right operand of a right outer join or a full outer join and the column name does not refer to a column of a *table-reference* within the nested table expression's fullselect. The reference is undefined.

Avoid ambiguous references by qualifying a column name with a uniquely defined table designator. If the column is contained in several object tables with different names, the table names can be used as designators. Ambiguous references can also be avoided without the use of the table designator by giving unique names to the columns of one of the object tables using the column name list following the correlation name.

When qualifying a column with the exposed table name form of a table designator, either the qualified or unqualified form of the exposed table name may be used. However, the qualifier used and the table used must be the same after fully qualifying the table name, view name or nickname and the table designator.

1. If the authorization ID of the statement is CORPDATA:

```
SELECT CORPDATA.EMPLOYEE.WORKDEPT
FROM EMPLOYEE
```

is a valid statement.

2. If the authorization ID of the statement is REGION:

```
SELECT CORPDATA.EMPLOYEE.WORKDEPT
FROM EMPLOYEE * incorrect *
```

is invalid, because EMPLOYEE represents the table REGION.EMPLOYEE, but the qualifier for WORKDEPT represents a different table, CORPDATA.EMPLOYEE.

## Column name qualifiers in correlated references

A *fullselect* is a form of a query that may be used as a component of various SQL statements. A fullselect used within a search condition of any statement is called a *subquery*. A fullselect used to retrieve a single value as an expression within a statement is called a *scalar fullselect* or *scalar subquery*. A fullselect used in the FROM clause of a query is called a *nested table expression*. Subqueries in search conditions, scalar subqueries and nested table expressions are referred to as subqueries through the remainder of this topic.

A subquery may include subqueries of its own, and these may, in turn, include subqueries. Thus an SQL statement may contain a hierarchy of subqueries. Those elements of the hierarchy that contain subqueries are said to be at a higher level than the subqueries they contain.

Every element of the hierarchy contains one or more table designators. A subquery can reference not only the columns of the tables identified at its own level in the hierarchy, but also the columns of the tables identified previously in the hierarchy, back to the highest level of the hierarchy. A reference to a column of a table identified at a higher level is called a *correlated reference*.

For compatibility with existing standards for SQL, both qualified and unqualified column names are allowed as correlated references. However, it is good practice to qualify all column references used in subqueries; otherwise, identical column names may lead to unintended results. For example, if a table in a hierarchy is altered to contain the same column name as the correlated reference and the statement is prepared again, the reference will apply to the altered table.

When a column name in a subquery is qualified, each level of the hierarchy is searched, starting at the same subquery as the qualified column name appears and continuing to the higher levels of the hierarchy until a table designator that matches the qualifier is found. Once found, it is verified that the table contains the given column. If the table is found at a higher level than the level containing column name, then it is a correlated reference to the level where the table designator was found. A nested table expression must be preceded with the optional TABLE keyword in order to search the hierarchy above the fullselect of the nested table expression.

When the column name in a subquery is not qualified, the tables referenced at each level of the hierarchy are searched, starting at the same subquery where the column name appears and continuing to higher levels of the hierarchy, until a match for the column name is found. If the column is found in a table at a higher level than the level containing column name, then it is a correlated reference to the level where the table containing the column was found. If the column name is found in more than one table at a particular level, the reference is ambiguous and considered an error.

In either case, T, used in the following example, refers to the table designator that contains column C. A column name, T.C (where T represents either an implicit or an explicit qualifier), is a correlated reference if, and only if, these conditions are met:

- T.C is used in an expression of a subquery.
- T does not designate a table used in the from clause of the subquery.
- T designates a table used at a higher level of the hierarchy that contains the subquery.

Since the same table, view or nickname can be identified at many levels, unique correlation names are recommended as table designators. If T is used to designate a table at more than one level (T is the table name itself or is a duplicate correlation name), T.C refers to the level where T is used that most directly contains the subquery that includes T.C. If a correlation to a higher level is needed, a unique correlation name must be used.

The correlated reference T.C identifies a value of C in a row or group of T to which two search conditions are being applied: condition 1 in the subquery, and condition

2 at some higher level. If condition 2 is used in a WHERE clause, the subquery is evaluated for each row to which condition 2 is applied. If condition 2 is used in a HAVING clause, the subquery is evaluated for each group to which condition 2 is applied.

For example, in the following statement, the correlated reference X.WORKDEPT (in the last line) refers to the value of WORKDEPT in table EMPLOYEE at the level of the first FROM clause. (That clause establishes X as a correlation name for EMPLOYEE.) The statement lists employees who make less than the average salary for their department.

```
SELECT EMPNO, LASTNAME, WORKDEPT
FROM EMPLOYEE X
WHERE SALARY < (SELECT AVG(SALARY)
 FROM EMPLOYEE
 WHERE WORKDEPT = X.WORKDEPT)
```

The next example uses THIS as a correlation name. The statement deletes rows for departments that have no employees.

```
DELETE FROM DEPARTMENT THIS
WHERE NOT EXISTS(SELECT *
 FROM EMPLOYEE
 WHERE WORKDEPT = THIS.DEPTNO)
```

## References to variables

A *variable* in an SQL statement specifies a value that can be changed when the SQL statement is executed. There are several types of variables used in SQL statements:

### host variable

Host variables are defined by statements of a host language. For more information about how to refer to host variables, see “References to host variables” on page 582.

### transition variable

Transition variables are defined in a trigger and refer to either the old or new values of columns. For more information about how to refer to transition variables, see “CREATE TRIGGER statement” in the *SQL Reference, Volume 2*.

### SQL variable

SQL variables are defined by an SQL compound statement in an SQL function, SQL method, SQL procedure, trigger, or dynamic SQL statement. For more information about SQL variables, see “References to SQL parameters, SQL variables, and global variables” in the *SQL Reference, Volume 2*.

### global variable

Global variables are defined by the CREATE VARIABLE statement. For more information about global variables, see “CREATE VARIABLE” and “References to SQL parameters, SQL variables, and global variables” in the *SQL Reference, Volume 2*.

### module variable

Module variables are defined by the ALTER MODULE statement using the ADD VARIABLE or PUBLISH VARIABLE operation. For more information about module variables, see “ALTER MODULE”.

### SQL parameter

SQL parameters are defined by a CREATE FUNCTION, CREATE METHOD, or CREATE PROCEDURE statement. For more information



about SQL parameters, see “References to SQL parameters, SQL variables, and global variables” in the *SQL Reference, Volume 2* .

### parameter marker

Parameter markers are specified in a dynamic SQL statement where host variables would be specified if the statement were a static SQL statement. An SQL descriptor or parameter binding is used to associate a value with a parameter marker during dynamic SQL statement processing. For more information about parameter markers, see Parameter markers.

## References to host variables

A *host variable* is either:

- A variable in a host language such as a C variable, a C++ variable, a COBOL data item, a FORTRAN variable, or a Java variable

or:

- A host language construct that was generated by an SQL precompiler from a variable declared using SQL extensions

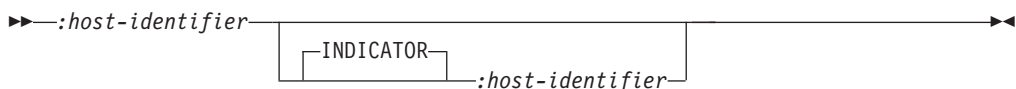
that is referenced in an SQL statement. Host variables are either directly defined by statements in the host language or are indirectly defined using SQL extensions.

A host variable in an SQL statement must identify a host variable described in the program according to the rules for declaring host variables.

All host variables used in an SQL statement must be declared in an SQL DECLARE section in all host languages except REXX. No variables may be declared outside an SQL DECLARE section with names identical to variables declared inside an SQL DECLARE section. An SQL DECLARE section begins with BEGIN DECLARE SECTION and ends with END DECLARE SECTION.

The meta-variable *host-variable*, as used in the syntax diagrams, shows a reference to a host variable. A host-variable as the target variable in a SET variable statement or in the INTO clause of a FETCH, SELECT INTO, or VALUES INTO statement, identifies a host variable to which a value from a column of a row or an expression is assigned. In all other contexts a host-variable specifies a value to be passed to the database manager from the application program.

The meta-variable *host-variable* in syntax diagrams can generally be expanded to:



Each *host-identifier* must be declared in the source program. The variable designated by the second host-identifier must have a data type of small integer.

The first host-identifier designates the *main variable*. Depending on the operation, it either provides a value to the database manager or is provided a value from the database manager. An input host variable provides a value in the runtime application code page. An output host variable is provided a value that, if necessary, is converted to the runtime application code page when the data is copied to the output application variable. A given host variable can serve as both an input and an output variable in the same program.



The second host-identifier designates its *indicator variable*. The purposes of the indicator variable are to:

- Specify the null value. A negative value of the indicator variable specifies the null value. A value of -2 indicates a numeric conversion or arithmetic expression error occurred in deriving the result
- Record the original length of a truncated string (if the source of the value is not a large object type)
- Record the seconds portion of a time if the time is truncated on assignment to a host variable.

For example, if :HV1:HV2 is used to specify an insert or update value, and if HV2 is negative, the value specified is the null value. If HV2 is not negative the value specified is the value of HV1.

Similarly, if :HV1:HV2 is specified in an INTO clause of a FETCH, SELECT INTO, or VALUES INTO statement, and if the value returned is null, HV1 is not changed, and HV2 is set to a negative value. If the database is configured with DFT\_SQLMATHWARN yes (or was during binding of a static SQL statement), HV2 could be -2. If HV2 is -2, a value for HV1 could not be returned because of an error converting to the numeric type of HV1, or an error evaluating an arithmetic expression that is used to determine the value for HV1. When accessing a database with a client version earlier than DB2 Universal Database Version 5, HV2 will be -1 for arithmetic exceptions. If the value returned is not null, that value is assigned to HV1 and HV2 is set to zero (unless the assignment to HV1 requires string truncation of a non-LOB string; in which case HV2 is set to the original length of the string). If an assignment requires truncation of the seconds part of a time, HV2 is set to the number of seconds.

If the second host identifier is omitted, the host-variable does not have an indicator variable. The value specified by the host-variable reference :HV1 is always the value of HV1, and null values cannot be assigned to the variable. Thus, this form should not be used in an INTO clause unless the corresponding column cannot contain null values. If this form is used and the column contains nulls, the database manager will generate an error at run time.

An SQL statement that references host variables must be within the scope of the declaration of those host variables. For host variables referenced in the SELECT statement of a cursor, that rule applies to the OPEN statement rather than to the DECLARE CURSOR statement.

## Example

Using the PROJECT table, set the host variable PNAME (VARCHAR(26)) to the project name (PROJNAME), the host variable STAFF (dec(5,2)) to the mean staffing level (PRSTAFF), and the host variable MAJPROJ (char(6)) to the major project (MAJPROJ) for project (PROJNO) 'IF1000'. Columns PRSTAFF and MAJPROJ may contain null values, so provide indicator variables STAFF\_IND (smallint) and MAJPROJ\_IND (smallint).

```
SELECT PROJNAME, PRSTAFF, MAJPROJ
INTO :PNAME, :STAFF :STAFF_IND, :MAJPROJ :MAJPROJ_IND
FROM PROJECT
WHERE PROJNO = 'IF1000'
```

**MBCS Considerations:** Whether multi-byte characters can be used in a host variable name depends on the host language.

## Host variables in dynamic SQL

In dynamic SQL statements, parameter markers are used instead of host variables. A parameter marker represents a position in a dynamic SQL statement where the application will provide a value; that is, where a host variable would be found if the statement string were a static SQL statement. The following example shows a static SQL statement using host variables:

```
INSERT INTO DEPARTMENT
VALUES (:hv_deptno, :hv_deptname, :hv_mgrno, :hv_admrdept)
```

This example shows a dynamic SQL statement using unnamed parameter markers:

```
INSERT INTO DEPARTMENT VALUES (?, ?, ?, ?)
```

This example shows a dynamic SQL statement using named parameter markers:

```
INSERT INTO DEPARTMENT
VALUES (:deptno, :deptname, :mgrno, :admrdept)
```

Named parameter markers can be used to improve the readability of dynamic statement. Although named parameter markers look like host variables, named parameter markers have no associated value and therefore a value must be provided for the parameter marker when the statement is executed. If the INSERT statement using named parameter markers has been prepared and given the prepared statement name of DYNSTMT, then values can be provided for the parameter markers using the following statement:

```
EXECUTE DYNSTMT
USING :hv_deptno, :hv_deptname :hv_mgrno, :hv_admrdept
```

This same EXECUTE statement could be used if the INSERT statement using unnamed parameter markers had been prepared and given the prepared statement name of DYNSTMT.

## References to BLOB, CLOB, and DBCLOB host variables

Regular BLOB, CLOB, and DBCLOB variables, LOB locator variables (see “References to locator variables”), and LOB file reference variables (see “References to BLOB, CLOB, and DBCLOB file reference variables” on page 585) can be defined in all host languages. Where LOBs are allowed, the term *host-variable* in a syntax diagram can refer to a regular host variable, a locator variable, or a file reference variable. Since these are not native data types, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable. In the case of REXX, LOBs are mapped to strings.

It is sometimes possible to define a large enough variable to hold an entire large object value. If this is true and if there is no performance benefit to be gained by deferred transfer of data from the server, a locator is not needed. However, since host language or space restrictions will often dictate against storing an entire large object in temporary storage at one time and/or because of performance benefit, a large object may be referenced via a locator and portions of that object may be selected into or updated from host variables that contain only a portion of the large object at one time.

## References to locator variables

A *locator variable* is a host variable that contains the locator representing a LOB value on the application server.

A locator variable in an SQL statement must identify a locator variable described in the program according to the rules for declaring locator variables. This is always indirectly through an SQL statement.

The term locator variable, as used in the syntax diagrams, shows a reference to a locator variable. The meta-variable *locator-variable* can be expanded to include a *host-identifier* the same as that for *host-variable*.

As with all other host variables, a large object locator variable may have an associated indicator variable. Indicator variables for large object locator host variables behave in the same way as indicator variables for other data types. When a null value is returned from the database, the indicator variable is set and the locator host variable is unchanged. This means a locator can never point to a null value.

If a locator-variable that does not currently represent any value is referenced, an error is raised (SQLSTATE 0F001).

At transaction commit, or any transaction termination, all locators acquired by that transaction are released.

## References to BLOB, CLOB, and DBCLOB file reference variables

BLOB, CLOB, and DBCLOB file reference variables are used for direct file input and output for LOBs, and can be defined in all host languages. Since these are not native data types, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable. In the case of REXX, LOBs are mapped to strings.

A file reference variable represents (rather than contains) the file, just as a LOB locator represents, rather than contains, the LOB bytes. Database queries, updates and inserts may use file reference variables to store or to retrieve single column values.

A file reference variable has the following properties:

### Data Type

BLOB, CLOB, or DBCLOB. This property is specified when the variable is declared.

### Direction

This must be specified by the application program at run time (as part of the File Options value). The direction is one of:

- Input (used as a source of data on an EXECUTE statement, an OPEN statement, an UPDATE statement, an INSERT statement, or a DELETE statement).
- Output (used as the target of data on a FETCH statement or a SELECT INTO statement).

### File name

This must be specified by the application program at run time. It is one of:

- The complete path name of the file (which is advised).
- A relative file name. If a relative file name is provided, it is appended to the current path of the client process.

Within an application, a file should only be referenced in one file reference variable.

### File Name Length

This must be specified by the application program at run time. It is the length of the file name (in bytes).

### File Options

An application must assign one of a number of options to a file reference variable before it makes use of that variable. Options are set by an INTEGER value in a field in the file reference variable structure. One of the following values must be specified for each file reference variable:

- Input (from client to server)

#### SQL\_FILE\_READ

This is a regular file that can be opened, read and closed. (The option is SQL-FILE-READ in COBOL, `sql_file_read` in FORTRAN, and READ in REXX.)

- Output (from server to client)

#### SQL\_FILE\_CREATE

Create a new file. If the file already exists, an error is returned. (The option is SQL-FILE-CREATE in COBOL, `sql_file_create` in FORTRAN, and CREATE in REXX.)

#### SQL\_FILE\_OVERWRITE (Overwrite)

If an existing file with the specified name exists, it is overwritten; otherwise a new file is created. (The option is SQL-FILE-OVERWRITE in COBOL, `sql_file_overwrite` in FORTRAN, and OVERWRITE in REXX.)

#### SQL\_FILE\_APPEND

If an existing file with the specified name exists, the output is appended to it; otherwise a new file is created. (The option is SQL-FILE-APPEND in COBOL, `sql_file_append` in FORTRAN, and APPEND in REXX.)

### Data Length

This is unused on input. On output, the implementation sets the data length to the length of the new data written to the file. The length is in bytes.

As with all other host variables, a file reference variable may have an associated indicator variable.

## Example of an output file reference variable (in C)

Given a declare section coded as:

```
EXEC SQL BEGIN DECLARE SECTION
SQL TYPE IS CLOB_FILE hv_text_file;
char hv_patent_title[64];
EXEC SQL END DECLARE SECTION
```

Following preprocessing this would be:

```
EXEC SQL BEGIN DECLARE SECTION
/* SQL TYPE IS CLOB_FILE hv_text_file; */
struct {
 unsigned long name_length; // File Name Length
 unsigned long data_length; // Data Length
 unsigned long file_options; // File Options
```

```

 char name[255]; // File Name
 } hv_text_file;
 char hv_patent_title[64];
EXEC SQL END DECLARE SECTION

```

Then, the following code can be used to select from a CLOB column in the database into a new file referenced by :hv\_text\_file.

```

strcpy(hv_text_file.name, "/u/gainer/papers/sigmod.94");
hv_text_file.name_length = strlen("/u/gainer/papers/sigmod.94");
hv_text_file.file_options = SQL_FILE_CREATE;

EXEC SQL SELECT content INTO :hv_text_file from papers
 WHERE TITLE = 'The Relational Theory behind Juggling';

```

### Example of an input file reference variable (in C)

Given the same declare section as above, the following code can be used to insert the data from a regular file referenced by :hv\_text\_file into a CLOB column.

```

strcpy(hv_text_file.name, "/u/gainer/patents/chips.13");
hv_text_file.name_length = strlen("/u/gainer/patents/chips.13");
hv_text_file.file_options = SQL_FILE_READ;
strcpy(:hv_patent_title, "A Method for Pipelining Chip Consumption");

EXEC SQL INSERT INTO patents(title, text)
 VALUES(:hv_patent_title, :hv_text_file);

```

### References to structured type host variables

Structured type variables can be defined in all host languages except FORTRAN, REXX, and Java. Since these are not native data types, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable.

As with all other host variables, a structured type variable may have an associated indicator variable. Indicator variables for structured type host variables behave in the same way as indicator variables for other data types. When a null value is returned from the database, the indicator variable is set and the structured type host variable is unchanged.

The actual host variable for a structured type is defined as a built-in data type. The built-in data type associated with the structured type must be assignable:

- from the result of the FROM SQL transform function for the structured type as defined by the specified TRANSFORM GROUP option of the precompile command; and
- to the parameter of the TO SQL transform function for the structured type as defined by the specified TRANSFORM GROUP option of the precompile command.

If using a parameter marker instead of a host variable, the appropriate parameter type characteristics must be specified in the SQLDA. This requires a "doubled" set of SQLVAR structures in the SQLDA, and the SQLDATATYPE\_NAME field of the secondary SQLVAR must be filled with the schema and type name of the structured type. If the schema is omitted in the SQLDA structure, an error results (SQLSTATE 07002).

## Example

Define the host variables *hv\_poly* and *hv\_point* (of type POLYGON, using built-in type BLOB(1048576)) in a C program.

```
EXEC SQL BEGIN DECLARE SECTION;
 static SQL
 TYPE IS POLYGON AS BLOB(1M)
 hv_poly, hv_point;
EXEC SQL END DECLARE SECTION;
```

## SQL path

The SQL path is an ordered list of schema names. The database manager uses the SQL path to resolve the schema name for unqualified data type names (both built-in types and distinct types), global variable names, module names, function names, and procedure names that appear in any context other than as the main object of a CREATE, DROP, COMMENT, GRANT or REVOKE statement. For details, see “Qualification of unqualified object names”.

For example, if the SQL path is SYSIBM, SYSFUN, SYSPROC, SYSIBMADM, SMITH, XGRAPHICS2 and an unqualified distinct type name MYTYPE was specified, the database manager looks for MYTYPE first in schema SYSIBM, then SYSFUN, then SYSPROC, then SYSIBMADM, then SMITH, and then XGRAPHICS2.

The SQL path used depends on the SQL statement:

- For static SQL statements (except for a CALL variable statement), the SQL path used is the SQL path specified when the containing package, procedure, function, trigger, or view was created.
- For dynamic SQL statements (and for a CALL variable statement), the SQL path is the value of the CURRENT PATH special register. CURRENT PATH can be set by the SET PATH statement.

If the SQL path is not explicitly specified, the SQL path is the system path followed by the authorization ID of the statement. .

## Qualification of unqualified object names

Unqualified object names are implicitly qualified. The rules for qualifying a name differ depending on the type of object that the name identifies.

### Unqualified alias, index, package, sequence, table, trigger, and view names

Unqualified alias, index, package, sequence, table, trigger, and view names are implicitly qualified by the default schema.

For static SQL statements, the default schema is the default schema specified when the containing function, package, procedure, or trigger was created.

For dynamic SQL statements, the default schema is the default schema specified for the application process. The default schema can be specified for the application process by using the SET SCHEMA statement. If the default schema is not explicitly specified, the default schema is the authorization ID of the statement.

## Unqualified user-defined type, function, procedure, specific, global variable and module names

The qualification of data type (both built-in types and distinct types), global variable, module, function, procedure, and specific names depends on the SQL statement in which the unqualified name appears:

- If an unqualified name is the main object of a CREATE, ALTER, COMMENT, DROP, GRANT, or REVOKE statement, the name is implicitly qualified using the same rules as for qualifying unqualified table names (See “Unqualified alias, index, package, sequence, table, trigger, and view names”). The main object of an ADD, COMMENT, DROP or PUBLISH operation of the ALTER MODULE statement must be specified without any qualifier.
- If the context of the reference is within a module, the database manager searches the module for the object, applying the appropriate resolution for the type of object to find a match. If no match is found, the search continues as specified in the next bullet.
- Otherwise, the implicit schema name is determined as follows:
  - For distinct type names, the database manager searches the SQL path and selects the first schema in the SQL path such that the data type exists in the schema.
  - For global variables, the database manager searches the SQL path and selects the first schema in the SQL path such that the global variable exists in the schema.
  - For procedure names, the database manager uses the SQL path in conjunction with procedure resolution.
  - For function names, the database manager uses the SQL path in conjunction with function resolution .
  - For specific names specified for sourced functions, see “CREATE FUNCTION (Sourced)”.

## Resolving qualified object names

Objects that are defined in a module that are available for use outside the module must be qualified by the module name. Since a module is a schema object that can also be implicitly qualified, the published module objects can be qualified using an unqualified module name or a schema-qualified module name. When an unqualified module name is used, the reference to the module object appears the same as a schema-qualified object that is not part of a module. Within a specific scope, such as a compound SQL statement, a two-part identifier could also be:

- a column name qualified by a table name
- a row field name qualified by a variable name
- a variable name qualified by a label
- a routine parameter name qualified by a routine name

These objects are resolved within their scope, before considering either schema objects or module object. The following process is used to resolve objects with two-part identifiers that could be a schema object or a module object.

- If the context of the reference is within a module and the qualifier matches the module name, the database manager searches the module for the object, applying the appropriate resolution for the type of object to find a match among published and unpublished module objects. If no match is found, the search continues as specified in the next bullets.



- Assume that the qualifier is a schema name and, if the schema exists, resolve the object in the schema.
- If the qualifier is not an existing schema or the object is not found in the schema that matches the qualifier and the qualifier did not match the context module name, search for the first module that matches the qualifier in the schemas on the SQL path. If authorized to the matching module, resolve to the object in that module, considering only published module objects.
- If the qualifier is not found as a module on the SQL path and the qualifier did not match the context module name, check for a module public synonym that matches the qualifier. If found, resolve the object in the module identified by the module public synonym, considering only published module objects.

---

## Part 4. Security plug-ins

Only the default IBM-supplied operating-system based authentication and group plug-ins are supported in Common Criteria compliant environments. User-written or third-party plug-ins are not supported. In addition, Kerberos-based authorization is not supported. The sections that follow are for informational purposes only.



---

## Chapter 9. An overview of security plug-ins

---

### Security plug-ins

Authentication for the DB2 database system is done using *security plug-ins*. A security plug-in is a dynamically-loadable library that provides authentication security services.

The DB2 database system provides the following types of plug-ins:

- Group retrieval plug-in: retrieves group membership information for a given user.
- Client authentication plug-in: manages authentication on a DB2 client.
- Server authentication plug-in: manages authentication on a DB2 server.

The DB2 database manager supports two mechanisms for plug-in authentication:

#### User ID/password authentication

This involves authentication using a user ID and password. The following authentication types are implemented using user ID/password authentication plug-ins:

- CLIENT
- SERVER
- SERVER\_ENCRYPT
- DATA\_ENCRYPT
- DATA\_ENCRYPT\_CMP

These authentication types determine how and where authentication of a user occurs. The authentication type used depends on the authentication type specified by the *authentication* database manager configuration parameter. If the SRVCON\_AUTH parameter is specified it takes precedence over AUTHENTICATION when dealing with connect or attach operations.

#### GSS-API authentication

GSS-API is formally known as *Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) and *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744). Kerberos authentication is also implemented using GSS-API. The following authentication types are implemented using GSS-API authentication plug-ins:

- KERBEROS
- GSSPLUGIN
- KRB\_SERVER\_ENCRYPT
- GSS\_SERVER\_ENCRYPT

KRB\_SERVER\_ENCRYPT and GSS\_SERVER\_ENCRYPT support both GSS-API authentication and user ID/password authentication; however, GSS-API authentication is the preferred authentication type.

**Note:** Authentication types determine how and where a user is authenticated. To use a particular authentication type, update the authentication database manager configuration parameter.

Each of the plug-ins can be used independently or in conjunction with one or more of the other plug-ins. For example, you might only use a server authentication plug-in and assume the DB2 defaults for client and group authentication. Alternatively, you might have only a group or client authentication plug-in. The only situation where both a client and server plug-in are required is for GSS-API authentication plug-ins.

The default behavior is to use a user ID/password plug-in that implements an operating-system-level mechanism for authentication. In previous releases, the default behavior is to directly use operating-system-level authentication without a plug-in implementation. Client-side Kerberos support is available on Solaris, AIX, Windows, and Linux operating systems. For Windows platforms, Kerberos support is enabled by default.

DB2 database systems include sets of plug-ins for group retrieval, user ID/password authentication, and for Kerberos authentication. With the security plug-in architecture you can customize DB2 client and server authentication behavior by either developing your own plug-ins, or buying plug-ins from a third party.

## Deployment of security plug-ins on DB2 clients

DB2 clients can support one group plug-in, one user ID/password authentication plug-in, and will negotiate with the DB2 server for a particular GSS-API plug-in. This negotiation consists of a scan by the client of the DB2 server's list of implemented GSS-API plug-ins for the first authentication plug-in name that matches an authentication plug-in implemented on the client. The server's list of plug-ins is specified in the *srvcon\_gssplugin\_list* database manager configuration parameter value, which contains the names of all of the plug-ins that are implemented on the server. The following figure portrays the security plug-in infrastructure on a DB2 client.

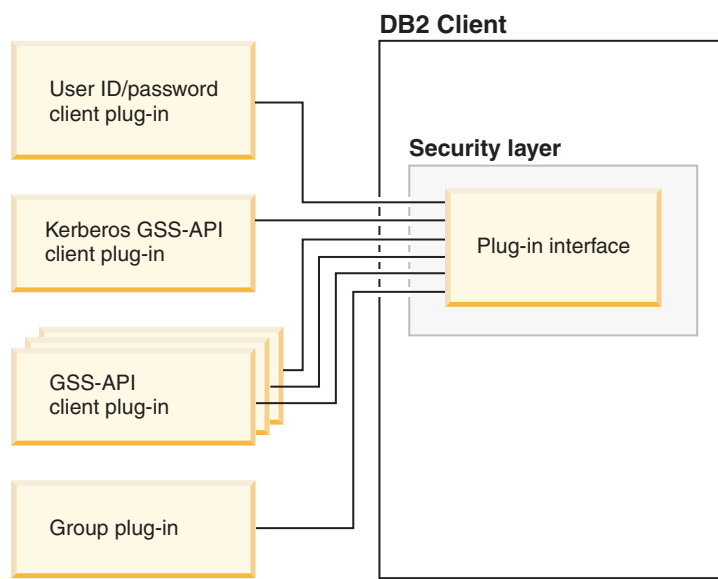


Figure 7. Deploying Security Plug-ins on DB2 Clients

## Deployment of security plug-ins on DB2 servers

DB2 servers can support one group plug-in, one user ID/password authentication plug-in, and multiple GSS-API plug-ins. The multiple GSS-API plug-ins are specified in the *svrcon\_gssplugin\_list* database manager configuration parameter value as a list. Only one GSS-API plug-in in this list can be a Kerberos plug-in.

In addition to server-side security plug-ins, you might also need to deploy client authorization plug-ins on your database server. When you run instance-level operations like *db2start* and *db2trc*, the DB2 database manager performs authorization checking for these operations using client authentication plug-ins. Therefore, you should install the client authentication plug-in that corresponds to the server plug-in that is specified by the *authentication* database manager configuration parameter. There is a main distinction between *authentication* and *svrcon\_auth*. Specifically, they could be set to different values to cause one mechanism to be used to authenticate database connections and another mechanism to be used for local authorization. The most common usage is *svrcon\_auth* set as *GSSPLUGIN* and *authentication* set as *SERVER*. If you do not use client authentication plug-ins on the database server, instance level operations such as *db2start* will fail. For example, if the authentication type is *SERVER* and no user-supplied client plug-in is used, the DB2 database system will use the IBM-shipped default client operating-system plug-in. The following figure portrays the security plug-in infrastructure on a DB2 server.

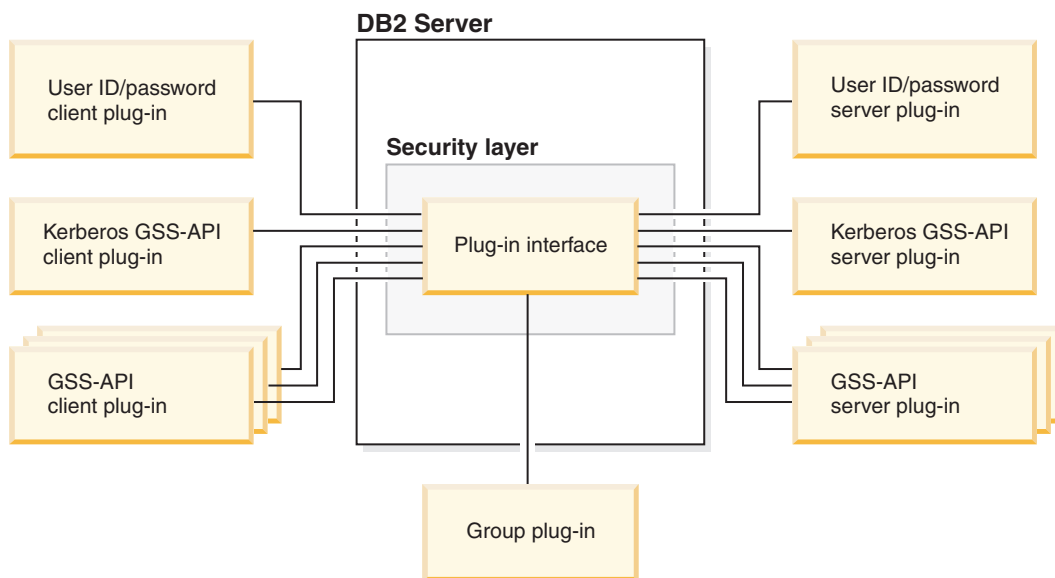


Figure 8. Deploying Security Plug-ins on DB2 Servers

**Note:** The integrity of your DB2 database system installation can be compromised if the deployment of security plug-ins are not adequately coded, reviewed, and tested. The DB2 database system takes precaution against many common types of failures, but it cannot guarantee complete integrity when user-written security plug-ins are deployed.

### Enabling security plug-ins

The system administrator can specify the names of the plug-ins to use for each authentication mechanism by updating certain plug-in-related database manager configuration parameters. If these parameters are null, they will default to the

DB2-supplied plug-ins for group retrieval, user ID/password management, or Kerberos (if authentication is set to Kerberos -- on the server). DB2 does not provide a default GSS-API plug-in. Therefore, if system administrators specify an authentication type of GSSPLUGIN in *authentication* parameter, they must also specify a GSS-API authentication plug-in in *srocon\_gssplugin\_list*.

## How DB2 loads security plug-ins

All of the supported plug-ins identified by the database manager configuration parameters are loaded when the database manager starts.

The DB2 client will load a plug-in appropriate for the security mechanism negotiated with the server during connect or attach operations. It is possible that a client application can cause multiple security plug-ins to be concurrently loaded and used. This situation can occur, for example, in a threaded program that has concurrent connections to different databases from different instances.

Actions other than connect or attach operations require authorization (such as updating the database manager configuration, starting and stopping the database manager, turning DB2 trace on and off) as well. For such actions, the DB2 client program will load a plug-in specified in another database manager configuration parameter. If *authentication* is set to GSSPLUGIN, DB2 database manager will use the plug-in specified by *local\_gssplugin*. If *authentication* is set to KERBEROS, DB2 database manager will use the plug-in specified by *clnt\_krb\_plugin*. Otherwise, DB2 database manager will use the plug-in specified by *clnt\_pw\_plugin*.

Security plug-ins APIs can be called from either an IPv4 platform or an IPv6 platform. An IPv4 address is a 32-bit address that has a readable form a.b.c.d, where each of a-d represents a decimal number from 0-255. An IPv6 address is a 128-bit address of the form a:b:c:d:e:f:g:h, where each of a-h represents 4 hex digits.

## Developing security plug-ins

If you are developing a security plug-in, you need to implement the standard authentication functions that DB2 database manager will use. If you are using your own customized security plug-in, you can use a user ID of up to 255 characters on a connect statement issued through the CLP or a dynamic SQL statement. For the available types of plug-ins, the functionality you will need to implement is as follows:

### Group retrieval

Gets the list of groups to which a user belongs.

### User ID/password authentication

- Identifies the default security context (client only).
- Validates and optionally changes a password.
- Determines if a given string represents a valid user (server only).
- Modifies the user ID or password provided on the client before it is sent to the server (client only).
- Returns the DB2 authorization ID associated with a given user.

### GSS-API authentication

- Implements the required GSS-API functions.
- Identifies the default security context (client only).



- Generates initial credentials based on a user ID and password and optionally changes password (client only).
- Creates and accepts security tickets.
- Returns the DB2 authorization ID associated with a given GSS-API security context.

---

## Security plug-in library locations

After you acquire your security plug-ins (either by developing them yourself, or purchasing them from a third party), copy them to specific locations on your database server.

DB2 clients look for client-side user authentication plug-ins in the following directory:

- UNIX 32-bit: `$DB2PATH/security32/plugin/client`
- UNIX 64-bit: `$DB2PATH/security64/plugin/client`
- WINDOWS 32-bit and 64-bit: `$DB2PATH\security\plugin\instance name\client`

**Note:** On Windows-based platforms, the subdirectories *instance name* and *client* are not created automatically. The instance owner has to manually create them.

The DB2 database manager looks for server-side user authentication plug-ins in the following directory:

- UNIX 32-bit: `$DB2PATH/security32/plugin/server`
- UNIX 64-bit: `$DB2PATH/security64/plugin/server`
- WINDOWS 32-bit and 64-bit: `$DB2PATH\security\plugin\instance name\server`

**Note:** On Windows-based platforms, the subdirectories *instance name* and *server* are not created automatically. The instance owner has to manually create them.

The DB2 database manager looks for group plug-ins in the following directory:

- UNIX 32-bit: `$DB2PATH/security32/plugin/group`
- UNIX 64-bit: `$DB2PATH/security64/plugin/group`
- WINDOWS 32-bit and 64-bit: `$DB2PATH\security\plugin\instance name\group`

**Note:** On Windows-based platforms, the subdirectories *instance name* and *group* are not created automatically. The instance owner has to manually create them.

---

## Security plug-in naming conventions

Security plug-in libraries must have a platform-specific file name extension. Security plug-in libraries written in C or C++ must have a platform-specific file name extension:

- Windows: `.dll`
- AIX: `.a` or `.so`, and if both extensions exist, `.a` extension is used.
- Linux, HP IPF and Solaris: `.so`
- HP-UX on PA-RISC: `.sl` or `.so`, and if both extensions exist, `.sl` extension is used.

**Note:** Users can also develop security plug-ins with the DB2 Universal JDBC Driver.

For example, assume you have a security plug-in library called MyPlugin. For each supported operating system, the appropriate library file name follows:

- Windows 32-bit: MyPlugin.dll
- Windows 64-bit: MyPlugin64.dll
- AIX 32 or 64-bit: MyPlugin.a or MyPlugin.so
- SUN 32 or 64-bit, Linux 32 or 64 bit, HP 32 or 64 bit on IPF: MyPlugin.so
- HP-UX 32 or 64-bit on PA-RISC: MyPlugin.sl or MyPlugin.so

**Note:** The suffix "64" is only required on the library name for 64-bit Windows security plug-ins.

When you update the database manager configuration with the name of a security plug-in, use the full name of the library without the "64" suffix and omit both the file extension and any qualified path portion of the name. Regardless of the operating system, a security plug-in library called MyPlugin would be registered as follows:

```
UPDATE DBM CFG USING CLNT_PW_PLUGIN MyPlugin
```

The security plug-in name is case sensitive, and must exactly match the library name. DB2 database systems use the value from the relevant database manager configuration parameter to assemble the library path, and then uses the library path to load the security plug-in library.

To avoid security plug-in name conflicts, you should name the plug-in using the authentication method used, and an identifying symbol of the firm that wrote the plug-in. For instance, if the company Foo, Inc. wrote a plug-in implementing the authentication method F00sometmethod, the plug-in could have a name like F00sometmethod.dll.

The maximum length of a plug-in name (not including the file extension and the "64" suffix) is limited to 32 bytes. There is no maximum number of plug-ins supported by the database server, but the maximum length of the comma-separated list of plug-ins in the database manager configuration is 255 bytes. Two defines located in the include file `sqlenv.h` identifies these two limits:

```
#define SQL_PLUGIN_NAME_SZ 32 /* plug-in name */
#define SQL_SRVCON_GSSPLUGIN_LIST_SZ 255 /* GSS API plug-in list */
```

The security plug-in library files must have the following file permissions:

- Owned by the instance owner.
- Readable by all users on the system.
- Executable by all users on the system.

---

## Security plug-in support for two-part user IDs

The DB2 database manager on Windows supports the use of two-part user IDs, and the mapping of two-part user IDs to two-part authorization IDs.

For example, consider a Windows operating system two-part user ID composed of a domain and user ID such as: MEDWAY\pieter. In this example, MEDWAY is a domain and pieter is the user name. In DB2 database systems, you can specify whether this two-part user ID should be mapped to either a one-part authorization ID or a two-part authorization ID.

The mapping of a two-part user ID to a two-part authorization ID is supported, but is not the default behavior. By default, both one-part user IDs and two-part user IDs map to one-part authorization IDs. The mapping of a two-part user ID to a two-part authorization ID is supported, but is not the default behavior.

The default mapping of a two-part user ID to a one-part user ID allows a user to connect to the database using:

```
db2 connect to db user MEDWAY\pieter using pw
```

In this situation, if the default behavior is used, the user ID MEDWAY\pieter is resolved to the authorization ID PIETER. If the support for mapping a two-part user ID to a two-part authorization ID is enabled, the authorization ID would be MEDWAY\PIETER.

To enable DB2 to map two-part user IDs to two-part authorization IDs, DB2 supplies two sets of authentication plug-ins:

- One set exclusively maps a one-part user ID to a one-part authorization ID and maps a two-part user-ID to a one-part authorization ID.
- Another set maps both one-part user ID or two-part user ID to a two-part authorization ID.

If a user name in your work environment can be mapped to multiple accounts defined in different locations (such as local account, domain account, and trusted domain accounts), you can specify the plug-ins that enable two-part authorization ID mapping.

It is important to note that a one-part authorization ID, such as, PIETER and a two-part authorization ID that combines a domain and a user ID like MEDWAY\pieter are functionally distinct authorization IDs. The set of privileges associated with one of these authorization IDs can be completely distinct from the set of privileges associated with the other authorization ID. Care should be taken when working with one-part and two-part authorization IDs.

The following table identifies the kinds of plug-ins supplied by DB2 database systems, and the plug-in names for the specific authentication implementations.

*Table 31. DB2 security plug-ins*

| Authentication type       | Name of one-part user ID plug-in | Name of two-part user ID plug-in |
|---------------------------|----------------------------------|----------------------------------|
| User ID/password (client) | IBMOSauthclient                  | IBMOSauthclientTwoPart           |
| User ID/password (server) | IBMOSauthserver                  | IBMOSauthserverTwoPart           |
| Kerberos                  | IBMkrb5                          | IBMkrb5TwoPart                   |

**Note:** On Windows 64-bit platforms, the characters "64" are appended to the plug-in names listed here.

When you specify an authentication type that requires a user ID/password or Kerberos plug-in, the plug-ins that are listed in the "Name of one-part user ID plug-in" column in the previous table are used by default.

To map a two-part user ID to a two-part authorization ID, you must specify that the two-part plug-in, which is not the default plug-in, be used. Security plug-ins are specified at the instance level by setting the security related database manager configuration parameters as follows:

For server authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_pw_plugin` to `IBMOSauthserverTwoPart`
- `clnt_pw_plugin` to `IBMOSauthclientTwoPart`

For client authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_pw_plugin` to `IBMOSauthserverTwoPart`
- `clnt_pw_plugin` to `IBMOSauthclientTwoPart`

For Kerberos authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_gssplugin_list` to `IBMOSkrb5TwoPart`
- `clnt_krb_plugin` to `IBMkrb5TwoPart`

The security plug-in libraries accept two-part user IDs specified in a Microsoft Windows Security Account Manager compatible format. For example, in the format: *domain\user ID*. Both the domain and user ID information will be used by the DB2 authentication and authorization processes at connection time.

You should consider implementing the two-part plug-ins when creating new databases to avoid conflicts with one-part authorization IDs in existing databases. New databases that use two-part authorization IDs must be created in a separate instance from databases that use single-part authorization IDs.

---

## 32-bit and 64-bit considerations for security plug-ins

In general, a 32-bit DB2 instance uses the 32-bit security plug-in and a 64-bit DB2 instance uses the 64-bit security plug-in. However, on a 64-bit instance, DB2 supports 32-bit applications, which require the 32-bit plug-in library.

A database instance where both the 32-bit and the 64-bit applications can run is known as a hybrid instance. If you have a hybrid instance and intend to run 32-bit applications, ensure that the required 32-bit security plug-ins are available in the 32-bit plug-in directory. For 64-bit DB2 instances on Linux and UNIX operating systems, excluding Linux on IPF, the directories `security32` and `security64` appear. For a 64-bit DB2 instance on Windows on x64 or IPF, both 32-bit and 64-bit security plug-ins are located in the same directory, but 64-bit plug-in names have a suffix, "64".

If you want to upgrade from a 32-bit instance to a 64-bit instance, you should obtain versions of your security plug-ins that are recompiled for 64-bit.

If you acquired your security plug-ins from a vendor that does not supply 64-bit plug-in libraries, you can implement a 64-bit stub that executes a 32-bit application. In this situation, the security plug-in is an external program rather than a library.

---

## Security plug-in problem determination

Problems with security plug-ins are reported in two ways: through SQL errors and through the administration notification log.

Following are the SQLCODE values related to security plug-ins:

- SQLCODE -1365 is returned when a plug-in error occurs during db2start or db2stop.
- SQLCODE -1366 is returned whenever there is a local authorization problem.
- SQLCODE -30082 is returned for all connection-related plug-in errors.

The administration notification logs are a good resource for debugging and administrating security plug-ins. To see the an administration notification log file on UNIX, check `sqlllib/db2dump/instance name.N.nfy`. To see an administration notification log on Windows operating systems, use the Event Viewer tool. The Event Viewer tool can be found by navigating from the Windows operating system "Start" button to Settings -> Control Panel -> Administrative Tools -> Event Viewer. Following are the administration notification log values related to security plug-ins:

- 13000 indicates that a call to a GSS-API security plug-in API failed with an error, and returned an optional error message.

```
SQLT_ADMIN_GSS_API_ERROR (13000)
Plug-in "plug-in name" received error code "error code" from
GSS API "gss api name" with the error message "error message"
```

- 13001 indicates that a call to a DB2 security plug-in API failed with an error, and returned an optional error message.

```
SQLT_ADMIN_PLUGIN_API_ERROR(13001)
Plug-in "plug-in name" received error code "error code" from DB2
security plug-in API "gss api name" with the error message
"error message"
```

- 13002 indicates that DB2 failed to unload a plug-in.

```
SQLT_ADMIN_PLUGIN_UNLOAD_ERROR (13002)
Unable to unload plug-in "plug-in name". No further action required.
```

- 13003 indicates a bad principal name.

```
SQLT_ADMIN_INVALID_PRIN_NAME (13003)
The principal name "principal name" used for "plug-in name"
is invalid. Fix the principal name.
```

- 13004 indicates that the plug-in name is not valid. Path separators (On UNIX "/" and on Windows "\\") are not allowed in the plug-in name.

```
SQLT_ADMIN_INVALID_PLGN_NAME (13004)
The plug-in name "plug-in name" is invalid. Fix the plug-in name.
```

- 13005 indicates that the security plug-in failed to load. Ensure the plug-in is in the correct directory and that the appropriate database manager configuration parameters are updated.

```
SQLT_ADMIN_PLUGIN_LOAD_ERROR (13005)
Unable to load plug-in "plug-in name". Verify the plug-in existence and
directory where it is located is correct.
```

- 13006 indicates that an unexpected error was encountered by a security plug-in. Gather all the db2support information, if possible capture a db2trc, and then call IBM support for further assistance.

```
SQLT_ADMIN_PLUGIN_UNEXP_ERROR (13006)
Plug-in encountered unexpected error. Contact IBM Support for further assistance.
```

**Note:** If you are using security plug-ins on a Windows 64-bit database server and are seeing a load error for a security plug-in, see the topics about 32-bit and 64-bit considerations and security plug-in naming conventions. The 64-bit plug-in library requires the suffix "64" on the library name, but the entry in the security plug-in database manager configuration parameters should not indicate this suffix.

---

## Deploying a group retrieval plug-in

To customize the DB2 security system's group retrieval behavior, you can develop your own group retrieval plug-in or buy one from a third party.

After you acquire a group retrieval plug-in that is suitable for your database management system, you can deploy it.

- To deploy a group retrieval plug-in on the database server, perform the following steps:
  1. Copy the group retrieval plug-in library into the server's group plug-in directory.
  2. Update the database manager configuration parameter *group\_plugin* with the name of the plug-in.
- To deploy a group retrieval plug-in on database clients, perform the following steps:
  1. Copy the group retrieval plug-in library in the client's group plug-in directory.
  2. On the database client, update the database manager configuration parameter *group\_plugin* with the name of the plug-in.

---

## Deploying a user ID/password plug-in

To customize the DB2 security system's user ID/password authentication behavior, you can develop your own user ID/password authentication plug-ins or buy one from a third party.

Depending on their intended usage, all user ID-password based authentication plug-ins must be placed in either the client plug-in directory or the server plug-in directory. If a plug-in is placed in the client plug-in directory, it will be used both for local authorization checking and for validating the client when it attempts to connect with the server. If the plug-in is placed in the server plug-in directory, it will be used for handling incoming connections to the server and for checking whether an authorization ID exists and is valid whenever the GRANT statement is issued without specifying either the keyword USER or GROUP. In most situations, user ID/password authentication requires only a server-side plug-in. It is possible, though generally deemed less useful, to have only a client user ID/password plug-in. It is possible, though quite unusual to require matching user ID/password plug-ins on both the client and the server.

**Note:** You must stop the DB2 server or any applications using the plug-ins before you deploy a *new* version of an *existing* plug-in. Undefined behavior including traps will occur if a process is still using a plug-in when a new version (with the same name) is copied over it. This restriction is not in effect when you deploy a plugin for the first time or when the plug-in is not in use.

After you acquire user ID/password authentication plug-ins that are suitable for your database management system, you can deploy them.

- To deploy a user ID/password authentication plug-in on the database server, perform the following steps on the database server:
  1. Copy the user ID/password authentication plug-in library in the server plug-in directory.
  2. Update the database manager configuration parameter *srvcon\_pw\_plugin* with the name of the server plug-in. This plug-in is used by the server when it is handling CONNECT and ATTACH requests.



3. Either:
  - Set the database manager configuration parameter *srvcon\_auth* to the CLIENT, SERVER, SERVER\_ENCRYPT, DATA\_ENCRYPT, or DATA\_ENCRYPT\_CMP authentication type. Or:
  - Set the database manager configuration parameter *srvcon\_auth* to NOT\_SPECIFIED and set *authentication* to CLIENT, SERVER, SERVER\_ENCRYPT, DATA\_ENCRYPT, or DATA\_ENCRYPT\_CMP authentication type.
- To deploy a user ID/password authentication plug-in on database clients, perform the following steps on each client:
  1. Copy the user ID/password authentication plug-in library in the client plug-in directory.
  2. Update the database manager configuration parameter *clnt\_pw\_plugin* with the name of the client plug-in. This plug-in is loaded and called regardless of where the authentication is being done, not only when the database configuration parameter, *authentication* is set to CLIENT.
- For local authorization on a client, server, or gateway using a user ID/password authentication plug-in, perform the following steps on each client, server, or gateway:
  1. Copy the user ID/password authentication plug-in library in the client plug-in directory on the client, server, or gateway.
  2. Update the database manager configuration parameter *clnt\_pw\_plugin* with the name of the plug-in.
  3. Set the *authentication* database manager configuration parameter to CLIENT, SERVER, SERVER\_ENCRYPT, DATA\_ENCRYPT, or DATA\_ENCRYPT\_CMP.

---

## Deploying a GSS-API plug-in

To customize the DB2 security system's authentication behavior, you can develop your own authentication plug-ins using the GSS-API, or buy one from a third party.

In the case of plug-in types other than Kerberos, you must have matching plug-in names on the client and the server along with the same plug-in type. The plug-ins on the client and server need not be from the same vendor, but they must generate and consume compatible GSS-API tokens. Any combination of Kerberos plug-ins deployed on the client and the server is acceptable since Kerberos plug-ins are standardized. However, different implementations of less standardized GSS-API mechanisms, such as *x.509* certificates, might only be partially compatible with DB2 database systems. Depending on their intended usage, all GSS-API authentication plug-ins must be placed in either the client plug-in directory or the server plug-in directory. If a plug-in is placed in the client plug-in directory, it will be used for local authorization checking and when a client attempts to connect with the server. If the plug-in is placed in the server plug-in directory, it will be used for handling incoming connections to the server and for checking whether an authorization ID exists and is valid whenever the GRANT statement is issued without specifying either the keyword USER or GROUP.

**Note:** You must stop the DB2 server or any applications using the plug-ins before you deploy a *new* version of an *existing* plug-in. Undefined behavior including traps will occur if a process is still using a plug-in when a new version (with the same name) is copied over it. This restriction is not in effect when you deploy a plugin for the first time or when the plug-in is not in use.



After you acquire GSS-API authentication plug-ins that are suitable for your database management system, you can deploy them.

- To deploy a GSS-API authentication plug-in on the database server, perform the following steps on the server:
  1. Copy the GSS-API authentication plug-in library in the server plug-in directory. You can copy numerous GSS-API plug-ins into this directory.
  2. Update the database manager configuration parameter *srvcn\_gssplugin\_list* with an ordered, comma-delimited list of the names of the plug-ins installed in the GSS-API plug-in directory.
  3. Either:
    - Setting the database manager configuration parameter *srvcn\_auth* to GSSPLUGIN or GSS\_SERVER\_ENCRYPT is a way to enable the server to use GSSAPI PLUGIN authentication method. Or:
    - Setting the database manager configuration parameter *srvcn\_auth* to NOT\_SPECIFIED and setting *authentication* to GSSPLUGIN or GSS\_SERVER\_ENCRYPT is a way to enable the server to use GSSAPI PLUGIN authentication method.
- To deploy a GSS-API authentication plug-in on database clients, perform the following steps on each client:
  1. Copy the GSS-API authentication plug-in library in the client plug-in directory. You can copy numerous GSS-API plug-ins into this directory. The client selects a GSS-API plug-in for authentication during a CONNECT or ATTACH operation by picking the first GSS-API plug-in contained in the server's plug-in list that is available on the client.
  2. Optional: Catalog the databases that the client will access, indicating that the client will only accept a GSS-API authentication plug-in as the authentication mechanism. For example:

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION GSSPLUGIN
```
- For local authorization on a client, server, or gateway using a GSS-API authentication plug-in, perform the following steps:
  1. Copy the GSS-API authentication plug-in library in the client plug-in directory on the client, server, or gateway.
  2. Update the database manager configuration parameter *local\_gssplugin* with the name of the plug-in.
  3. Set the *authentication* database manager configuration parameter to GSSPLUGIN, or GSS\_SERVER\_ENCRYPT.

---

## Deploying a Kerberos plug-in

To customize the DB2 security system's Kerberos authentication behavior, you can develop your own Kerberos authentication plug-ins or buy one from a third party. Note that the Kerberos security plug-in will not support IPv6.

**Note:** You must stop the DB2 server or any applications using the plug-ins before you deploy a *new* version of an *existing* plug-in. Undefined behavior including traps will occur if a process is still using a plug-in when a new version (with the same name) is copied over it. This restriction is not in effect when you deploy a plug-in for the first time or when the plug-in is not in use.

After you acquire Kerberos authentication plug-ins that are suitable for your database management system, you can deploy them.

- To deploy a Kerberos authentication plug-in on the database server, perform the following steps on the server:
  1. Copy the Kerberos authentication plug-in library in the server plug-in directory.
  2. Update the database manager configuration parameter **srvcon\_gssplugin\_list**, which is presented as an ordered, comma delimited list, to include the Kerberos server plug-in name. Only one plug-in in this list can be a Kerberos plug-in. If this list is blank and **authentication** is set to **KERBEROS** or **KRB\_SVR\_ENCRYPT**, the default DB2 Kerberos plug-in: **IBMkrb5** will be used.
  3. If necessary, set the **srvcon\_auth** database manager configuration parameter to override the current authentication type. If the **srvcon\_auth** database manager configuration parameter is not set, the DB2 database manager uses the value of the **authentication** configuration parameter. If the **authentication** configuration parameter is currently set to any of the following authentication types, you can deploy and use a Kerberos plug-in:
    - **KERBEROS**
    - **KRB\_SERVER\_ENCRYPT**
    - **GSSPLUGIN**
    - **GSS\_SERVER\_ENCRYPT**

If you need to override the current authentication type, set the **srvcon\_auth** configuration parameter to one of the following authentication types:

    - **KERBEROS**
    - **KRB\_SERVER\_ENCRYPT**
    - **GSSPLUGIN**
    - **GSS\_SERVER\_ENCRYPT**
- To deploy a Kerberos authentication plug-in on database clients, perform the following steps on each client:
  1. Copy the Kerberos authentication plug-in library in the client plug-in directory.
  2. Update the database manager configuration parameter **clnt\_krb\_plugin** with the name of the Kerberos plug-in. If **clnt\_krb\_plugin** is blank, DB2 assumes that the client cannot use Kerberos authentication. This setting is only appropriate when the server cannot support plug-ins. If both the server and the client support security plug-ins, the default server plug-in, *IBMkrb5* would be used over the client value of **clnt\_krb\_plugin**. For local authorization on a client, server, or gateway using a Kerberos authentication plug-in, perform the following steps:
    - a. Copy the Kerberos authentication plug-in library in the client plug-in directory on the client, server, or gateway.
    - b. Update the database manager configuration parameter **clnt\_krb\_plugin** with the name of the plug-in.
    - c. Set the **authentication** database manager configuration parameter to **KERBEROS**, or **KRB\_SERVER\_ENCRYPT**.
  3. Optional: Catalog the databases that the client will access, indicating that the client will only use a Kerberos authentication plug-in. For example:
 

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION KERBEROS
 TARGET PRINCIPAL service/host@REALM
```

**Note:** For platforms supporting Kerberos, the IBMkrb5 library will be present in the client plug-in directory. The DB2 database manager recognizes this library as a valid GSS-API plug-in, because Kerberos plug-ins are implemented using GSS-API plug-in.

---

## Restrictions on security plug-ins

There are certain restrictions on the use of security plug-ins.

### DB2 database family support restrictions

You cannot use a GSS-API plug-in to authenticate connections between DB2 clients on Linux, UNIX, and Windows and another DB2 family servers such as DB2 for z/OS. You also cannot authenticate connections from another DB2 database family product, acting as a client, to a DB2 server on Linux, UNIX, or Windows.

If you use a DB2 client on Linux, UNIX, or Windows to connect to other DB2 database family servers, you can use client-side user ID/password plug-ins (such as the IBM-shipped operating system authentication plug-in), or you can write your own user ID/password plug-in. You can also use the built-in Kerberos plug-ins, or implement your own.

With a DB2 client on Linux, UNIX, or Windows, you should not catalog a database using the GSSPLUGIN authentication type.

**Restrictions on the AUTHID identifier.** Version 9.5, and later, of the DB2 database system allows you to have an 128-byte authorization ID, but when the authorization ID is interpreted as an operating system user ID or group name, the operating system naming restrictions apply (for example, a limitation to 8 or 30 character user IDs and 30 character group names). Therefore, while you can grant an 128-byte authorization ID, it is not possible to connect as a user that has that authorization ID. If you write your own security plugin, you should be able to take full advantage of the extended sizes for the authorization ID. For example, you can give your security plugin a 30-byte user ID and it can return an 128-byte authorization ID during authentication that you are able to connect with.

### InfoSphere Federation Server support restrictions

DB2 II does not support the use of delegated credentials from a GSS\_API plug-in to establish outbound connections to data sources. Connections to data sources must continue to use the CREATE USER MAPPING command.

### Database Administration Server support restrictions

The DB2 Administration Server (DAS) does not support security plug-ins. The DAS only supports the operating system authentication mechanism.

### Security plug-in problem and restriction for DB2 clients (Windows)

When developing security plug-ins that will be deployed in DB2 clients on Windows operating systems, do not unload any auxiliary libraries in the plug-in termination function. This restriction applies to all types of client security plug-ins, including group, user ID and password, Kerberos, and GSS-API plug-ins. Since these termination APIs such as db2secPluginTerm, db2secClientAuthPluginTerm

and `db2secServerAuthPluginTerm` are not called on any Windows platform, you need to do the appropriate resource cleanup.

This restriction is related to cleanup issues associated with the unloading of DLLs on Windows.

## Loading plug-in libraries on AIX with extension of `.a` or `.so`

On AIX, security plug-in libraries can have a file name extension of `.a` or `.so`. The mechanism used to load the plug-in library depends on which extension is used:

- Plug-in libraries with a file name extension of `.a`

Plug-in libraries with file name extensions of `.a` are assumed to be archives containing shared object members. These members must be named `shr.o` (32-bit) or `shr64.o` (64-bit). A single archive can contain both the 32-bit and 64-bit members, allowing it to be deployed on both types of platforms.

For example, to build a 32-bit archive style plug-in library:

```
xlc_r -qmkshrobj -o shr.o MyPlugin.c -bE:MyPlugin.exp
ar rv MyPlugin.a shr.o
```

- Plug-in libraries with a file name extension of `.so`

Plug-in libraries with file name extensions of `.so` are assumed to be dynamically loadable shared objects. Such an object is either 32-bit or 64-bit, depending on the compiler and linker options used when it was built. For example, to build a 32-bit plug-in library:

```
xlc_r -qmkshrobj -o MyPlugin.so MyPlugin.c -bE:MyPlugin.exp
```

On all platforms other than AIX, security plug-in libraries are always assumed to be dynamically loadable shared objects.

## GSS-API security plug-ins do not support message encryption and signing

Message encryption and signing is not available in GSS-API security plug-ins.



---

## Chapter 10. Developing security plug-ins

This chapter will focus on creating user specific security plug-ins along with restrictions and error messages. To develop a security plug-in, you need an initialization function, and the plug-in must return a integer value to indicate success or failure of the execution of the API.

---

### How DB2 loads security plug-ins

So that the DB2 database system has the necessary information to call security plug-in functions, a security plug-in must have a correctly set up initialization function.

Each plug-in library must contain an initialization function with a specific name determined by the plug-in type:

- Server side authentication plug-in: `db2secServerAuthPluginInit()`
- Client side authentication plug-in: `db2secClientAuthPluginInit()`
- Group plug-in: `db2secGroupPluginInit()`

This function is known as the plug-in initialization function. The plug-in initialization function initializes the specified plug-in and provides DB2 with information that it requires to call the plug-in's functions. The plug-in initialization function accepts the following parameters:

- The highest version number of the function pointer structure that the DB2 instance invoking the plugin can support
- A pointer to a structure containing pointers to all the APIs requiring implementation
- A pointer to a function that adds log messages to the `db2diag` log files
- A pointer to an error message string
- The length of the error message

The following is a function signature for the initialization function of a group retrieval plug-in:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
 db2int32 version,
 void *group_fns,
 db2secLogMessage *logMessage_fn,
 char **errmsg,
 db2int32 *errmsglen);
```

**Note:** If the plug-in library is compiled as C++, all functions must be declared with: `extern "C"`. DB2 relies on the underlying operating system dynamic loader to handle the C++ constructors and destructors used inside of a C++ user-written plug-in library.

The initialization function is the only function in the plug-in library that uses a prescribed function name. The other plug-in functions are referenced through function pointers returned from the initialization function. Server plug-ins are loaded when the DB2 server starts. Client plug-ins are loaded when required on the client. Immediately after DB2 loads a plug-in library, it will resolve the location of this initialization function and call it. The specific task of this function is as follows:

- Cast the functions pointer to a pointer to an appropriate functions structure
- Fill in the pointers to the other functions in the library
- Fill in the version number of the function pointer structure being returned

DB2 can potentially call the plug-in initialization function more than once. This situation can occur when an application dynamically loads the DB2 client library, unloads it, and reloads it again, then performs authentication functions from a plug-in both before and after reloading. In this situation, the plug-in library might not be unloaded and then re-loaded; however, this behavior varies depending on the operating system.

Another example of DB2 issuing multiple calls to a plug-in initialization function occurs during the execution of stored procedures or federated system calls, where the database server can itself act as a client. If the client and server plug-ins on the database server are in the same file, DB2 could call the plug-in initialization function twice.

If the plug-in detects that `db2secGroupPluginInit` is called more than once, it should handle this event as if it was directed to terminate and reinitialize the plug-in library. As such, the plug-in initialization function should do the entire cleanup tasks that a call to `db2secPluginTerm` would do before returning the set of function pointers again.

On a DB2 server running on a UNIX or Linux-based operating system, DB2 can potentially load and initialize plug-in libraries more than once in different processes.

---

## Calling sequences for the security plug-in APIs

The sequence with which the DB2 database manager calls the security plug-in APIs varies according to the scenario in which the security plug-in API is called.

These are the main scenarios in which the DB2 database manager will call security plug-in APIs:

- On a client for a database connection (implicit and explicit)
  - CLIENT
  - Server based (SERVER, SERVER\_ENCRYPT, DATA\_ENCRYPT)
  - GSSAPI and Kerberos
- On a client, server, or gateway for local authorization
- On a server for a database connection
- On a server for a grant statement
- On a server to get a list of groups to which an authorization ID belongs

**Note:** The DB2 database servers treat database actions requiring local authorizations, such as `db2start`, `db2stop`, and `db2trc` like client applications.

For each of these operations, the sequence with which the DB2 database manager calls the security plug-in APIs is different. Following are the sequences of APIs called by the DB2 database manager for each of these scenarios.

### CLIENT - implicit

When the user-configured authentication type is CLIENT, the DB2 client application will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext()`;



- `db2secValidatePassword();`
- `db2secFreetoken();`

For an implicit authentication, that is, when you connect without specifying a particular user ID or password, the `db2secValidatePassword` API is called if you are using a user ID/password plug-in. This API permits plug-in developers to prohibit implicit authentication if necessary.

#### **CLIENT - explicit**

On an explicit authentication, that is, when you connect to a database in which both the user ID and password are specified, if the *authentication* database manager configuration parameter is set to CLIENT the DB2 client application will call the following security plug-in APIs multiple times if the implementation requires it:

- `db2secRemapUserid();`
- `db2secValidatePassword();`
- `db2secFreeToken();`

#### **Server based (SERVER, SERVER\_ENCRYPT, DATA\_ENCRYPT) - implicit**

On an implicit authentication, when the client and server have negotiated user ID/password authentication (for instance, when the *srvcon\_auth* parameter at the server is set to SERVER; SERVER\_ENCRYPT, DATA\_ENCRYPT, or DATA\_ENCRYPT\_CMP), the client application will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext();`
- `db2secFreeToken();`

#### **Server based (SERVER, SERVER\_ENCRYPT, DATA\_ENCRYPT) - explicit**

On an explicit authentication, when the client and server have negotiated userid/password authentication (for instance, when the *srvcon\_auth* parameter at the server is set to SERVER; SERVER\_ENCRYPT, DATA\_ENCRYPT, or DATA\_ENCRYPT\_CMP), the client application will call the following security plug-in APIs:

- `db2secRemapUserid();`

#### **GSSAPI and Kerberos - implicit**

On an implicit authentication, when the client and server have negotiated GSS-API or Kerberos authentication (for instance, when the *srvcon\_auth* parameter at the server is set to KERBEROS; KRB\_SERVER\_ENCRYPT, GSSPLUGIN, or GSS\_SERVER\_ENCRYPT), the client application will call the following security plug-in APIs. (The call to `gss_init_sec_context()` will use GSS\_C\_NO\_CREDENTIAL as the input credential.)

- `db2secGetDefaultLoginContext();`
- `db2secProcessServerPrincipalName();`
- `gss_init_sec_context();`
- `gss_release_buffer();`
- `gss_release_name();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`

With multi-flow GSS-API support, `gss_init_sec_context()` can be called multiple times if the implementation requires it.

#### **GSSAPI and Kerberos - explicit**

If the negotiated authentication type is GSS-API or Kerberos, the client application will call the following security plug-in APIs for GSS-API

plug-ins in the following sequence. These APIs are used for both implicit and explicit authentication unless otherwise stated.

- `db2secProcessServerPrincipalName()`;
- `db2secGenerateInitialCred()`; (For explicit authentication only)
- `gss_init_sec_context()`;
- `gss_release_buffer ()`;
- `gss_release_name()`;
- `gss_release_cred()`;
- `db2secFreeInitInfo()`;
- `gss_delete_sec_context()`;
- `db2secFreeToken()`;

The API `gss_init_sec_context()` may be called multiple times if a mutual authentication token is returned from the server and the implementation requires it.

#### **On a client, server, or gateway for local authorization**

For a local authorization, the DB2 command being used will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext()`;
- `db2secGetGroupsForUser()`;
- `db2secFreeToken()`;
- `db2secFreeGroupList()`;

These APIs will be called for both user ID/password and GSS-API authentication mechanisms.

#### **On a server for a database connection**

For a database connection on the database server, the DB2 agent process or thread will call the following security plug-in APIs for the user ID/password authentication mechanism:

- `db2secValidatePassword()`; Only if the *authentication* database configuration parameter is not CLIENT
- `db2secGetAuthIDs()`;
- `db2secGetGroupsForUser()`;
- `db2secFreeToken()`;
- `db2secFreeGroupList()`;

For a CONNECT to a database, the DB2 agent process or thread will call the following security plug-in APIs for the GSS-API authentication mechanism:

- `gss_accept_sec_context()`;
- `gss_release_buffer()`;
- `db2secGetAuthIDs()`;
- `db2secGetGroupsForUser()`;
- `gss_delete_sec_context()`;
- `db2secFreeGroupListMemory()`;

#### **On a server for a GRANT statement**

For a GRANT statement that does not specify the USER or GROUP keyword, (for example, "GRANT CONNECT ON DATABASE TO user1"), the DB2

agent process or thread must be able to determine if user1 is a user, a group, or both. Therefore, the DB2 agent process or thread will call the following security plug-in APIs:

- `db2secDoesGroupExist()`;
- `db2secDoesAuthIDExist()`;

#### **On a server to get a list of groups to which an authid belongs**

From your database server, when you need to get a list of groups to which an authorization ID belongs, the DB2 agent process or thread will call the following security plug-in API with only the authorization ID as input:

- `db2secGetGroupsForUser()`;

There will be no token from other security plug-ins.

---

## **Restrictions for developing security plug-in libraries**

There are certain restrictions that affect how you develop plug-in libraries.

Following are the restrictions for developing plug-in libraries.

### **C-linkage**

Plug-in libraries must be linked with C-linkage. Header files providing the prototypes, data structures needed to implement the plug-ins, and error code definitions are provided for C/C++ only. Functions that DB2 will resolve at load time must be declared with `extern "C"` if the plug-in library is compiled as C++.

### **.NET common language runtime is not supported**

The .NET common language runtime (CLR) is not supported for compiling and linking source code for plug-in libraries.

### **Signal handlers**

Plug-in libraries must not install signal handlers or change the signal mask, because this will interfere with DB2's signal handlers. Interfering with the DB2 signal handlers could seriously interfere with DB2's ability to report and recover from errors, including traps in the plug-in code itself. Plug-in libraries should also never throw C++ exceptions, as this can also interfere with DB2's error handling.

### **Thread-safe**

Plug-in libraries must be thread-safe and re-entrant. The plug-in initialization function is the only API that is not required to be re-entrant. The plug-in initialization function could potentially be called multiple times from different processes; in which case, the plug-in will cleanup all used resources and reinitialize itself.

### **Exit handlers and overriding standard C library and operating system calls**

Plug-in libraries should not override standard C library or operating system calls. Plug-in libraries should also not install exit handlers or `pthread_atfork` handlers. The use of exit handlers is not recommended because they could be unloaded before the program exits.

### **Library dependencies**

On Linux or UNIX, the processes that load the plug-in libraries can be `setuid` or `setgid`, which means that they will not be able to rely on the `$LD_LIBRARY_PATH`, `$SHLIB_PATH`, or `$LIBPATH` environment variables to find dependent libraries. Therefore, plug-in libraries should not depend on additional libraries, unless any dependant libraries are accessible through other methods, such as the following:

- By being in `/lib` or `/usr/lib`
- By having the directories they reside in being specified OS-wide (such as in the `ld.so.conf` file on Linux)
- By being specified in the `RPATH` in the plug-in library itself

This restriction is not applicable to Windows operating systems.

### Symbol collisions

When possible, plug-in libraries should be compiled and linked with any available options that reduce the likelihood of symbol collisions, such as those that reduce unbound external symbolic references. For example, use of the `"-Bsymbolic"` linker option on HP, Solaris, and Linux can help prevent problems related to symbol collisions. However, for plug-ins written on AIX, do not use the `"-brtl"` linker option explicitly or implicitly.

### 32-bit and 64-bit applications

32-bit applications must use 32-bit plug-ins. 64-bit applications must use 64-bit plug-ins. Refer to the topic about 32-bit and 64-bit considerations for more details.

### Text strings

Input text strings are not guaranteed to be null-terminated, and output strings are not required to be null-terminated. Instead, integer lengths are given for all input strings, and pointers to integers are given for lengths to be returned.

### Passing authorization ID parameters

An authorization ID (`authid`) parameter that DB2 passes into a plug-in (an input `authid` parameter) will contain an upper-case `authid`, with padded blanks removed. An `authid` parameter that a plug-in returns to DB2 (an output `authid` parameter) does not require any special treatment, but DB2 will fold the `authid` to upper-case and pad it with blanks according to the internal DB2 standard.

### Size limits for parameters

The plug-in APIs use the following as length limits for parameters:

```
#define DB2SEC_MAX_AUTHID_LENGTH 255
#define DB2SEC_MAX_USERID_LENGTH 255
#define DB2SEC_MAX_USERSPACE_LENGTH 255
#define DB2SEC_MAX_PASSWORD_LENGTH 255
#define DB2SEC_MAX_DBNAME_LENGTH 128
```

A particular plug-in implementation may require or enforce smaller maximum lengths for the authorization IDs, user IDs, and passwords. In particular, the operating system authentication plug-ins supplied with DB2 database systems are restricted to the maximum user, group and namespace length limits enforced by the operating system for cases where the operating system limits are lower than those stated above.

### Security plug-in library extensions in AIX

On AIX systems, security plug-in libraries can have a file name extension of `.a` or `.so`. The mechanism used to load the plug-in library depends on which extension is used:

- Plug-in libraries with a file name extension of `.a` are assumed to be archives containing shared object members. These members must be named `shr.o` (32-bit) or `shr64.o` (64-bit). A single archive can contain both the 32-bit and 64-bit members, allowing it to be deployed on both types of platforms.

For example, to build a 32-bit archive style plug-in library:

```
xlc_r -qmkshrobj -o shr.o MyPlugin.c -bE:MyPlugin.exp
ar rv MyPlugin.a shr.o
```

- Plug-in libraries with a file name extension of `.so` are assumed to be dynamically loadable shared objects. Such an object is either 32-bit or 64-bit, depending on the compiler and linker options used when it was built. For example, to build a 32-bit plug-in library:

```
xlc_r -qmkshrobj -o MyPlugin.so MyPlugin.c -bE:MyPlugin.exp
```

On all platforms other than AIX, security plug-in libraries are always assumed to be dynamically loadable shared objects.

## Return codes for security plug-ins

All security plug-in APIs must return an integer value to indicate the success or failure of the execution of the API. A return code value of 0 indicates that the API ran successfully. All negative return codes, with the exception of -3, -4, and -5, indicate that the API encountered an error.

All negative return codes returned from the security-plug-in APIs are mapped to SQLCODE -1365, SQLCODE -1366, or SQLCODE -30082, with the exception of return codes with the -3, -4, or -5. The values -3, -4, and -5 are used to indicate whether or not an authorization ID represents a valid user or group.

All the security plug-in API return codes are defined in `db2secPlugin.h`, which can be found in the DB2 include directory: `SQLLIB/include`.

Details regarding all of the security plug-in return codes are presented in the following table:

Table 32. Security plug-in return codes

| Return code | Define value                     | Meaning                                                                                                                                                                | Applicable APIs                                                                                    |
|-------------|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| 0           | DB2SEC_PLUGIN_OK                 | The plug-in API executed successfully.                                                                                                                                 | All                                                                                                |
| -1          | DB2SEC_PLUGIN_UNKNOWNERROR       | The plug-in API encountered an unexpected error.                                                                                                                       | All                                                                                                |
| -2          | DB2SEC_PLUGIN_BADUSER            | The user ID passed in as input is not defined.                                                                                                                         | db2secGenerateInitialCred<br>db2secValidatePassword<br>db2secRemapUserid<br>db2secGetGroupsForUser |
| -3          | DB2SEC_PLUGIN_INVALIDUSERORGROUP | No such user or group.                                                                                                                                                 | db2secDoesAuthIDExist<br>db2secDoesGroupExist                                                      |
| -4          | DB2SEC_PLUGIN_USERSTATUSNOTKNOWN | Unknown user status. This is not treated as an error by DB2; it is used by a GRANT statement to determine if an authid represents a user or an operating system group. | db2secDoesAuthIDExist                                                                              |

Table 32. Security plug-in return codes (continued)

| Return code | Define value                              | Meaning                                                                                                                                                                 | Applicable APIs                                                                   |
|-------------|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| -5          | DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN         | Unknown group status. This is not treated as an error by DB2; it is used by a GRANT statement to determine if an authid represents a user or an operating system group. | db2secDoesGroupExist                                                              |
| -6          | DB2SEC_PLUGIN_UID_EXPIRED                 | User ID expired.                                                                                                                                                        | db2secValidatePassword<br>db2GetGroupsForUser<br>db2secGenerateInitialCred        |
| -7          | DB2SEC_PLUGIN_PWD_EXPIRED                 | Password expired.                                                                                                                                                       | db2secValidatePassword<br>db2GetGroupsForUser<br>db2secGenerateInitialCred        |
| -8          | DB2SEC_PLUGIN_USER_REVOKED                | User revoked.                                                                                                                                                           | db2secValidatePassword<br>db2GetGroupsForUser                                     |
| -9          | DB2SEC_PLUGIN_USER_SUSPENDED              | User suspended.                                                                                                                                                         | db2secValidatePassword<br>db2GetGroupsForUser                                     |
| -10         | DB2SEC_PLUGIN_BADPWD                      | Bad password.                                                                                                                                                           | db2secValidatePassword<br>db2secRemapUserid<br>db2secGenerateInitialCred          |
| -11         | DB2SEC_PLUGIN_BAD_NEWPASSWORD             | Bad new password.                                                                                                                                                       | db2secValidatePassword<br>db2secRemapUserid                                       |
| -12         | DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED | Change password not supported.                                                                                                                                          | db2secValidatePassword<br>db2secRemapUserid<br>db2secGenerateInitialCred          |
| -13         | DB2SEC_PLUGIN_NOMEM                       | Plug-in attempt to allocate memory failed due to insufficient memory.                                                                                                   | All                                                                               |
| -14         | DB2SEC_PLUGIN_DISKERROR                   | Plug-in encountered a disk error.                                                                                                                                       | All                                                                               |
| -15         | DB2SEC_PLUGIN_NOPERM                      | Plug-in attempt to access a file failed because of wrong permissions on the file.                                                                                       | All                                                                               |
| -16         | DB2SEC_PLUGIN_NETWORKERROR                | Plug-in encountered a network error.                                                                                                                                    | All                                                                               |
| -17         | DB2SEC_PLUGIN_CANTLOADLIBRARY             | Plug-in is unable to load a required library.                                                                                                                           | db2secGroupPluginInit<br>db2secClientAuthPluginInit<br>db2secServerAuthPluginInit |
| -18         | DB2SEC_PLUGIN_CANT_OPEN_FILE              | Plug-in is unable to open and read a file for a reason other than missing file or inadequate file permissions.                                                          | All                                                                               |

Table 32. Security plug-in return codes (continued)

| Return code | Define value                          | Meaning                                                                                                                                                                          | Applicable APIs                                                                   |
|-------------|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| -19         | DB2SEC_PLUGIN_FILENOTFOUND            | Plug-in is unable to open and read a file, because the file is missing from the file system.                                                                                     | All                                                                               |
| -20         | DB2SEC_PLUGIN_CONNECTION_DISALLOWED   | The plug-in is refusing the connection because of the restriction on which database is allowed to connect, or the TCP/IP address cannot connect to a specific database.          | All server-side plug-in APIs.                                                     |
| -21         | DB2SEC_PLUGIN_NO_CRED                 | GSS API plug-in only: initial client credential is missing.                                                                                                                      | db2secGetDefaultLoginContext<br>db2secServerAuthPluginInit                        |
| -22         | DB2SEC_PLUGIN_CRED_EXPIRED            | GSS API plug-in only: client credential has expired.                                                                                                                             | db2secGetDefaultLoginContext<br>db2secServerAuthPluginInit                        |
| -23         | DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME      | GSS API plug-in only: the principal name is invalid.                                                                                                                             | db2secProcessServerPrincipalName                                                  |
| -24         | DB2SEC_PLUGIN_NO_CON_DETAILS          | This return code is returned by the db2secGetConDetails callback (for example, from DB2 to the plug-in) to indicate that DB2 is unable to determine the client's TCP/IP address. | db2secGetConDetails                                                               |
| -25         | DB2SEC_PLUGIN_BAD_INPUT_PARAMETERS    | Some parameters are not valid or are missing when plug-in API is called.                                                                                                         | All                                                                               |
| -26         | DB2SEC_PLUGIN_INCOMPATIBLE_VER        | The version of the APIs reported by the plug-in is not compatible with DB2.                                                                                                      | db2secGroupPluginInit<br>db2secClientAuthPluginInit<br>db2secServerAuthPluginInit |
| -27         | DB2SEC_PLUGIN_PROCESS_LIMIT           | Insufficient resources are available for the plug-in to create a new process.                                                                                                    | All                                                                               |
| -28         | DB2SEC_PLUGIN_NO_LICENSES             | The plug-in encountered a user license problem. A possibility exists that the underlying mechanism license has reached the limit.                                                | All                                                                               |
| -29         | DB2SEC_PLUGIN_ROOT_NEEDED             | The plug-in is trying to run an application that requires root privileges.                                                                                                       | All                                                                               |
| -30         | DB2SEC_PLUGIN_UNEXPECTED_SYSTEM_ERROR | The plug-in encountered an unexpected system error. A possibility exists that the current system configuration is not supported.                                                 | All                                                                               |



---

## Error message handling for security plug-ins

When an error occurs in a security plug-in API, the API can return an ASCII text string in the `errmsg` field to provide a more specific description of the problem than the return code.

For example, the `errmsg` string can contain "File /home/db2inst1/mypasswd.txt does not exist." DB2 will write this entire string into the DB2 administration notification log, and will also include a truncated version as a token in some SQL messages. Because tokens in SQL messages can only be of limited length, these messages should be kept short, and important variable portions of these messages should appear at the front of the string. To aid in debugging, consider adding the name of the security plug-in to the error message.

For non-urgent errors, such as password expired errors, the `errmsg` string will only be dumped when the `DIAGLEVEL` database manager configuration parameter is set at 4.

The memory for these error messages must be allocated by the security plug-in. Therefore, the plug-ins must also provide an API to free this memory: `db2secFreeErrorMsg`.

The `errmsg` field will only be checked by DB2 if an API returns a non-zero value. Therefore, the plug-in should not allocate memory for this returned error message if there is no error.

At initialization time a message logging function pointer, `logMessage_fn`, is passed to the group, client, and server plug-ins. The plug-ins can use the function to log any debugging information to the `db2diag` log files. For example:

```
// Log an message indicate init successful
(*(logMessage_fn))(DB2SEC_LOG_CRITICAL,
 "db2secGroupPluginInit successful",
 strlen("db2secGroupPluginInit successful"));
```

For more details about each parameter for the `db2secLogMessage` function, refer to the initialization API for each of the plug-in types.

---

## Chapter 11. Security plug-in APIs

---

### Security plug-in APIs

To enable you to customize the DB2 database system authentication and group membership lookup behavior, the DB2 database system provides APIs that you can use to modify existing plug-in modules or build new security plug-in modules.

When you develop a security plug-in module, you need to implement the standard authentication or group membership lookup functions that the DB2 database manager will invoke. For the three available types of plug-in modules, the functionality you need to implement is as follows:

#### **Group retrieval**

Retrieves group membership information for a given user and determines if a given string represents a valid group name.

#### **User ID/password authentication**

Authentication that identifies the default security context (client only), validates and optionally changes a password, determines if a given string represents a valid user (server only), modifies the user ID or password provided on the client before it is sent to the server (client only), returns the DB2 authorization ID associated with a given user.

#### **GSS-API authentication**

Authentication that implements the required GSS-API functions, identifies the default security context (client side only), generates initial credentials based on user ID and password, and optionally changes password (client side only), creates and accepts security tickets, and returns the DB2 authorization ID associated with a given GSS-API security context.

The following are the definitions for terminology used in the descriptions of the plug-in APIs.

#### **Plug-in**

A dynamically loadable library that DB2 will load to access user-written authentication or group membership lookup functions.

#### **Implicit authentication**

A connection to a database without specifying a user ID or a password.

#### **Explicit authentication**

A connection to a database in which both the user ID and password are specified.

#### **Authid**

An internal ID representing an individual or group to which authorities and privileges within the database are granted. Internally, a DB2 authid is folded to upper-case and is a minimum of 8 characters (blank padded to 8 characters). Currently, DB2 requires authids, user IDs, passwords, group names, namespaces, and domain names that can be represented in 7-bit ASCII.

#### **Local authorization**

Authorization that is local to the server or client that implements it, that checks if a user is authorized to perform an action (other than connecting

to the database), such as starting and stopping the database manager, turning DB2 trace on and off, or updating the database manager configuration.

### Namespace

A collection or grouping of users within which individual user identifiers must be unique. Common examples include Windows domains and Kerberos Realms. For example, within the Windows domain "usa.company.com" all user names must be unique. For example, "user1@usa.company.com". The same user ID in another domain, as in the case of "user1@canada.company.com", however refers to a different person. A fully qualified user identifier includes a user ID and namespace pair; for example, "user@domain.name" or "domain\user".

**Input** Indicates that DB2 will fill in the value for the security plug-in API parameter.

### Output

Indicates that the security plug-in API will fill in the value for the API parameter.

---

## Group plug-in APIs

### APIs for group retrieval plug-ins

For the group retrieval plug-in module, you need to implement the following APIs:

- db2secGroupPluginInit

**Note:** The db2secGroupPluginInit API takes as input a pointer, \*logMessage\_fn, to an API with the following prototype:

```
SQL_API_RC (SQL_API_FN db2secLogMessage)
(
 db2int32 level,
 void *data,
 db2int32 length
);
```

The db2secLogMessage API allows the plug-in to log messages to the db2diag log files for debugging or informational purposes. This API is provided by the DB2 database system, so you need not implement it.

- db2secPluginTerm
- db2secGetGroupsForUser
- db2secDoesGroupExist
- db2secFreeGroupListMemory
- db2secFreeErrorMsg
- The only API that must be resolvable externally is db2secGroupPluginInit. This API will take a void \* parameter, which should be cast to the type:

```
typedef struct db2secGroupFunctions_1
{
 db2int32 version;
 db2int32 pluginType;
 SQL_API_RC (SQL_API_FN * db2secGetGroupsForUser)
 (
 const char *authid,
 db2int32 authidlen,
 const char *userid,
 db2int32 useridlen,
 const char *usernamespace,
```

```

db2int32 usernamespace1en,
db2int32 usernamespacetype,
const char *dbname,
db2int32 dbnamelen,
const void *token,
db2int32 tokentype,
db2int32 location,
const char *authpluginname,
db2int32 authpluginnamelen,
void **group1ist,
db2int32 *numgroups,
char **errmsgs,
db2int32 *errmsgslen
);

SQL_API_RC (SQL_API_FN * db2secDoesGroupExist)
(
const char *groupname,
db2int32 groupnamelen,
char **errmsgs,
db2int32 *errmsgslen
);

SQL_API_RC (SQL_API_FN * db2secFreeGroupListMemory)
(
void *ptr,
char **errmsgs,
db2int32 *errmsgslen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
char *msgtobefree
);

SQL_API_RC (SQL_API_FN * db2secPluginTerm)
(
char **errmsgs,
db2int32 *errmsgslen
);

} db2secGroupFunctions_1;

```

The db2secGroupPluginInit API assigns the addresses for the rest of the externally available functions.

**Note:** The `_1` indicates that this is the structure corresponding to version 1 of the API. Subsequent interface versions will have the extension `_2`, `_3`, and so on.

## db2secGroupPluginInit API - Initialize group plug-in

Initialization API, for the group-retrieval plug-in, that the DB2 database manager calls immediately after loading the plug-in.

### API and data structure syntax

```

SQL_API_RC SQL_API_FN db2secGroupPluginInit
(
db2int32 version,
void *group_fns,
db2secLogMessage *logMessage_fn,
char **errmsgs,
db2int32 *errmsgslen);

```

## db2secGroupPluginInit API parameters

### version

Input. The highest version of the API supported by the instance loading that plugin. The value DB2SEC\_API\_VERSION (in db2secPlugin.h) contains the latest version number of the API that the DB2 database manager currently supports.

### group\_fns

Output. A pointer to the db2secGroupFunctions\_<version\_number> (also known as group\_functions\_<version\_number>) structure. The db2secGroupFunctions\_<version\_number> structure contains pointers to the APIs implemented for the group-retrieval plug-in. In future, there might be different versions of the APIs (for example, db2secGroupFunctions\_<version\_number>), so the group\_fns parameter is cast as a pointer to the db2secGroupFunctions\_<version\_number> structure corresponding to the version the plug-in has implemented. The first parameter of the group\_functions\_<version\_number> structure tells DB2 the version of the APIs that the plug-in has implemented. Note: The casting is done only if the DB2 version is higher or equal to the version of the APIs that the plug-in has implemented. The version number represents the version of the APIs implemented by the plugin, and the pluginType should be set to DB2SEC\_PLUGIN\_TYPE\_GROUP.

### logMessage\_fn

Input. A pointer to the db2secLogMessage API, which is implemented by the DB2 database system. The db2secGroupPluginInit API can call the db2secLogMessage API to log messages to the db2diag log files for debugging or informational purposes. The first parameter (level) of db2secLogMessage API specifies the type of diagnostic errors that will be recorded in the db2diag log files and the last two parameters respectively are the message string and its length. The valid values for the first parameter of db2secLogMessage API (defined in db2secPlugin.h) are:

- DB2SEC\_LOG\_NONE: (0) No logging
- DB2SEC\_LOG\_CRITICAL: (1) Severe Error encountered
- DB2SEC\_LOG\_ERROR: (2) Error encountered
- DB2SEC\_LOG\_WARNING: (3) Warning
- DB2SEC\_LOG\_INFO: (4) Informational

The message text will show up in the diag.log only if the value of the 'level' parameter of the db2secLogMessage API is less than or equal to the diaglevel database manager configuration parameter. So for example, if you use the DB2SEC\_LOG\_INFO value, the message text will only show up in the db2diag log files if the diaglevel database manager configuration parameter is set to 4.

### errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGroupPluginInit API execution is not successful.

### errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## db2secPluginTerm - Clean up group plug-in resources

Frees resources used by the group-retrieval plug-in.

This API is called by the DB2 database manager just before it unloads the group-retrieval plug-in. It should be implemented in a manner that it does a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources in order to free them. This API is not called on any Windows platform.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secPluginTerm)
 (char **errmsg,
 db2int32 *errmsglen);
```

### db2secPluginTerm API parameters

#### errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secPluginTerm API execution is not successful.

#### errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## db2secGetGroupsForUser API - Get list of groups for user

Returns the list of groups to which a user belongs.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secGetGroupsForUser)
 (const char *authid,
 db2int32 authidlen,
 const char *userid,
 db2int32 useridlen,
 const char *usernamespace,
 db2int32 usernamespace,
 db2int32 usernamespace,
 const char *dbname,
 db2int32 dbnamelen,
 void *token,
 db2int32 tokentype,
 db2int32 location,
 const char *authpluginname,
 db2int32 authpluginname,
 void **group,
 db2int32 *numgroups,
 char **errmsg,
 db2int32 *errmsglen);
```

### db2secGetGroupsForUser API parameters

**authid** Input. This parameter value is an SQL authid, which means that DB2 converts it to an uppercase character string with no trailing blanks. DB2 will always provide a non-null value for the authid parameter. The API must be able to return a list of groups to which the authid belongs without depending on the other input parameters. It is permissible to return a shortened or empty list if this cannot be determined.

If a user does not exist, the API must return the return code DB2SEC\_PLUGIN\_BADUSER. DB2 does not treat the case of a user not existing as an error, since it is permissible for an authid to not have any groups associated with it. For example, the db2secGetAuthids API can

return an authid that does not exist on the operating system. The authid is not associated with any groups, however, it can still be assigned privileges directly.

If the API cannot return a complete list of groups using only the authid, then there will be some restrictions on certain SQL functions related to group support. For a list of possible problem scenarios, refer to the Usage notes section in this topic.

**authidlen**

Input. Length in bytes of the authid parameter value. The DB2 database manager always provides a non-zero value for the authidlen parameter.

**userid** Input. This is the user ID corresponding to the authid. When this API is called on the server in a non-connect scenario, this parameter will not be filled by DB2.

**useridlen**

Input. Length in bytes of the userid parameter value.

**usernamespace**

Input. The namespace from which the user ID was obtained. When the user ID is not available, this parameter will not be filled by the DB2 database manager.

**usernamespacelen**

Input. Length in bytes of the usernamespace parameter value.

**usernamespacestype**

Input. The type of namespace. Valid values for the usernamespacestype parameter (defined in db2secPlugin.h) are:

- DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE Corresponds to a username style like domain\myname
- DB2SEC\_NAMESPACE\_USER\_PRINCIPAL Corresponds to a username style like myname@domain.ibm.com

Currently, the DB2 database system only supports the value DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE. When the user ID is not available, the usernamespacestype parameter is set to the value DB2SEC\_NAMESPACE\_USER\_NAMESPACE\_UNDEFINED (defined in db2secPlugin.h).

**dbname**

Input. Name of the database being connected to. This parameter can be NULL in a non-connect scenario.

**dbnamelen**

Input. Length in bytes of the dbname parameter value. This parameter is set to 0 if dbname parameter is NULL in a non-connect scenario.

**token** Input. A pointer to data provided by the authentication plug-in. It is not used by DB2. It provides the plug-in writer with the ability to coordinate user and group information. This parameter might not be provided in all cases (for example, in a non-connect scenario), in which case it will be NULL. If the authentication plug-in used is GSS-API based, the token will be set to the GSS-API context handle (gss\_ctx\_id\_t).

**tokentype**

Input. Indicates the type of data provided by the authentication plug-in. If the authentication plug-in used is GSS-API based, the token will be set to the GSS-API context handle (gss\_ctx\_id\_t). If the authentication plug-in



used is user ID/password based, it will be a generic type. Valid values for the tokentype parameter (defined in db2secPlugin.h) are:

- DB2SEC\_GENERIC: Indicates that the token is from a user ID/password based plug-in.
- DB2SEC\_GSSAPI\_CTX\_HANDLE: Indicates that the token is from a GSS-API (including Kerberos) based plug-in.

#### **location**

Input. Indicates whether DB2 is calling this API on the client side or server side. Valid values for the location parameter (defined in db2secPlugin.h) are:

- DB2SEC\_SERVER\_SIDE: The API is to be called on the database server.
- DB2SEC\_CLIENT\_SIDE: The API is to be called on a client.

#### **authpluginname**

Input. Name of the authentication plug-in that provided the data in the token. The db2secGetGroupsForUser API might use this information in determining the correct group memberships. This parameter might not be filled by DB2 if the authid is not authenticated (for example, if the authid does not match the current connected user).

#### **authpluginnamelen**

Input. Length in bytes of the authpluginname parameter value.

#### **grouplist**

Output. List of groups to which the user belongs. The list of groups must be returned as a pointer to a section of memory allocated by the plug-in containing concatenated varchars (a varchar is a character array in which the first byte indicates the number of bytes following it). The length is an unsigned char (1 byte) and that limits the maximum length of a groupname to 255 characters. For example, "\006GROUP1\007MYGROUP\008MYGROUP3". Each group name should be a valid DB2 authid. The memory for this array must be allocated by the plug-in. The plug-in must therefore provide an API, such as the db2secFreeGroupListMemory API that DB2 will call to free the memory.

#### **numgroups**

Output. The number of groups contained in the grouplist parameter.

#### **errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGetGroupsForUser API execution is not successful.

#### **errormsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## **Usage notes**

The following is a list of scenarios when problems can occur if an incomplete group list is returned by this API to DB2:

- Alternate authorization is provided in CREATE SCHEMA statement. Group lookup will be performed against the AUTHORIZATION NAME parameter if there are nested CREATE statements in the CREATE SCHEMA statement.
- Processing a jar file in an MPP environment. In an MPP environment, the jar processing request is sent from the coordinator node with the session authid.

The catalog node received the requests and process the jar files based on the privilege of the session authid (the user executing the jar processing requests).

- Install jar file. The session authid needs to have one of the following rights: DBADM, or CREATEIN (implicit or explicit on the jar schema). The operation will fail if the above rights are granted to group containing the session authid, but not explicitly to the session authid.
  - Remove jar file. The session authid needs to have one of the following rights: DBADM, or DROPIN (implicit or explicit on the jar schema), or is the definer of the jar file. The operation will fail if the above rights are granted to group containing the session authid, but not explicitly to the session authid, and if the session authid is not the definer of the jar file.
  - Replace jar file. This is same as removing the jar file, followed by installing the jar file. Both of the above apply.
- When SET SESSION\_USER statement is issued. Subsequent DB2 operations are run under the context of the authid specified by this statement. These operations will fail if the privileges required are owned by one of the SESSION\_USER's group is not explicitly granted to the SESSION\_USER authid.

## db2secDoesGroupExist API - Check if group exists

Determines if an authid represents a group.

If the groupname exists, the API must be able to return the value DB2SEC\_PLUGIN\_OK, to indicate success. It must also be able to return the value DB2SEC\_PLUGIN\_INVALIDUSERORGROUP if the group name is not valid. It is permissible for the API to return the value DB2SEC\_PLUGIN\_GROUPSTATUSNOTKNOWN if it is impossible to determine if the input is a valid group. If an invalid group (DB2SEC\_PLUGIN\_INVALIDUSERORGROUP) or group not known (DB2SEC\_PLUGIN\_GROUPSTATUSNOTKNOWN) value is returned, DB2 might not be able to determine whether the authid is a group or user when issuing the GRANT statement without the keywords USER and GROUP, which would result in the error SQLCODE -569, SQLSTATE 56092 being returned to the user.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secDoesGroupExist)
 (const char *groupname,
 db2int32 groupnamelen,
 char **errmsgmsg,
 db2int32 *errmsgglen);
```

### db2secDoesGroupExist API parameters

#### groupname

Input. An authid, upper-cased, with no trailing blanks.

#### groupnamelen

Input. Length in bytes of the groupname parameter value.

#### errmsgmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secDoesGroupExist API execution is not successful.

#### errmsgglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsgmsg parameter.

## db2secFreeGroupListMemory API - Free group list memory

Frees the memory used to hold the list of groups from a previous call to db2secGetGroupsForUser API.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secFreeGroupListMemory)
 (void *ptr,
 char **errmsg,
 db2int32 *errmsglen);
```

### db2secFreeGroupListMemory API parameters

**ptr** Input. Pointer to the memory to be freed.

#### errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secFreeGroupListMemory API execution is not successful.

#### errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in the errmsg parameter.

## db2secFreeErrorMsg API - Free error message memory

Frees the memory used to hold an error message from a previous API call. This is the only API that does not return an error message. If this API returns an error, DB2 will log it and continue.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secFreeErrorMsg)
 (char *errmsg);
```

### db2secFreeErrorMsg API parameters

#### msgtofree

Input. A pointer to the error message allocated from a previous API call.

---

## User authentication plug-in APIs

### APIs for user ID/password authentication plug-ins

For the user ID/password plug-in module, you need to implement the following client-side APIs:

- db2secClientAuthPluginInit

**Note:** The db2secClientAuthPluginInit API takes as input a pointer, \*logMessage\_fn, to an API with the following prototype:

```
SQL_API_RC (SQL_API_FN db2secLogMessage)
(
 db2int32 level,
 void *data,
 db2int32 length
);
```

The db2secLogMessage API allows the plug-in to log messages to the db2diag log files for debugging or informational purposes. This API is provided by the DB2 database system, so you need not implement it.

- db2secClientAuthPluginTerm
- db2secGenerateInitialCred (Only used for gssapi)
- db2secRemapUserid (Optional)
- db2secGetDefaultLoginContext
- db2secValidatePassword
- db2secProcessServerPrincipalName (This is only for GSS-API)
- db2secFreeToken (Functions to free memory held by the DLL)
- db2secFreeErrorMsg
- db2secFreeInitInfo
- The only API that must be resolvable externally is db2secClientAuthPluginInit. This API will take a void \* parameter, which should be cast to either:

```
typedef struct db2secUseridPasswordClientAuthFunctions_1
{
 db2int32 version;
 db2int32 plugintype;

 SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)
 (
 char authid[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32 *authidlen,
 char userid[DB2SEC_MAX_USERID_LENGTH],
 db2int32 *useridlen,
 db2int32 useridtype,
 char usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
 db2int32 *userspaceilen,
 db2int32 *userspacetype,
 const char *dbname,
 db2int32 dbnamelen,
 void **token,
 char **errorMsg,
 db2int32 *errormsglen
);
 /* Optional */
 SQL_API_RC (SQL_API_FN * db2secRemapUserid)
 (
 char userid[DB2SEC_MAX_USERID_LENGTH],
 db2int32 *useridlen,
 char usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
 db2int32 *userspaceilen,
 db2int32 *userspacetype,
 char password[DB2SEC_MAX_PASSWORD_LENGTH],
 db2int32 *passwordlen,
 char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
 db2int32 *newpasswordlen,
 const char *dbname,
 db2int32 dbnamelen,
 char **errorMsg,
 db2int32 *errormsglen
);

 SQL_API_RC (SQL_API_FN * db2secValidatePassword)
 (
 const char *userid,
 db2int32 useridlen,
 const char *userspace,
 db2int32 userspaceilen,
 db2int32 userspacetype,
 const char *password,
 db2int32 passwordlen,
 const char *newpassword,
 db2int32 newpasswordlen,
 const char *dbname,
);
};
```

```

db2int32 dbnameLen,
db2UInt32 connection_details,
void **token,
char **errorMsg,
db2int32 *errormsgLen
);

SQL_API_RC (SQL_API_FN * db2secFreeToken)
(
void **token,
char **errorMsg,
db2int32 *errormsgLen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
char *errorMsg
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char **errorMsg,
db2int32 *errormsgLen
);
}

or

typedef struct db2secGssapiClientAuthFunctions_1
{
db2int32 version;
db2int32 pluginType;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)
(
char authid[DB2SEC_MAX_AUTHID_LENGTH],
db2int32 *authidLen,
char userid[DB2SEC_MAX_USERID_LENGTH],
db2int32 *useridLen,
db2int32 useridType,
char usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
db2int32 *userspaceLen,
db2int32 *userspacetype,
const char *dbName,
db2int32 dbnameLen,
void **token,
char **errorMsg,
db2int32 *errormsgLen
);

SQL_API_RC (SQL_API_FN * db2secProcessServerPrincipalName)
(
const void *data,
gss_name_t *gssName,
char **errorMsg,
db2int32 *errormsgLen
);

SQL_API_RC (SQL_API_FN * db2secGenerateInitialCred)
(
const char *userid,
db2int32 *useridLen,
const char *userspace,
db2int32 *userspaceLen,
db2int32 userspacetype,
const char *password,
db2int32 *passwordLen,

```

```

const char *newpassword,
db2int32 newpasswordlen,
const char *dbname,
db2int32 dbnamelen,
gss_cred_id_t *pGSSCredHandle,
void **initInfo,
char **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeToken)
(
void *token,
char **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
char *errmsg
);

SQL_API_RC (SQL_API_FN * db2secFreeInitInfo)
(
void *initInfo,
char **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char **errmsg,
db2int32 *errmsglen
);

/* GSS-API specific functions -- refer to db2secPlugin.h
for parameter list*/

OM_uint32 (SQL_API_FN * gss_init_sec_context)(<parameter list>);
OM_uint32 (SQL_API_FN * gss_delete_sec_context)(<parameter list>);
OM_uint32 (SQL_API_FN * gss_display_status)(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_buffer)(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_cred)(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_name)(<parameter list>);
}

```

You should use the `db2secUseridPasswordClientAuthFunctions_1` structure if you are writing an user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use the `db2secGssapiClientAuthFunctions_1` structure.

For the user ID/password plug-in library, you will need to implement the following server-side APIs:

- `db2secServerAuthPluginInit`

The `db2secServerAuthPluginInit` API takes as input a pointer, `*logMessage_fn`, to the `db2secLogMessage` API, and a pointer, `*getConDetails_fn`, to the `db2secGetConDetails` API with the following prototypes:

```

SQL_API_RC (SQL_API_FN db2secLogMessage)
(
db2int32 level,
void *data,
db2int32 length
);

```

```

SQL_API_RC (SQL_API_FN db2secGetConDetails)
(
 db2int32 conDetailsVersion,
 const void *pConDetails
);

```

The db2secLogMessage API allows the plug-in to log messages to the db2diag log files for debugging or informational purposes. The db2secGetConDetails API allows the plug-in to obtain details about the client that is trying to attempt to have a database connection. Both the db2secLogMessage API and db2secGetConDetails API are provided by the DB2 database system, so you do not need to implement them. The db2secGetConDetails API in turn, takes as its second parameter, pConDetails, a pointer to one of the following structures:

db2sec\_con\_details\_1:

```

typedef struct db2sec_con_details_1
{
 db2int32 clientProtocol;
 db2UInt32 clientIPAddress;
 db2UInt32 connect_info_bitmap;
 db2int32 dbnameLen;
 char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
} db2sec_con_details_1;

```

db2sec\_con\_details\_2:

```

typedef struct db2sec_con_details_2
{
 db2int32 clientProtocol; /* See SQL_PROTOCOL_ in sqlenv.h */
 db2UInt32 clientIPAddress; /* Set if protocol is TCP/IP4 */
 db2UInt32 connect_info_bitmap;
 db2int32 dbnameLen;
 char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
 db2UInt32 clientIP6Address[4]; /* Set if protocol is TCP/IP6 */
} db2sec_con_details_2;

```

db2sec\_con\_details\_3:

```

typedef struct db2sec_con_details_3
{
 db2int32 clientProtocol; /* See SQL_PROTOCOL_ in sqlenv.h */
 db2UInt32 clientIPAddress; /* Set if protocol is TCP/IP4 */
 db2UInt32 connect_info_bitmap;
 db2int32 dbnameLen;
 char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
 db2UInt32 clientIP6Address[4]; /* Set if protocol is TCP/IP6 */
 db2UInt32 clientPlatform; /* SQLM_PLATFORM_ from sqlmon.h */
 db2UInt32 _reserved[16];
} db2sec_con_details_3;

```

The possible values for conDetailsVersion are DB2SEC\_CON\_DETAILS\_VERSION\_1, DB2SEC\_CON\_DETAILS\_VERSION\_2, and DB2SEC\_CON\_DETAILS\_VERSION\_3 representing the version of the API.

**Note:** While using db2sec\_con\_details\_1, db2sec\_con\_details\_2, or db2sec\_con\_details\_3, consider the following:

- Existing plugins that are using the db2sec\_con\_details\_1 structure and the DB2SEC\_CON\_DETAILS\_VERSION\_1 value will continue to work as they did with Version 8.2 when calling the db2GetConDetails API. If this API is called on an IPv4 platform, the client IP address is returned in the clientIPAddress field of the structure. If this API is called on an IPv6 platform, a value of 0 is returned in the clientIPAddress field. To retrieve the client IP address on an IPv6 platform, the security plug-in code should be changed to use either the db2sec\_con\_details\_2 structure and the DB2SEC\_CON\_DETAILS\_VERSION\_2



value, or the db2sec\_con\_details\_3 structure and the DB2SEC\_CON\_DETAILS\_VERSION\_3 value .

- New plugins should use the db2sec\_con\_details\_3 structure and the DB2SEC\_CON\_DETAILS\_VERSION\_3 value. If the db2secGetConDetails API is called on an IPv4 platform, the client IP address is returned in the clientIPAddress field of the db2sec\_con\_details\_3 structure and if the API is called on an IPv6 platform the client IP address is returned in the clientIP6Address field of the db2sec\_con\_details\_3 structure. The *clientProtocol* field of the connection details structure will be set to one of SQL\_PROTOCOL\_TCPIP (IPv4, with v1 of the structure), SQL\_PROTOCOL\_TCPIP4 (IPv4, with v2 of the structure) or SQL\_PROTOCOL\_TCPIP6 (IPv6, with v2 or v3 of the structure).
- The structure db2sec\_con\_details\_3 is identical to the structure db2sec\_con\_details\_2 except that it contains an additional field (*clientPlatform*) that identifies the client platform type (as reported by the communication layer) using platform type constants defined in sqlmon.h, such as SQLM\_PLATFORM\_AIX.

- db2secServerAuthPluginTerm
- db2secValidatePassword
- db2secGetAuthIDs
- db2secDoesAuthIDExist
- db2secFreeToken
- db2secFreeErrorMsg
- The only API that must be resolvable externally is db2secServerAuthPluginInit. This API will take a void \* parameter, which should be cast to either:

```
typedef struct db2secUseridPasswordServerAuthFunctions_1
{
 db2int32 version;
 db2int32 pluginType;

 /* parameter lists left blank for readability
 see above for parameters */
 SQL_API_RC (SQL_API_FN * db2secValidatePassword)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secFreeToken)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();
} userid_password_server_auth_functions;
```

or

```
typedef struct db2secGssapiServerAuthFunctions_1
{
 db2int32 version;
 db2int32 pluginType;
 gss_buffer_desc serverPrincipalName;
 gss_cred_id_t ServerCredHandle;
 SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();

 /* GSS-API specific functions
 refer to db2secPlugin.h for parameter list*/
 OM_uint32 (SQL_API_FN * gss_accept_sec_context)(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_display_name)(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_delete_sec_context)(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_display_status)(<parameter list>);
```

```

OM_uint32 (SQL_API_FN * gss_release_buffer)(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_cred)(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_name)(<parameter list>);

} gssapi_server_auth_functions;

```

You should use the `db2secUseridPasswordServerAuthFunctions_1` structure if you are writing an user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use the `db2secGssapiServerAuthFunctions_1` structure.

## db2secClientAuthPluginInit API - Initialize client authentication plug-in

Initialization API, for the client authentication plug-in, that the DB2 database manager calls immediately after loading the plug-in.

### API and data structure syntax

```

SQL_API_RC SQL_API_FN db2secClientAuthPluginInit
(db2int32 version,
 void *client_fns,
 db2secLogMessage *logMessage_fn,
 char **errmsg,
 db2int32 *errmsglen);

```

### db2secClientAuthPluginInit API parameters

#### version

Input. The highest version number of the API that the DB2 database manager currently supports. The `DB2SEC_API_VERSION` value (in `db2secPlugin.h`) contains the latest version number of the API that DB2 currently supports.

#### client\_fns

Output. A pointer to memory provided by the DB2 database manager for a `db2secGssapiClientAuthFunctions_<version_number>` structure (also known as `gssapi_client_auth_functions_<version_number>`), if GSS-API authentication is used, or a `db2secUseridPasswordClientAuthFunctions_<version_number>` structure (also known as `userid_password_client_auth_functions_<version_number>`), if user ID/password authentication is used. The `db2secGssapiClientAuthFunctions_<version_number>` structure and `db2secUseridPasswordClientAuthFunctions_<version_number>` structure respectively contain pointers to the APIs implemented for the GSS-API authentication plug-in and user ID/password authentication plug-in. In future versions of DB2, there might be different versions of the APIs, so the `client_fns` parameter is cast as a pointer to the `gssapi_client_auth_functions_<version_number>` structure corresponding to the version the plug-in has implemented.

The first parameter of the `gssapi_client_auth_functions_<version_number>` structure or the `userid_password_client_auth_functions_<version_number>` structure tells the DB2 database manager the version of the APIs that the plug-in has implemented.

**Note:** The casting is done only if the DB2 version is higher or equal to the version of the APIs that the plug-in has implemented.

Inside the `gssapi_server_auth_functions_<version_number>` or `userid_password_server_auth_functions_<version_number>` structure, the `plugintype` parameter should be set to one of `DB2SEC_PLUGIN_TYPE_USERID_PASSWORD`, `DB2SEC_PLUGIN_TYPE_GSSAPI`, or `DB2SEC_PLUGIN_TYPE_KERBEROS`. Other values can be defined in future versions of the API.

#### **logMessage\_fn**

Input. A pointer to the `db2secLogMessage` API, which is implemented by the DB2 database manager. The `db2secClientAuthPluginInit` API can call the `db2secLogMessage` API to log messages to the `db2diag` log files for debugging or informational purposes. The first parameter (level) of `db2secLogMessage` API specifies the type of diagnostic errors that will be recorded in the `db2diag` log files and the last two parameters respectively are the message string and its length. The valid values for the first parameter of `db2secLogMessage` API (defined in `db2secPlugin.h`) are:

- `DB2SEC_LOG_NONE` (0) No logging
- `DB2SEC_LOG_CRITICAL` (1) Severe Error encountered
- `DB2SEC_LOG_ERROR` (2) Error encountered
- `DB2SEC_LOG_WARNING` (3) Warning
- `DB2SEC_LOG_INFO` (4) Informational

The message text will show up in the `db2diag` log files only if the value of the 'level' parameter of the `db2secLogMessage` API is less than or equal to the `diaglevel` database manager configuration parameter. For example, if you use the `DB2SEC_LOG_INFO` value, the message text will only appear in the `db2diag` log files if the `diaglevel` database manager configuration parameter is set to 4.

#### **errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the `db2secClientAuthPluginInit` API execution is not successful.

#### **errormsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in `errmsg` parameter.

## **db2secClientAuthPluginTerm API - Clean up client authentication plug-in resources**

Frees resources used by the client authentication plug-in.

This API is called by the DB2 database manager just before it unloads the client authentication plug-in. It should be implemented in a manner that it does a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources in order to free them. This API is not called on any Windows platform.

### **API and data structure syntax**

```
SQL_API_RC (SQL_API_FN *db2secClientAuthPluginTerm)
(char **errmsg,
 db2int32 *errormsglen);
```

## db2secClientAuthPluginTerm API parameters

### errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secClientAuthPluginTerm API execution is not successful.

### errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## db2secRemapUserid API - Remap user ID and password

This API is called by the DB2 database manager on the client side to remap a given user ID and password (and possibly new password and username space) to values different from those given at connect time.

The DB2 database manager only calls this API if a user ID and a password are supplied at connect time. This prevents a plug-in from remapping a user ID by itself to a user ID/password pair. This API is optional and is not called if it is not provided or implemented by the security plug-in.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secRemapUserid)
(char userid[DB2SEC_MAX_USERID_LENGTH],
 db2int32 *useridlen,
 char usernamepace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
 db2int32 *usernamepacelen,
 db2int32 *usernamepacetype,
 char password[DB2SEC_MAX_PASSWORD_LENGTH],
 db2int32 *passwordlen,
 char newpasswd[DB2SEC_MAX_PASSWORD_LENGTH],
 db2int32 *newpasswdlen,
 const char *dbname,
 db2int32 dbnameelen,
 char **errmsg,
 db2int32 *errormsglen);
```

### db2secRemapUserid API parameters

**userid** Input or output. The user ID to be remapped. If there is an input user ID value, then the API must provide an output user ID value that can be the same or different from the input user ID value. If there is no input user ID value, then the API should not return an output user ID value.

### useridlen

Input or output. Length in bytes of the userid parameter value.

### usernamepace

Input or output. The namespace of the user ID. This value can optionally be remapped. If no input parameter value is specified, but an output value is returned, then the usernamepace will only be used by the DB2 database manager for CLIENT type authentication and is disregarded for other authentication types.

### usernamepacelen

Input or output. Length in bytes of the usernamepace parameter value. Under the limitation that the usernamepacetype parameter must be set to the value DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE (defined in db2secPlugin.h), the maximum length currently supported is 15 bytes.

**usernamespacetype**

Input or output. Old and new namespace type value. Currently, the only supported namespace type value is DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE (corresponds to a username style like domain\myname).

**password**

Input or output. As an input, it is the password that is to be remapped. As an output it is the remapped password. If an input value is specified for this parameter, the API must be able to return an output value that differs from the input value. If no input value is specified, the API must not return an output password value.

**passwordlen**

Input or output. Length in bytes of the password parameter value.

**newpasswd**

Input or output. As an input, it is the new password that is to be set. As an output it is the confirmed new password.

**Note:** This is the new password that is passed by the DB2 database manager into the newpassword parameter of the db2secValidatePassword API on the client or the server (depending on the value of the authentication database manager configuration parameter). If a new password was passed as input, then the API must be able to return an output value and it can be a different new password. If there is no new password passed in as input, then the API should not return an output new password.

**newpasswdlen**

Input or output. Length in bytes of the newpasswd parameter value.

**dbname**

Input. Name of the database to which the client is connecting.

**dbnamelen**

Input. Length in bytes of the dbname parameter value.

**errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secRemapUserId API execution is not successful.

**errmsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## db2secGetDefaultLoginContext API - Get default login context

Determines the user associated with the default login context, in other words, determines the DB2 authid of the user invoking a DB2 command without explicitly specifying a user ID (either an implicit authentication to a database, or a local authorization). This API must return both an authid and a user ID.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secGetDefaultLoginContext)
(char authid[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32 *authidlen,
 char userid[DB2SEC_MAX_USERID_LENGTH],
 db2int32 *useridlen,
 db2int32 useridtype,
```

```

char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
db2int32 *usernamespacelen,
db2int32 *usernamespacectype,
const char *dbname,
db2int32 dbnamehlen,
void **token,
char **errormsg,
db2int32 *errormsglen);

```

## db2secGetDefaultLoginContext API parameters

**authid** Output. The parameter in which the authid should be returned. The returned value must conform to DB2 authid naming rules, or the user will not be authorized to perform the requested action.

### authidlen

Output. Length in bytes of the authid parameter value.

**userid** Output. The parameter in which the user ID associated with the default login context should be returned.

### useridlen

Output. Length in bytes of the userid parameter value.

### useridtype

Input. Indicates if the real or effective user ID of the process is being specified. On Windows, only the real user ID exists. On UNIX and Linux, the real user ID and effective user ID can be different if the uid user ID for the application is different than the ID of the user executing the process. Valid values for the userid parameter (defined in db2secPlugin.h) are:

#### DB2SEC\_PLUGIN\_REAL\_USER\_NAME

Indicates that the real user ID is being specified.

#### DB2SEC\_PLUGIN\_EFFECTIVE\_USER\_NAME

Indicates that the effective user ID is being specified.

**Note:** Some plug-in implementations might not distinguish between the real and effective userid. In particular, a plug-in that does not use the UNIX or Linux identity of the user to establish the DB2 authorization ID can safely ignore this distinction.

### usernamespace

Output. The namespace of the user ID.

### usernamespacehlen

Output. Length in bytes of the usernamespace parameter value. Under the limitation that the usernamespacectype parameter must be set to the value DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE (defined in db2secPlugin.h), the maximum length currently supported is 15 bytes.

### usernamespacectype

Output. Namespacectype value. Currently, the only supported namespace type is DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE (corresponds to a username style like domain\myname).

### dbname

Input. Contains the name of the database being connected to, if this call is being used in the context of a database connection. For local authorization actions or instance attachments, this parameter is set to NULL.

### dbnamehlen

Input. Length in bytes of the dbname parameter value.

**token** Output. This is a pointer to data allocated by the plug-in that it might pass to subsequent authentication calls in the plug-in, or possibly to the group retrieval plug-in. The structure of this data is determined by the plug-in writer.

**errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGetDefaultLoginContext API execution is not successful.

**errmsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## db2secGenerateInitialCred API - Generate initial credentials

The db2secGenerateInitialCred API obtains the initial GSS-API credentials based on the user ID and password that are passed in.

For Kerberos, this is the ticket-granting ticket (TGT). The credential handle that is returned in pGSSCredHandle parameter is the handle that is used with the gss\_init\_sec\_context API and must be either an INITIATE or BOTH credential. The db2secGenerateInitialCred API is only called when a user ID, and possibly a password are supplied. Otherwise, the DB2 database manager specifies the value GSS\_C\_NO\_CREDENTIAL when calling the gss\_init\_sec\_context API to signify that the default credential obtained from the current login context is to be used.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secGenerateInitialCred)
(const char *userid,
 db2int32 useridlen,
 const char *usernamespace,
 db2int32 usernamespacelen,
 db2int32 usernamespacestype,
 const char *password,
 db2int32 passwordlen,
 const char *newpassword,
 db2int32 newpasswordlen,
 const char *dbname,
 db2int32 dbnameLen,
 gss_cred_id_t *pGSSCredHandle,
 void **InitInfo,
 char **errmsg,
 db2int32 *errmsglen);
```

### db2secGenerateInitialCred API parameters

**userid** Input. The user ID whose password is to be verified on the database server.

**useridlen**

Input. Length in bytes of the userid parameter value.

**usernamespace**

Input. The namespace from which the user ID was obtained.

**usernamespaceLen**

Input. Length in bytes of the usernamespace parameter value.

**usernamespacestype**

Input. The type of namespace.



**password**

Input. The password to be verified.

**passwordlen**

Input. Length in bytes of the password parameter value.

**newpassword**

Input. A new password if the password is to be changed. If no change is requested, the newpassword parameter is set to NULL. If it is not NULL, the API should validate the old password before setting the password to its new value. The API does not have to honour a request to change the password, but if it does not, it should immediately return with the return value DB2SEC\_PLUGIN\_CHANGEPASSWORD\_NOTSUPPORTED without validating the old password.

**newpasswordlen**

Input. Length in bytes of the newpassword parameter value.

**dbname**

Input. The name of the database being connected to. The API is free to ignore this parameter, or the API can return the value DB2SEC\_PLUGIN\_CONNECTION\_DISALLOWED if it has a policy of restricting access to certain databases to users who otherwise have valid passwords.

**dbnamelen**

Input. Length in bytes of the dbname parameter value.

**pGSSCredHandle**

Output. Pointer to the GSS-API credential handle.

**InitInfo**

Output. A pointer to data that is not known to DB2. The plug-in can use this memory to maintain a list of resources that are allocated in the process of generating the credential handle. The DB2 database manager calls the db2secFreeInitInfo API at the end of the authentication process, at which point these resources are freed. If the db2secGenerateInitialCred API does not need to maintain such a list, then it should return NULL.

**errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGenerateInitialCred API execution is not successful.

**Note:** For this API, error messages should not be created if the return value indicates a bad user ID or password. An error message should only be returned if there is an internal error in the API that prevented it from completing properly.

**errormsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## **db2secValidatePassword API - Validate password**

Provides a method for performing user ID and password style authentication during a database connect operation.

**Note:** When the API is run on the client side, the API code is run with the privileges of the user executing the CONNECT statement. This API will only be called on the client side if the authentication configuration parameter is set to CLIENT.

When the API is run on the server side, the API code is run with the privileges of the instance owner.

The plug-in writer should take the above into consideration if authentication requires special privileges (such as root level system access on UNIX).

This API must return the value DB2SEC\_PLUGIN\_OK (success) if the password is valid, or an error code such as DB2SEC\_PLUGIN\_BADPWD if the password is invalid.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secValidatePassword)
(const char *userid,
 db2int32 useridlen,
 const char *usernamespace,
 db2int32 usernamespaceLen,
 db2int32 usernamespaceType,
 const char *password,
 db2int32 passwordlen,
 const char *newpasswd,
 db2int32 newpasswdlen,
 const char *dbname,
 db2int32 dbnameLen,
 db2UInt32 connection_details,
 void **token,
 char **errorMsg,
 db2int32 *errorMsgLen);
```

### db2secValidatePassword API parameters

**userid** Input. The user ID whose password is to be verified.

**useridlen**

Input. Length in bytes of the userid parameter value.

**usernamespace**

Input. The namespace from which the user ID was obtained.

**usernamespaceLen**

Input. Length in bytes of the usernamespace parameter value.

**usernamespaceType**

Input. The type of namespace. Valid values for the usernamespaceType parameter (defined in db2secPlugin.h) are:

- DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE Corresponds to a username style like domain\myname
- DB2SEC\_NAMESPACE\_USER\_PRINCIPAL Corresponds to a username style like myname@domain.ibm.com

Currently, the DB2 database system only supports the value DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE. When the user ID is not available, the usernamespaceType parameter is set to the value DB2SEC\_USER\_NAMESPACE\_UNDEFINED (defined in db2secPlugin.h).

**password**

Input. The password to be verified.

**passwordlen**

Input. Length in bytes of the password parameter value.

**newpasswd**

Input. A new password, if the password is to be changed. If no change is requested, this parameter is set to NULL. If this parameter is not NULL, the API should validate the old password before changing it to the new password. The API does not have to fulfill a request to change the password, but if it does not, it should immediately return with the return value `DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED` without validating the old password.

**newpasswdlen**

Input. Length in bytes of the newpasswd parameter value.

**dbname**

Input. The name of the database being connected to. The API is free to ignore the dbname parameter, or it can return the value `DB2SEC_PLUGIN_CONNECTIONREFUSED` if it has a policy of restricting access to certain databases to users who otherwise have valid passwords. This parameter can be NULL.

**dbnamelen**

Input. Length in bytes of the dbname parameter value. This parameter is set to 0 if dbname parameter is NULL.

**connection\_details**

Input. A 32-bit parameter of which 3 bits are currently used to store the following information:

- The rightmost bit indicates whether the source of the user ID is the default from the `db2secGetDefaultLoginContext` API, or was explicitly provided during the connect.
- The second-from-right bit indicates whether the connection is local (using Inter Process Communication (IPC) or a connect from one of the nodes in the `db2nodes.cfg` in the partitioned database environment), or remote (through a network or loopback). This gives the API the ability to decide whether clients on the same machine can connect to the DB2 server without a password. Due to the default operating-system-based user ID/password plugin, local connections are permitted without a password from clients on the same machine (assuming the user has connect privileges).
- The third-from-right bit indicates whether the DB2 database manager is calling the API from the server side or client side.

The bit values are defined in `db2secPlugin.h`:

- `DB2SEC_USERID_FROM_OS` (0x00000001) Indicates that the user ID is obtained from OS and not explicitly given on the connect statement.
- `DB2SEC_CONNECTION_ISLOCAL` (0x00000002) Indicates a local connection.
- `DB2SEC_VALIDATING_ON_SERVER_SIDE` (0x00000004) Indicates whether the DB2 database manager is calling from the server side or client side to validate password. If this bit value is set, then the DB2 database manager is calling from server side; otherwise, it is calling from the client side.

The DB2 database system default behavior for an implicit authentication is to allow the connection without any password validation. However, plug-in developers have the option to disallow implicit authentication by returning a `DB2SEC_PLUGIN_BADPASSWORD` error.

**token** Input. A pointer to data which can be passed as a parameter to subsequent API calls during the current connection. Possible APIs that might be called include db2secGetAuthIDs API and db2secGetGroupsForUser API.

**errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secValidatePassword API execution is not successful.

**errormsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## db2secProcessServerPrincipalName API - Process service principal name returned from server

The db2secProcessServerPrincipalName API processes the service principal name returned from the server and returns the principal name in the gss\_name\_t internal format to be used with the gss\_init\_sec\_context API.

The db2secProcessServerPrincipalName API also processes the service principal name cataloged with the database directory when Kerberos authentication is used. Ordinarily, this conversion uses the gss\_import\_name API. After the context is established, the gss\_name\_t object is freed through the call to gss\_release\_name API. The db2secProcessServerPrincipalName API returns the value DB2SEC\_PLUGIN\_OK if gssName parameter points to a valid GSS name; a DB2SEC\_PLUGIN\_BAD\_PRINCIPAL\_NAME error code is returned if the principal name is invalid.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secProcessServerPrincipalName)
(const char *name,
 db2int32 namelen,
 gss_name_t *gssName,
 char **errmsg,
 db2int32 *errormsglen);
```

### db2secProcessServerPrincipalName API parameters

**name** Input. Text name of the service principal in GSS\_C\_NT\_USER\_NAME format; for example, service/host@REALM.

**namelen**

Input. Length in bytes of the name parameter value.

**gssName**

Output. Pointer to the output service principal name in the GSS-API internal format.

**errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secProcessServerPrincipalName API execution is not successful.

**errormsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## db2secFreeToken API - Free memory held by token

Frees the memory held by a token. This API is called by the DB2 database manager when it no longer needs the memory held by the token parameter.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secFreeToken)
(void *token,
 char **errmsg,
 db2int32 *errormsglen);
```

### db2secFreeToken API parameters

**token** Input. Pointer to the memory to be freed.

#### errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secFreeToken API execution is not successful.

#### errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## db2secFreeInitInfo API - Clean up resources held by the db2secGenerateInitialCred

Frees any resources allocated by the db2secGenerateInitialCred API. This can include, for example, handles to underlying mechanism contexts or a credential cache created for the GSS-API credential cache.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secFreeInitInfo)
(void *initinfo,
 char **errmsg,
 db2int32 *errormsglen);
```

### db2secFreeInitInfo API parameters

#### initinfo

Input. A pointer to data that is not known to the DB2 database manager. The plug-in can use this memory to maintain a list of resources that are allocated in the process of generating the credential handle. These resources are freed by calling this API.

#### errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secFreeInitInfo API execution is not successful.

#### errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## db2secServerAuthPluginInit - Initialize server authentication plug-in

The db2secServerAuthPluginInit API is the initialization API for the server authentication plug-in that the DB2 database manager calls immediately after loading the plug-in.

In the case of GSS-API, the plug-in is responsible for filling in the server's principal name in the `serverPrincipalName` parameter inside the `gssapi_server_auth_functions` structure at initialization time and providing the server's credential handle in the `serverCredHandle` parameter inside the `gssapi_server_auth_functions` structure. The freeing of the memory allocated to hold the principal name and the credential handle must be done by the `db2secServerAuthPluginTerm` API by calling the `gss_release_name` and `gss_release_cred` APIs.

## API and data structure syntax

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginInit
(db2int32 version,
 void *server_fns,
 db2secGetConDetails *getConDetails_fn,
 db2secLogMessage *logMessage_fn,
 char **errorMsg,
 db2int32 *errormsglen);
```

## db2secServerAuthPluginInit API parameters

### version

Input. The highest version number of the API that the DB2 database manager currently supports. The `DB2SEC_API_VERSION` value (in `db2secPlugin.h`) contains the latest version number of the API that the DB2 database manager currently supports.

### server\_fns

Output. A pointer to memory provided by the DB2 database manager for a `db2secGssapiServerAuthFunctions_<version_number>` structure (also known as `gssapi_server_auth_functions_<version_number>`), if GSS-API authentication is used, or a `db2secUseridPasswordServerAuthFunctions_<version_number>` structure (also known as `userid_password_server_auth_functions_<version_number>`), if `userid/password` authentication is used. The `db2secGssapiServerAuthFunctions_<version_number>` structure and `db2secUseridPasswordServerAuthFunctions_<version_number>` structure respectively contain pointers to the APIs implemented for the GSS-API authentication plug-in and `userid/password` authentication plug-in.

The `server_fns` parameter is cast as a pointer to the `gssapi_server_auth_functions_<version_number>` structure corresponding to the version the plug-in has implemented. The first parameter of the `gssapi_server_auth_functions_<version_number>` structure or the `userid_password_server_auth_functions_<version_number>` structure tells the DB2 database manager the version of the APIs that the plug-in has implemented.

**Note:** The casting is done only if the DB2 version is higher or equal to the version of the APIs that the plug-in has implemented.

Inside the `gssapi_server_auth_functions_<version_number>` or `userid_password_server_auth_functions_<version_number>` structure, the `plugintype` parameter should be set to one of `DB2SEC_PLUGIN_TYPE_USERID_PASSWORD`, `DB2SEC_PLUGIN_TYPE_GSSAPI`, or `DB2SEC_PLUGIN_TYPE_KERBEROS`. Other values can be defined in future versions of the API.

### **getConDetails\_fn**

Input. Pointer to the db2secGetConDetails API, which is implemented by DB2. The db2secServerAuthPluginInit API can call the db2secGetConDetails API in any one of the other authentication APIs to obtain details related to the database connection. These details include information about the communication mechanism associated with the connection (such as the IP address, in the case of TCP/IP), which the plug-in writer might need to reference when making authentication decisions. For example, the plug-in could disallow a connection for a particular user, unless that user is connecting from a particular IP address. The use of the db2secGetConDetails API is optional.

If the db2secGetConDetails API is called in a situation not involving a database connection, it returns the value DB2SEC\_PLUGIN\_NO\_CON\_DETAILS, otherwise, it returns 0 on success.

The db2secGetConDetails API takes two input parameters; pConDetails, which is a pointer to the db2sec\_con\_details\_<version\_number> structure, and conDetailsVersion, which is a version number indicating which db2sec\_con\_details structure to use. Possible values are DB2SEC\_CON\_DETAILS\_VERSION\_1 when db2sec\_con\_details1 is used or DB2SEC\_CON\_DETAILS\_VERSION\_2 when db2sec\_con\_details2. The recommended version number to use is DB2SEC\_CON\_DETAILS\_VERSION\_2.

Upon a successful return, the db2sec\_con\_details structure (either db2sec\_con\_details1 or db2sec\_con\_details2) will contain the following information:

- The protocol used for the connection to the server. The listing of protocol definitions can be found in the file sqlenv.h (located in the include directory) (SQL\_PROTOCOL\_\*). This information is filled out in the clientProtocol parameter.
- The TCP/IP address of the inbound connect to the server if the clientProtocol is SQL\_PROTOCOL\_TCPIP or SQL\_PROTOCOL\_TCPIP4. This information is filled out in the clientIPAddress parameter.
- The database name the client is attempting to connect to. This will not be set for instance attachments. This information is filled out in the dbname and dbnameLen parameters.
- A connection information bit-map that contains the same details as documented in the connection\_details parameter of the db2secValidatePassword API. This information is filled out in the connect\_info\_bitmap parameter.
- The TCP/IP address of the inbound connect to the server if the clientProtocol is SQL\_PROTOCOL\_TCPIP6. This information is filled out in the clientIP6Address parameter and it is only available if DB2SEC\_CON\_DETAILS\_VERSION\_2 is used for db2secGetConDetails API call.

### **logMessage\_fn**

Input. A pointer to the db2secLogMessage API, which is implemented by the DB2 database manager. The db2secClientAuthPluginInit API can call the db2secLogMessage API to log messages to the db2diag log files for debugging or informational purposes. The first parameter (level) of db2secLogMessage API specifies the type of diagnostic errors that will be recorded in the db2diag log files and the last two parameters respectively



are the message string and its length. The valid values for the first parameter of dbsecLogMessage API (defined in db2secPlugin.h) are:

**DB2SEC\_LOG\_NONE (0)**

No logging

**DB2SEC\_LOG\_CRITICAL (1)**

Severe Error encountered

**DB2SEC\_LOG\_ERROR (2)**

Error encountered

**DB2SEC\_LOG\_WARNING (3)**

Warning

**DB2SEC\_LOG\_INFO (4)**

Informational

The message text will appear in the db2diag log files only if the value of the 'level' parameter of the db2secLogMessage API is less than or equal to the diaglevel database manager configuration parameter.

So for example, if you use the DB2SEC\_LOG\_INFO value, the message text will only appear in the db2diag log files if the diaglevel database manager configuration parameter is set to 4.

**errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secServerAuthPluginInit API execution is not successful.

**errmsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## **db2secServerAuthPluginTerm API - Clean up server authentication plug-in resources**

The db2secServerAuthPluginTerm API frees resources used by the server authentication plug-in.

This API is called by the DB2 database manager just before it unloads the server authentication plug-in. It should be implemented in a manner that it does a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources in order to free them. This API is not called on any Windows platform.

### **API and data structure syntax**

```
SQL_API_RC (SQL_API_FN *db2secServerAuthPluginTerm)
(char **errmsg,
 db2int32 *errmsglen);
```

### **db2secServerAuthPluginTerm API parameters**

**errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secServerAuthPluginTerm API execution is not successful.

**errmsgslen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

**db2secGetAuthIDs API - Get authentication IDs**

Returns an SQL authid for an authenticated user. This API is called during database connections for both user ID/password and GSS-API authentication methods.

**API and data structure syntax**

```
SQL_API_RC (SQL_API_FN *db2secGetAuthIDs)
(const char *userid,
 db2int32 useridlen,
 const char *usernamespace,
 db2int32 usernamespace,
 db2int32 usernamespace,
 const char *dbname,
 db2int32 dbnamelen,
 void **token,
 char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32 *SystemAuthIDlen,
 char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32 *InitialSessionAuthIDlen,
 char username[DB2SEC_MAX_USERID_LENGTH],
 db2int32 *username,
 db2int32 *initsessionidtype,
 char **errmsg,
 db2int32 *errmsgslen);
```

**db2secGetAuthIDs API parameters**

**userid** Input. The authenticated user. This is usually not used for GSS-API authentication unless a trusted context is defined to permit switch user operations without authentication. In those situations, the user name provided for the switch user request is passed in this parameter.

**useridlen**

Input. Length in bytes of the userid parameter value.

**usernamespace**

Input. The namespace from which the user ID was obtained.

**usernamespace**

Input. Length in bytes of the usernamespace parameter value.

**usernamespace**

Input. Namespace type value. currently, the only supported namespace type value is DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE (corresponds to a username style like domain\myname).

**dbname**

Input. The name of the database being connected to. The API can ignore this, or it can return differing authids when the same user connects to different databases. This parameter can be NULL.

**dbnamelen**

Input. Length in bytes of the dbname parameter value. This parameter is set to 0 if dbname parameter is NULL.

**token**

Input or output. Data that the plug-in might pass to the db2secGetGroupsForUser API. For GSS-API, this is a context handle (gss\_ctx\_id\_t). Ordinarily, token is an input-only parameter and its value is

taken from the db2secValidatePassword API. It can also be an output parameter when authentication is done on the client and therefore db2secValidatePassword API is not called. In environments where a trusted context is defined that allows switch user operations without authentication, the db2secGetAuthIDs API must be able to accommodate receiving a NULL value for this token parameter and be able to derive a system authorization ID based on the userid and useridlen input parameters above.

**SystemAuthID**

Output. The system authorization ID that corresponds to the ID of the authenticated user. The size is 255 bytes, but the DB2 database manager currently uses only up to (and including) 30 bytes.

**SystemAuthIDlen**

Output. Length in bytes of the SystemAuthID parameter value.

**InitialSessionAuthID**

Output. Authid used for this connection session. This is usually the same as the SystemAuthID parameter but can be different in some situations, for instance, when issuing a SET SESSION AUTHORIZATION statement. The size is 255 bytes, but the DB2 database manager currently uses only up to (and including) 30 bytes.

**InitialSessionAuthIDlen**

Output. Length in bytes of the InitialSessionAuthID parameter value.

**username**

Output. A username corresponding to the authenticated user and authid. This will only be used for auditing and will be logged in the "User ID" field in the audit record for CONNECT statement. If the API does not fill in the username parameter, the DB2 database manager copies it from the userid.

**usernameflen**

Output. Length in bytes of the username parameter value.

**initsessionidtype**

Output. Session authid type indicating whether or not the InitialSessionAuthid parameter is a role or an authid. The API should return one of the following values (defined in db2secPlugin.h):

- DB2SEC\_ID\_TYPE\_AUTHID (0)
- DB2SEC\_ID\_TYPE\_ROLE (1)

**errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGetAuthIDs API execution is not successful.

**errmsgflen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

## **db2secDoesAuthIDExist - Check if authentication ID exists**

Determines if the authid represents an individual user (for example, whether the API can map the authid to an external user ID).

The API should return the value DB2SEC\_PLUGIN\_OK if it is successful - the authid is valid, DB2SEC\_PLUGIN\_INVALID\_USERORGROUP if it is not valid, or DB2SEC\_PLUGIN\_USERSTATUSNOTKNOWN if the authid existence cannot be determined.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secDoesAuthIDExist)
 (const char *authid,
 db2int32 authidlen,
 char **errmsg,
 db2int32 *errmsglen);
```

### db2secDoesAuthIDExist API parameters

**authid** Input. The authid to validate. This is upper-cased, with no trailing blanks.

**authidlen**

Input. Length in bytes of the authid parameter value.

**errmsg**

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secDoesAuthIDExist API execution is not successful.

**errmsglen**

Output. A pointer to an integer that indicates the length of the error message string in errmsg parameter.

## GSS-API plug-in APIs

### Required APIs and definitions for GSS-API authentication plug-ins

Following is a complete list of GSS-APIs required for the DB2 security plug-in interface.

The supported APIs follow these specifications: *Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) and *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744). Before implementing a GSS-API based plug-in, you should have a complete understanding of these specifications.

Table 33. Required APIs and Definitions for GSS-API authentication plug-ins

| Name                 |                        | Description                                                          |
|----------------------|------------------------|----------------------------------------------------------------------|
| Client-side APIs     | gss_init_sec_context   | Initiate a security context with a peer application.                 |
| Server-side APIs     | gss_accept_sec_context | Accept a security context initiated by a peer application.           |
| Server-side APIs     | gss_display_name       | Convert an internal format name to text.                             |
| Common APIs          | gss_delete_sec_context | Delete an established security context.                              |
| Common APIs          | gss_display_status     | Obtain the text error message associated with a GSS-API status code. |
| Common APIs          | gss_release_buffer     | Delete a buffer.                                                     |
| Common APIs          | gss_release_cred       | Release local data structures associated with a GSS-API credential.  |
| Common APIs          | gss_release_name       | Delete internal format name.                                         |
| Required definitions | GSS_C_DELEG_FLAG       | Requests delegation.                                                 |

Table 33. Required APIs and Definitions for GSS-API authentication plug-ins (continued)

| Name                 |                           | Description                                                                                                             |
|----------------------|---------------------------|-------------------------------------------------------------------------------------------------------------------------|
| Required definitions | GSS_C_EMPTY_BUFFER        | Signifies that the <code>gss_buffer_desc</code> does not contain any data.                                              |
| Required definitions | GSS_C_GSS_CODE            | Indicates a GSS major status code.                                                                                      |
| Required definitions | GSS_C_INDEFINITE          | Indicates that the mechanism does not support context expiration.                                                       |
| Required definitions | GSS_C_MECH_CODE           | Indicates a GSS minor status code.                                                                                      |
| Required definitions | GSS_C_MUTUAL_FLAG         | Mutual authentication requested.                                                                                        |
| Required definitions | GSS_C_NO_BUFFER           | Signifies that the <code>gss_buffer_t</code> variable does not point to a valid <code>gss_buffer_desc</code> structure. |
| Required definitions | GSS_C_NO_CHANNEL_BINDINGS | No communication channel bindings.                                                                                      |
| Required definitions | GSS_C_NO_CONTEXT          | Signifies that the <code>gss_ctx_id_t</code> variable does not point to a valid context.                                |
| Required definitions | GSS_C_NO_CREDENTIAL       | Signifies that <code>gss_cred_id_t</code> variable does not point to a valid credential handle.                         |
| Required definitions | GSS_C_NO_NAME             | Signifies that the <code>gss_name_t</code> variable does not point to a valid internal name.                            |
| Required definitions | GSS_C_NO_OID              | Use default authentication mechanism.                                                                                   |
| Required definitions | GSS_C_NULL_OID_SET        | Use default mechanism.                                                                                                  |
| Required definitions | GSS_S_COMPLETE            | API completed successfully.                                                                                             |
| Required definitions | GSS_S_CONTINUE_NEEDED     | Processing is not complete and the API must be called again with the reply token received from the peer.                |

## Restrictions for GSS-API authentication plug-ins

The following is a list of restrictions for GSS-API authentication plug-ins.

- The default security mechanism is always assumed; therefore, there is no OID consideration.
- The only GSS services requested in `gss_init_sec_context()` are mutual authentication and delegation. The DB2 database manager always requests a ticket for delegation, but does not use that ticket to generate a new ticket.
- Only the default context time is requested.
- Context tokens from `gss_delete_sec_context()` are not sent from the client to the server and vice-versa.
- Anonymity is not supported.
- Channel binding is not supported
- If the initial credentials expire, the DB2 database manager does not automatically renew them.
- The GSS-API specification stipulates that even if `gss_init_sec_context()` or `gss_accept_sec_context()` fail, either function must return a token to send to

the peer. However, because of DRDA limitations, the DB2 database manager only sends a token if `gss_init_sec_context()` fails and generates a token on the first call.

---

## Security plug-in API versioning

The DB2 database system supports version numbering of the security plug-in APIs. These version numbers are integers starting with 1 for DB2 UDB, Version 8.2.

The version number that DB2 passes to the security plug-in APIs is the highest version number of the API that DB2 can support, and corresponds to the version number of the structure. If the plug-in can support a higher API version, it must return function pointers for the version that DB2 has requested. If the plug-in only supports a lower version of the API, the plug-in should fill in function pointers for the lower version. In either situation, the security plug-in APIs should return the version number for the API it is supporting in the version field of the functions structure.

For DB2, the version numbers of the security plug-ins will only change when necessary (for example, when there are changes to the parameters of the APIs). Version numbers will not automatically change with DB2 release numbers.

---

## Security plug-in samples

UNIX and Linux directories: The 'C' samples are located in `sqllib/samples/security/plugins` and the JCC GSS-API plugin samples (.java) are located in `sqllib/samples/java/jdbc`

Windows directory: The 'C' samples are located in `sqllib\samples\security\plugins` and the JCC GSS-API plugin samples (.java) are located in `sqllib\samples\java\jdbc`

*Table 34. Security plug-in sample program files*

| Sample program name                 | Program description                                                                                     |
|-------------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>combined.c</code>             | Combined user ID and password authentication and group lookup sample                                    |
| <code>group_file.c</code>           | Simple file-based group management plug-in sample                                                       |
| <code>gssapi_simple.c</code>        | Basic GSS-API authentication plug-in sample (both client and server)                                    |
| <code>IBMLDAPauthclient.c</code>    | Implements a client side DB2 security plugin that interacts with an LDAP user registry                  |
| <code>IBMLDAPauthserver.c</code>    | Implements a server side DB2 security plugin that interacts with an LDAP user registry                  |
| <code>IBMLDAPconfig.c</code>        | Contains functions related to finding and parsing the configuration file for a DB2 LDAP security plugin |
| <code>IBMLDAPgroups.c</code>        | Implements a DB2 security plugin for LDAP-based group lookup                                            |
| <code>IBMLDAPutils.c</code>         | Contains utility functions used in the DB2 LDAP security plugin                                         |
| <code>IBMLDAPutils.h</code>         | LDAP security plugin header file                                                                        |
| <code>JCCKerberosPlugin.java</code> | Implements a GSS-API Plugin that does Kerberos authentication using IBM DB2 Universal Driver            |

Table 34. Security plug-in sample program files (continued)

| Sample program name         | Program description                                                                                |
|-----------------------------|----------------------------------------------------------------------------------------------------|
| JCCKerberosPluginTest.java  | Uses JCCKerberosPlugin to get a DB2 Connection using IBM DB2 Universal Driver.                     |
| JCCSimpleGSSPlugin.java     | Implements a GSS-API Plugin that does userid and password checking using IBM DB2 Universal Driver. |
| JCCSimpleGSSContext.java    | Implements a GSSContext to be used by JCCSimpleGSSPlugin                                           |
| JCCSimpleGSSCredential.java | Implements a GSSCredential to be used by JCCSimpleGSSPlugin                                        |
| JCCSimpleGSSException.java  | Implements a GSSException to be used by JCCSimpleGSSPlugin                                         |
| JCCSimpleGSSName.java       | Implements a GSSName to be used by JCCSimpleGSSPlugin                                              |
| JCCSimpleGSSPluginTest.java | Uses JCCSimpleGSSPlugin to get a DB2 Connection using IBM DB2 Universal Driver.                    |



---

## Chapter 12. Security Plug-In Configuration Parameters

---

### clnt\_krb\_plugin - Client Kerberos plug-in

This parameter specifies the name of the default Kerberos plug-in library to be used for client-side authentication and local authorization.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

Null or IBMkrb5 [any valid string ]

By default, the value is null on Linux and UNIX systems, and IBMkrb5 on Windows operating systems. The plug-in is used when the client is authenticated using KERBEROS authentication, or when local authorization is performed and the authentication type in the DBM CFG is KERBEROS.

---

### clnt\_pw\_plugin - Client userid-password plug-in

This parameter specifies the name of the userid-password plug-in library to be used for client-side authentication and local authorization.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

Null [any valid string ]

By default, the value is null and the DB2-supplied userid-password plug-in library is used. The plug-in is used when the client is authenticated using CLIENT authentication, or when local authorization is performed and the authentication type in the DBM CFG is CLIENT, SERVER, SERVER\_ENCRYPT or DATA\_ENCRYPT. For non-root installations, if the DB2 userid and password plug-in library is used, the db2rfe command must be run before using your DB2 product.

---

## group\_plugin - Group plug-in

This parameter specifies the name of the group plug-in library.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default [range]

Null [any valid string ]

By default, this value is null, and DB2 uses the operating system group lookup. The plug-in will be used for all group lookups. For non-root installations, if the DB2 userid and password plug-in library is used, the db2rfe command must be run before using your DB2 product.

---

## local\_gssplugin - GSS API plug-in used for local instance level authorization

This parameter specifies the name of the default GSS API plug-in library to be used for instance level local authorization when the value of the *authentication* database manager configuration parameter is set to GSSPLUGIN or GSS\_SERVER\_ENCRYPT.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default [range]

Null [any valid string ]

---

## srvcon\_auth - Authentication type for incoming connections at the server

This parameter specifies how and where user authentication is to take place when handling incoming connections at the server; it is used to override the current authentication type.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**  
Configurable

**Default [range]**  
Null [CLIENT; SERVER; SERVER\_ENCRYPT; KERBEROS;  
KRB\_SERVER\_ENCRYPT; GSSPLUGIN; GSS\_SERVER\_ENCRYPT ]

If a value is not specified, DB2 uses the value of the *authentication* database manager configuration parameter.

For a description of each authentication type, see “authentication - Authentication type” on page 678.

## srvcon\_gssplugin\_list - List of GSS API plug-ins for incoming connections at the server

This parameter specifies the GSS API plug-in libraries that are supported by the database server. It handles incoming connections at the server when the *srvcon\_auth* parameter is specified as KERBEROS, KRB\_SERVER\_ENCRYPT, GSSPLUGIN or GSS\_SERVER\_ENCRYPT, or when *srvcon\_auth* is not specified, and authentication is specified as KERBEROS, KRB\_SERVER\_ENCRYPT, GSSPLUGIN or GSS\_SERVER\_ENCRYPT.

**Configuration type**  
Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**  
Configurable

**Default [range]**  
Null [any valid string ]

By default, the value is null. If the authentication type is GSSPLUGIN and this parameter is NULL, an error is returned. If the authentication type is KERBEROS and this parameter is NULL, the DB2-supplied kerberos module or library is used. This parameter is not used if another authentication type is used.

When the authentication type is KERBEROS and the value of this parameter is not NULL, the list must contain exactly one Kerberos plug-in, and that plug-in is used for authentication (all other GSS plug-ins in the list are ignored). If there is more than one Kerberos plug-in, an error is returned.

Each GSS API plug-in name must be separated by a comma (,) with no space either before or after the comma. Plug-in names should be listed in the order of preference.

---

## srvcon\_pw\_plugin - Userid-password plug-in for incoming connections at the server

This parameter specifies the name of the default userid-password plug-in library to be used for server-side authentication. It handles incoming connections at the server when the *srvcon\_auth* parameter is specified as CLIENT, SERVER, SERVER\_ENCRYPT, or DATA\_ENCRYPT or when *srvcon\_auth* is not specified, and *authentication* is specified as CLIENT, SERVER, SERVER\_ENCRYPT, or DATA\_ENCRYPT.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default [range]

Null [any valid string ]

By default, the value is null and the DB2-supplied userid-password plug-in library is used. The plug-in will be used for all group lookups. For non-root installations, if the DB2 userid and password plug-in library is used, the db2rfe command must be run before using your DB2 product.

---

## srv\_plugin\_mode - Server plug-in mode

This parameter specifies whether plug-ins are to run in fenced mode or unfenced mode. Unfenced mode is the only supported mode.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default [range]

UNFENCED

---

## Part 5. Configuration Parameters



---

## Chapter 13. Configuration Parameters

---

### Configuration parameters

When a DB2 database instance or a database is created, a corresponding configuration file is created with default parameter values. You can modify these parameter values to improve performance and other characteristics of the instance or database.

The disk space and memory allocated by the database manager on the basis of default values of the parameters might be sufficient to meet your needs. In some situations, however, you might not be able to achieve maximum performance using these default values.

*Configuration files* contain parameters that define values such as the resources allocated to the DB2 database products and to individual databases, and the diagnostic level. There are two types of configuration files:

- The database manager configuration file for each DB2 instance
- The database configuration file for each individual database.

The *database manager configuration file* is created when a DB2 instance is created. The parameters it contains affect system resources at the instance level, independent of any one database that is part of that instance. Values for many of these parameters can be changed from the system default values to improve performance or increase capacity, depending on your system's configuration.

There is one database manager configuration file for each client installation as well. This file contains information about the client enabler for a specific workstation. A subset of the parameters available for a server are applicable to the client.

Database manager configuration parameters are stored in a file named `db2system`. This file is created when the instance of the database manager is created. In Linux and UNIX environments, this file can be found in the `sql1lib` subdirectory for the instance of the database manager. In Windows, the default location of this file varies from edition to edition of the Windows family of operating systems; to verify the default directory on Windows, check the setting of the `DB2INSTPROF` registry variable using the command `DB2SET DB2INSTPROF`. You can also change the default instance directory by changing the `DB2INSTPROF` registry variable. If the `DB2INSTPROF` variable is set, the file is in the instance subdirectory of the directory specified by the `DB2INSTPROF` variable.

Other profile-registry variables that specify where run-time data files should go should query the value of `DB2INSTPROF`. This includes the following variables:

- `DB2CLINIPATH`
- `DIAGPATH`
- `SPM_LOG_PATH`

Database configuration parameters are stored in a file named `SQLDBC0N` for databases created before Version 8.2; all database configuration parameters are stored in a file named `SQLDBC0NF` for databases created in Version 8.2 and later. These files cannot be directly edited, and can only be changed or viewed via a supplied API or by a tool which calls that API.



In a partitioned database environment, this file resides on a shared file system so that all database partition servers have access to the same file. The configuration of the database manager is the same on all database partition servers.

Most of the parameters either affect the amount of system resources that will be allocated to a single instance of the database manager, or they configure the setup of the database manager and the different communications subsystems based on environmental considerations. In addition, there are other parameters that serve informative purposes only and cannot be changed. All of these parameters have global applicability independent of any single database stored under that instance of the database manager.

A *database configuration file* is created when a database is created, and resides where that database resides. There is one configuration file per database. Its parameters specify, among other things, the amount of resource to be allocated to that database. Values for many of the parameters can be changed to improve performance or increase capacity. Different changes may be required, depending on the type of activity in a specific database.

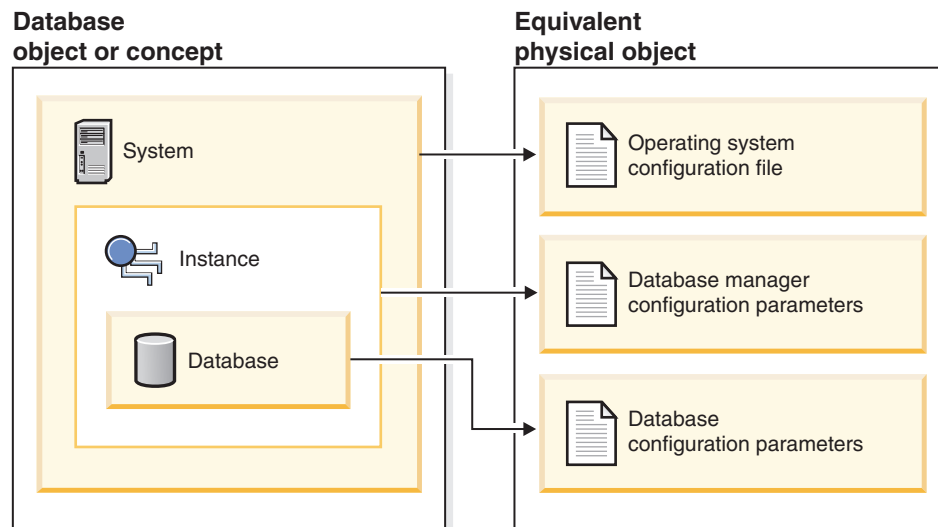


Figure 9. Relationship between database objects and configuration files

## Configuring the DB2 database manager with configuration parameters

The disk space and memory allocated by the database manager on the basis of default values of the parameters might be sufficient to meet your needs. In some situations, however, you might not be able to achieve maximum performance using these default values.

Since the default values are oriented towards machines that have relatively small memory resources and are dedicated as database servers, you might need to modify these values if your environment has:

- Large databases
- Large numbers of connections
- High performance requirements for a specific application
- Unique query or transaction loads or types

Each transaction processing environment is unique in one or more aspects. These differences can have a profound impact on the performance of the database manager when using the default configuration. For this reason, you are strongly advised to tune your configuration for your environment.

A good starting point for tuning your configuration is to use the Configuration Advisor or the AUTOCONFIGURE command which will generate values for parameters based on your responses to questions about workload characteristics.

Some configuration parameters can be set to AUTOMATIC, allowing the database manager to automatically manage these parameters to reflect the current resource requirements. To turn off the AUTOMATIC setting of a configuration parameter while maintaining the current internal setting, use the MANUAL keyword with the UPDATE DATABASE CONFIGURATION command. If the database manager updates the value of these parameters, the get db/dbm cfg show detail commands will show the new value.

Parameters for an individual database are stored in a configuration file named SQLDBCONF. This file is stored along with other control files for the database in the SQLnnnnn directory, where nnnnn is a number assigned when the database was created. Each database has its own configuration file, and most of the parameters in the file specify the amount of resources allocated to that database. The file also contains descriptive information, as well as flags that indicate the status of the database.

**Attention:** If you edit db2system, SQLDBCON, or SQLDBCONF using a method other than those provided by the database manager, you might make the database unusable. Do not change these files using methods other than those documented and supported by the database manager.

In a partitioned database environment, a separate SQLDBCONF file exists for each database partition. The values in the SQLDBCONF file may be the same or different at each database partition, but the recommendation is that in a homogeneous environment, the configuration parameter values should be the same on all database partitions. Typically, there could be a catalog node needing different database configuration parameters setting, while the other data partitions have different values again, depending on their machine types, and other information.

#### **Updating configuration parameters using the command line processor:**

Commands to change the settings can be entered as follows:

For database manager configuration parameters:

- GET DATABASE MANAGER CONFIGURATION (or GET DBM CFG)
- UPDATE DATABASE MANAGER CONFIGURATION (or UPDATE DBM CFG)
- RESET DATABASE MANAGER CONFIGURATION (or RESET DBM CFG) to reset *all* database manager parameters to their default values
- AUTOCONFIGURE.

For database configuration parameters:

- GET DATABASE CONFIGURATION (or GET DB CFG)
- UPDATE DATABASE CONFIGURATION (or UPDATE DB CFG)
- RESET DATABASE CONFIGURATION (or RESET DB CFG) to reset *all* database parameters to their default values
- AUTOCONFIGURE.

## Updating configuration parameters using application programming interfaces (APIs).

The APIs can be called from an application or a host-language program. Call the following DB2 APIs to view or update configuration parameters:

- db2AutoConfig - Access the Configuration Advisor
- db2CfgGet - Get the database manager or database configuration parameters
- db2CfgSet - Set the database manager or database configuration parameters

## Updating configuration parameters using common SQL application programming interface (API) procedures.

You can call the common SQL API procedures from an SQL-based application, a DB2 command line, or a command script. Call the following procedures to view or update configuration parameters:

- GET\_CONFIG - Get the database manager or database configuration parameters
- SET\_CONFIG - Set the database manager or database configuration parameters

## Updating configuration parameters using the Configuration Assistant

The Configuration Assistant can also be used to set the database manager configuration parameters on a client. Other parameters can be changed online; these are called *configurable online configuration parameters*.

## Viewing updated configuration values

For some database manager configuration parameters, the database manager must be stopped (db2stop) and then restarted (db2start) for the new parameter values to take effect.

For some database parameters, changes will only take effect when the database is reactivated, or switched from offline to online. In these cases, all applications must first disconnect from the database. (If the database was activated, or switched from offline to online, then it must be deactivated and reactivated.) Then, at the first new connect to the database, the changes will take effect.

If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default behavior of the UPDATE DBM CFG command will be to apply the change immediately. If you do not want the change applied immediately, use the DEFERRED option on the UPDATE DBM CFG command.

To change a database manager configuration parameter online:

```
db2 attach to <instance-name>
db2 update dbm cfg using <parameter-name> <value>
db2 detach
```

For clients, changes to the database manager configuration parameters take effect the next time the client connects to a server.

If you change a configurable online database configuration parameter while connected, the default behavior is to apply the change online, wherever possible. You should note that some parameter changes might take a noticeable amount of time to take effect due to the overhead associated with allocating space. To change configuration parameters online from the command line processor, a connection to the database is required. To change a database configuration parameter online:

```
db2 connect to <dbname>
db2 update db cfg using <parameter-name> <parameter-value>
db2 connect reset
```

Each configurable online configuration parameter has a *propagation class* associated with it. The propagation class indicates when you can expect a change to the configuration parameter to take effect. There are three propagation classes:

- **Immediate:** Parameters that change immediately upon command or API invocation. For example, *diaglevel* has a propagation class of immediate.
- **Statement boundary:** Parameters that change on statement and statement-like boundaries. For example, if you change the value of *sortheap*, all new requests will start using the new value.
- **Transaction boundary:** Parameters that change on transaction boundaries. For example, a new value for *dl\_expint* is updated after a COMMIT statement.

While new parameter values might not be immediately effective, viewing the parameter settings (using the GET DATABASE MANAGER CONFIGURATION or GET DATABASE CONFIGURATION command) will always show the latest updates. Viewing the parameter settings using the SHOW DETAIL clause on these commands will show both the latest updates and the values in memory.

### Rebinding applications after updating database configuration parameters

Changing some database configuration parameters can influence the access plan chosen by the SQL and XQuery optimizer. After changing any of these parameters, you should consider rebinding your applications to ensure the best access plan is being used for your SQL and XQuery statements. Any parameters that were modified online (for example, by using the UPDATE DATABASE CONFIGURATION IMMEDIATE command) will cause the SQL and XQuery optimizer to choose new access plans for new query statements. However, the query statement cache will not be purged of existing entries. To clear the contents of the query cache, use the FLUSH PACKAGE CACHE statement.

**Note:** A number of configuration parameters (for example, *userexit*) are described as having acceptable values of either “Yes” or “No”, or “On” or “Off” in the help and other DB2 documentation. To clarify, “Yes” should be considered equivalent to “On” and “No” should be considered equivalent to “Off”.

---

## Configuration parameters summary

The following tables list the parameters in the *database manager* and *database* configuration files for database servers. When changing the database manager and database configuration parameters, consider the detailed information for each parameter. Specific operating environment information including defaults is part of each parameter description.

### Database Manager Configuration Parameter Summary

For some database manager configuration parameters, the database manager must be stopped (db2stop) and restarted (db2start) for the new parameter values to take effect. Other parameters can be changed online; these are called *configurable online configuration parameters*. If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default

behavior of the UPDATE DBM CFG command applies the change immediately. If you do not want the change applied immediately, use the **DEFERRED** option on the UPDATE DBM CFG command.

The column “Auto” in the following table indicates whether the parameter supports the **AUTOMATIC** keyword on the UPDATE DBM CFG command.

When updating a parameter to automatic, it is also possible to specify a starting value as well as the **AUTOMATIC** keyword. Note that the value can mean something different for each parameter, and in some cases it is not applicable. Before specifying a value, read the parameter’s documentation to determine what it represents. In the following example, **num\_poolagents** will be updated to **AUTOMATIC** and DB2 will use 20 as the minimum number of idle agents to pool:

```
db2 update dbm cfg using num_poolagents 20 automatic
```

To unset the **AUTOMATIC** feature, the parameter can be updated to a value or the **MANUAL** keyword can be used. When a parameter is updated to **MANUAL**, the parameter is no longer automatic and is set to its current value (as displayed in the Current Value column from the GET DBM CFG SHOW DETAIL and GET DB CFG SHOW DETAIL commands).

The column “Perf. Impact” provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High** — indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters, which, in some cases, means that you will accept the default values provided.
- **Medium** — indicates that the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low** — indicates that the parameter has a less general or less significant impact on performance.
- **None** — indicates that the parameter does not directly impact performance. Although you do not have to tune these parameters for performance enhancement, they can be very important for other aspects of your system configuration, such as communication support, for example.

The columns “Token”, “Token Value”, and “Data Type” provide information that you will need when calling the db2CfgGet or the db2CfgSet API. This information includes configuration parameter identifiers, entries for the *token* element in the db2CfgParam data structure, and data types for values that are passed to the structure.

Table 35. Configurable Database Manager Configuration Parameters

| Parameter          | Cfg. Online | Auto. | Perf. Impact | Token                       | Token Value | Data Type | Additional Information                                                                                                       |
|--------------------|-------------|-------|--------------|-----------------------------|-------------|-----------|------------------------------------------------------------------------------------------------------------------------------|
| agent_stack_sz     | No          | No    | Low          | SQLF_KTN_AGENT_STACK_SZ     | 61          | UInt16    | “agent_stack_sz - Agent stack size” on page 749                                                                              |
| agentpri           | No          | No    | High         | SQLF_KTN_AGENTPRI           | 26          | Sint16    | “agentpri - Priority of agents” on page 751                                                                                  |
| alternate_auth_enc | No          | No    | Low          | SQLF_KTN_ALTERNATE_AUTH_ENC | 938         | UInt16    | “alternate_auth_enc - Alternate encryption algorithm for incoming connections at server configuration parameter” on page 752 |

Table 35. Configurable Database Manager Configuration Parameters (continued)

| Parameter                                                                                                                                             | Cfg. Online | Auto. | Perf. Impact | Token                                                                                                                                                                                                                                      | Token Value                                                | Data Type                                                                      | Additional Information                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|-------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|--------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| aslheapsz                                                                                                                                             | No          | No    | High         | SQLF_KTN_ASLHEAPSZ                                                                                                                                                                                                                         | 15                                                         | UInt32                                                                         | "aslheapsz - Application support layer heap size" on page 755            |
| audit_buf_sz                                                                                                                                          | No          | No    | High         | SQLF_KTN_AUDIT_BUF_SZ                                                                                                                                                                                                                      | 312                                                        | Sint32                                                                         | "audit_buf_sz - Audit buffer size" on page 677                           |
| authentication <sup>1</sup>                                                                                                                           | No          | No    | Low          | SQLF_KTN_AUTHENTICATION                                                                                                                                                                                                                    | 78                                                         | UInt16                                                                         | "authentication - Authentication type" on page 678                       |
| catalog_noauth                                                                                                                                        | Yes         | No    | None         | SQLF_KTN_CATALOG_NOAUTH                                                                                                                                                                                                                    | 314                                                        | UInt16                                                                         | "catalog_noauth - Cataloging allowed without authority" on page 680      |
| clnt_krb_plugin                                                                                                                                       | No          | No    | None         | SQLF_KTN_CLNT_KRB_PLUGIN                                                                                                                                                                                                                   | 812                                                        | char(33)                                                                       | "clnt_krb_plugin - Client Kerberos plug-in" on page 653                  |
| clnt_pw_plugin                                                                                                                                        | No          | No    | None         | SQLF_KTN_CLNT_PW_PLUGIN                                                                                                                                                                                                                    | 811                                                        | char(33)                                                                       | "clnt_pw_plugin - Client userid-password plug-in" on page 653            |
| cluster_mgr                                                                                                                                           | No          | No    | None         | SQLF_KTN_CLUSTER_MGR                                                                                                                                                                                                                       | 920                                                        | char(262)                                                                      | "cluster_mgr - Cluster manager name" on page 763                         |
| comm_bandwidth                                                                                                                                        | Yes         | No    | Medium       | SQLF_KTN_COMM_BANDWIDTH                                                                                                                                                                                                                    | 307                                                        | float                                                                          | "comm_bandwidth - Communications bandwidth" on page 764                  |
| conn_elapse                                                                                                                                           | Yes         | No    | Medium       | SQLF_KTN_CONN_ELAPSE                                                                                                                                                                                                                       | 508                                                        | UInt16                                                                         | "conn_elapse - Connection elapse time" on page 697                       |
| cpuspeed                                                                                                                                              | Yes         | No    | High         | SQLF_KTN_CPUSPEED                                                                                                                                                                                                                          | 42                                                         | float                                                                          | "cpuspeed - CPU speed" on page 766                                       |
| dft_account_str                                                                                                                                       | Yes         | No    | None         | SQLF_KTN_DFT_ACCOUNT_STR                                                                                                                                                                                                                   | 28                                                         | char(25)                                                                       | "dft_account_str - Default charge-back account" on page 774              |
| dft_monswitches<br>• dft_mon_bufpool<br>• dft_mon_lock<br>• dft_mon_sort<br>• dft_mon_stmt<br>• dft_mon_table<br>• dft_mon_timestamp<br>• dft_mon_uow | Yes         | No    | Medium       | SQLF_KTN_DFT_MONSWITCHES <sup>2</sup><br>• SQLF_KTN_DFT_MON_BUFPOOL<br>• SQLF_KTN_DFT_MON_LOCK<br>• SQLF_KTN_DFT_MON_SORT<br>• SQLF_KTN_DFT_MON_STMT<br>• SQLF_KTN_DFT_MON_TABLE<br>• SQLF_KTN_DFT_MON_TIMESTAMP<br>• SQLF_KTN_DFT_MON_UOW | 29<br>• 33<br>• 34<br>• 35<br>• 31<br>• 32<br>• 36<br>• 30 | UInt16<br>• UInt16<br>• UInt16<br>• UInt16<br>• UInt16<br>• UInt16<br>• UInt16 | "dft_monswitches - Default database system monitor switches" on page 777 |
| dftdbpath                                                                                                                                             | Yes         | No    | None         | SQLF_KTN_DFTDBPATH                                                                                                                                                                                                                         | 27                                                         | char(215)                                                                      | "dftdbpath - Default database path" on page 681                          |
| diaglevel                                                                                                                                             | Yes         | No    | Low          | SQLF_KTN_DIAGLEVEL                                                                                                                                                                                                                         | 64                                                         | UInt16                                                                         | "diaglevel - Diagnostic error capture level" on page 782                 |
| diagpath                                                                                                                                              | Yes         | No    | None         | SQLF_KTN_DIAGPATH                                                                                                                                                                                                                          | 65                                                         | char(215)                                                                      | "diagpath - Diagnostic data directory path" on page 782                  |
| dir_cache                                                                                                                                             | No          | No    | Medium       | SQLF_KTN_DIR_CACHE                                                                                                                                                                                                                         | 40                                                         | UInt16                                                                         | "dir_cache - Directory cache support" on page 783                        |
| discover <sup>3</sup>                                                                                                                                 | No          | No    | Medium       | SQLF_KTN_DISCOVER                                                                                                                                                                                                                          | 304                                                        | UInt16                                                                         | "discover - Discovery mode" on page 785                                  |
| discover_inst                                                                                                                                         | Yes         | No    | Low          | SQLF_KTN_DISCOVER_INST                                                                                                                                                                                                                     | 308                                                        | UInt16                                                                         | "discover_inst - Discover server instance" on page 786                   |
| fcm_num_buffers                                                                                                                                       | Yes         | Yes   | Medium       | SQLF_KTN_FCM_NUM_BUFFERS                                                                                                                                                                                                                   | 503                                                        | UInt32                                                                         | "fcm_num_buffers - Number of FCM buffers" on page 697                    |
| fcm_num_channels                                                                                                                                      | Yes         | Yes   | Medium       | SQLF_KTN_FCM_NUM_CHANNELS                                                                                                                                                                                                                  | 902                                                        | UInt32                                                                         | "fcm_num_channels - Number of FCM channels" on page 698                  |



Table 35. Configurable Database Manager Configuration Parameters (continued)

| Parameter             | Cfg. Online | Auto. | Perf. Impact | Token                    | Token Value | Data Type | Additional Information                                                                      |
|-----------------------|-------------|-------|--------------|--------------------------|-------------|-----------|---------------------------------------------------------------------------------------------|
| fed_noauth            | Yes         | No    | None         | SQLF_KTN_FED_NOAUTH      | 806         | UInt16    | "fed_noauth - Bypass federated authentication" on page 788                                  |
| federated             | Yes         | No    | Medium       | SQLF_KTN_FEDERATED       | 604         | Sint16    | "federated - Federated database system support" on page 789                                 |
| federated_async       | Yes         | Yes   | Medium       | SQLF_KTN_FEDERATED_ASYNC | 849         | Sint32    | "federated_async - Maximum asynchronous TQs per query configuration parameter" on page 789  |
| fenced_pool           | Yes         | Yes   | Medium       | SQLF_KTN_FENCED_POOL     | 80          | Sint32    | "fenced_pool - Maximum number of fenced processes" on page 790                              |
| group_plugin          | No          | No    | None         | SQLF_KTN_GROUP_PLUGIN    | 810         | char(33)  | "group_plugin - Group plug-in" on page 654                                                  |
| health_mon            | Yes         | No    | Low          | SQLF_KTN_HEALTH_MON      | 804         | UInt16    | "health_mon - Health monitoring" on page 795                                                |
| indexrec <sup>4</sup> | Yes         | No    | Medium       | SQLF_KTN_INDEXREC        | 20          | UInt16    | "indexrec - Index re-creation time" on page 796                                             |
| instance_memory       | Yes         | Yes   | Medium       | SQLF_KTN_INSTANCE_MEMORY | 803         | UInt64    | "instance_memory - Instance memory" on page 798                                             |
| intra_parallel        | No          | No    | High         | SQLF_KTN_INTRA_PARALLEL  | 306         | Sint16    | "intra_parallel - Enable intra-partition parallelism" on page 800                           |
| java_heap_sz          | No          | No    | High         | SQLF_KTN_JAVA_HEAP_SZ    | 310         | Sint32    | "java_heap_sz - Maximum Java interpreter heap size" on page 801                             |
| jdk_path              | No          | No    | None         | SQLF_KTN_JDK_PATH        | 311         | char(255) | "jdk_path - Software Developer's Kit for Java installation path" on page 803                |
| keepfenced            | No          | No    | Medium       | SQLF_KTN_KEEPFENCED      | 81          | UInt16    | "keepfenced - Keep fenced process" on page 803                                              |
| local_gssplugin       | No          | No    | None         | SQLF_KTN_LOCAL_GSSPLUGIN | 816         | char(33)  | "local_gssplugin - GSS API plug-in used for local instance level authorization" on page 654 |
| max_connections       | Yes         | Yes   | Medium       | SQLF_KTN_MAX_CONNECTIONS | 802         | Sint32    | "max_connections - Maximum number of client connections" on page 813                        |
| max_connretries       | Yes         | No    | Medium       | SQLF_KTN_MAX_CONNRETRIES | 509         | UInt16    | "max_connretries - Node connection retries" on page 699                                     |
| max_coordagents       | Yes         | Yes   | Medium       | SQLF_KTN_MAX_COORDAGENTS | 501         | Sint32    | "max_coordagents - Maximum number of coordinating agents" on page 814                       |
| max_querydegree       | Yes         | No    | High         | SQLF_KTN_MAX_QUERYDEGREE | 303         | Sint32    | "max_querydegree - Maximum query degree of parallelism" on page 815                         |
| max_time_diff         | No          | No    | Medium       | SQLF_KTN_MAX_TIME_DIFF   | 510         | UInt16    | "max_time_diff - Maximum time difference among nodes" on page 699                           |
| mon_heap_sz           | Yes         | Yes   | Low          | SQLF_KTN_MON_HEAP_SZ     | 79          | UInt16    | "mon_heap_sz - Database system monitor heap size" on page 821                               |
| notifylevel           | Yes         | No    | Low          | SQLF_KTN_NOTIFYLEVEL     | 605         | Sint16    | "notifylevel - Notify level" on page 824                                                    |
| num_initagents        | No          | No    | Medium       | SQLF_KTN_NUM_INITAGENTS  | 500         | UInt32    | "num_initagents - Initial number of agents in pool" on page 826                             |



Table 35. Configurable Database Manager Configuration Parameters (continued)

| Parameter             | Cfg. Online | Auto. | Perf. Impact | Token                          | Token Value | Data Type  | Additional Information                                                                                                 |
|-----------------------|-------------|-------|--------------|--------------------------------|-------------|------------|------------------------------------------------------------------------------------------------------------------------|
| num_initfenced        | No          | No    | Medium       | SQLF_KTN_NUM_INITFENCED        | 601         | Sint32     | "num_initfenced - Initial number of fenced processes" on page 827                                                      |
| num_poolagents        | Yes         | Yes   | High         | SQLF_KTN_NUM_POOLAGENTS        | 502         | Sint32     | "num_poolagents - Agent pool size" on page 830                                                                         |
| numdb                 | No          | No    | Low          | SQLF_KTN_NUMDB                 | 6           | Uint16     | "numdb - Maximum number of concurrently active databases including host and System i databases" on page 833            |
| query_heap_sz         | No          | No    | Medium       | SQLF_KTN_QUERY_HEAP_SZ         | 49          | Sint32     | "query_heap_sz - Query heap size" on page 837                                                                          |
| resync_interval       | No          | No    | None         | SQLF_KTN_RESYNC_INTERVAL       | 68          | Uint16     | "resync_interval - Transaction resync interval" on page 841                                                            |
| rqrioblk              | No          | No    | High         | SQLF_KTN_RQRIOBLK              | 1           | Uint16     | "rqrioblk - Client I/O block size" on page 842                                                                         |
| sheapthres            | No          | No    | High         | SQLF_KTN_SHEAPTHRES            | 21          | Uint32     | "sheapthres - Sort heap threshold" on page 845                                                                         |
| spm_log_file_sz       | No          | No    | Low          | SQLF_KTN_SPM_LOG_FILE_SZ       | 90          | Sint32     | "spm_log_file_sz - Sync point manager log file size" on page 851                                                       |
| spm_log_path          | No          | No    | Medium       | SQLF_KTN_SPM_LOG_PATH          | 313         | char(226)  | "spm_log_path - Sync point manager log file path" on page 852                                                          |
| spm_max_resync        | No          | No    | Low          | SQLF_KTN_SPM_MAX_RESYNC        | 91          | Sint32     | "spm_max_resync - Sync point manager resync agent limit" on page 853                                                   |
| spm_name              | No          | No    | None         | SQLF_KTN_SPM_NAME              | 92          | char(8)    | "spm_name - Sync point manager name" on page 853                                                                       |
| srvcon_auth           | No          | No    | None         | SQLF_KTN_SRVCON_AUTH           | 815         | Uint16     | "srvcon_auth - Authentication type for incoming connections at the server" on page 654                                 |
| srvcon_gssplugin_list | No          | No    | None         | SQLF_KTN_SRVCON_GSSPLUGIN_LIST | 814         | char(256)  | "srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server" on page 655                  |
| srv_plugin_mode       | No          | No    | None         | SQLF_KTN_SRV_PLUGIN_MODE       | 809         | Uint16     | "srv_plugin_mode - Server plug-in mode" on page 656                                                                    |
| srvcon_pw_plugin      | No          | No    | None         | SQLF_KTN_SRVCON_PW_PLUGIN      | 813         | char(33)   | "srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server" on page 656                        |
| ssl_svr_keydb         | No          | No    | None         | SQLF_KTN_SSL_SVR_KEYDB         | 930         | char(1023) | "ssl_svr_keydb - SSL key file path for incoming SSL connections at the server configuration parameter" on page 692     |
| ssl_svr_stash         | No          | No    | None         | SQLF_KTN_SSL_SVR_STASH         | 931         | char(1023) | "ssl_svr_stash - SSL stash file path for incoming SSL connections at the server configuration parameter" on page 692   |
| ssl_svr_label         | No          | No    | None         | SQLF_KTN_SSL_SVR_LABEL         | 932         | char(1023) | "ssl_svr_label - Label in the key file for incoming SSL connections at the server configuration parameter" on page 693 |
| ssl_svcename          | No          | No    | None         | SQLF_KTN_SSL_SVCENAME          | 933         | char(14)   | "ssl_svcename - SSL service name configuration parameter" on page 693                                                  |

Table 35. Configurable Database Manager Configuration Parameters (continued)

| Parameter                   | Cfg. Online | Auto. | Perf. Impact | Token                    | Token Value | Data Type  | Additional Information                                                                                                |
|-----------------------------|-------------|-------|--------------|--------------------------|-------------|------------|-----------------------------------------------------------------------------------------------------------------------|
| ssl_cipherspecs             | No          | No    | None         | SQLF_KTN_SSL_CIPHERSPECS | 934         | char(255)  | "ssl_cipherspecs - Supported cipher specifications at the server configuration parameter" on page 694                 |
| ssl_versions                | No          | No    | None         | SQLF_KTN_SSL_VERSIONS    | 935         | char(255)  | "ssl_versions - Supported SSL versions at the server configuration parameter" on page 694                             |
| ssl_clnt_keydb              | No          | No    | None         | SQLF_KTN_SSL_CLNT_KEYDB  | 936         | char(1023) | "ssl_clnt_keydb - SSL key file path for outbound SSL connections at the client configuration parameter" on page 695   |
| ssl_clnt_stash              | No          | No    | None         | SQLF_KTN_SSL_CLNT_STASH  | 937         | char(1023) | "ssl_clnt_stash - SSL stash file path for outbound SSL connections at the client configuration parameter" on page 695 |
| start_stop_time             | Yes         | No    | Low          | SQLF_KTN_START_STOP_TIME | 511         | UInt16     | "start_stop_time - Start and stop timeout" on page 700                                                                |
| svcname                     | No          | No    | None         | SQLF_KTN_SVCENAME        | 24          | char(14)   | "svcname - TCP/IP service name" on page 681                                                                           |
| sysadm_group                | No          | No    | None         | SQLF_KTN_SYSADM_GROUP    | 39          | char(128)  | "sysadm_group - System administration authority group name" on page 682                                               |
| sysctrl_group               | No          | No    | None         | SQLF_KTN_SYSCTRL_GROUP   | 63          | char(128)  | "sysctrl_group - System control authority group name" on page 683                                                     |
| sysmaint_group              | No          | No    | None         | SQLF_KTN_SYSMAINT_GROUP  | 62          | char(128)  | "sysmaint_group - System maintenance authority group name" on page 683                                                |
| sysmon_group                | No          | No    | None         | SQLF_KTN_SYSMON_GROUP    | 808         | char(128)  | "sysmon_group - System monitor authority group name" on page 684                                                      |
| tm_database                 | No          | No    | None         | SQLF_KTN_TM_DATABASE     | 67          | char(8)    | "tm_database - Transaction manager database name" on page 855                                                         |
| tp_mon_name                 | No          | No    | None         | SQLF_KTN_TP_MON_NAME     | 66          | char(19)   | "tp_mon_name - Transaction processor monitor name" on page 856                                                        |
| trust_allclnts <sup>5</sup> | No          | No    | None         | SQLF_KTN_TRUST_ALLCLNTS  | 301         | UInt16     | "trust_allclnts - Trust all clients" on page 685                                                                      |
| trust_clntauth              | No          | No    | None         | SQLF_KTN_TRUST_CLNTAUTH  | 302         | UInt16     | "trust_clntauth - Trusted clients authentication" on page 858                                                         |
| util_impact_lim             | Yes         | No    | High         | SQLF_KTN_UTIL_IMPACT_LIM | 807         | UInt32     | "util_impact_lim - Instance impact policy" on page 862                                                                |

Table 35. Configurable Database Manager Configuration Parameters (continued)

| Parameter                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Cfg. Online | Auto. | Perf. Impact | Token | Token Value | Data Type | Additional Information |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|-------|--------------|-------|-------------|-----------|------------------------|
| <p><b>Note:</b></p> <p>1. The valid values are defined in sqlenv.h.</p> <p>2.</p> <ul style="list-style-type: none"> <li>Bit 1 (xxxx xxx1): dft_mon_uow</li> <li>Bit 2 (xxxx xx1x): dft_mon_stmt</li> <li>Bit 3 (xxxx x1xx): dft_mon_table</li> <li>Bit 4 (xxxx 1xxx): dft_mon_buffpool</li> <li>Bit 5 (xxx1 xxxx): dft_mon_lock</li> <li>Bit 6 (xx1x xxxx): dft_mon_sort</li> <li>Bit 7 (x1xx xxxx): dft_mon_timestamp</li> </ul> <p>3. Valid values (defined in sqlutil.h):</p> <ul style="list-style-type: none"> <li>SQLF_DSCVR_KNOWN (1)</li> <li>SQLF_DSCVR_SEARCH (2)</li> </ul> <p>4. Valid values (defined in sqlutil.h):</p> <ul style="list-style-type: none"> <li>SQLF_INX_REC_SYSTEM (0)</li> <li>SQLF_INX_REC_REFERENCE (1)</li> </ul> <p>5. Valid values (defined in sqlutil.h):</p> <ul style="list-style-type: none"> <li>SQLF_TRUST_ALLCLNTS_NO (0)</li> <li>SQLF_TRUST_ALLCLNTS_YES (1)</li> <li>SQLF_TRUST_ALLCLNTS_DRDAONLY (2)</li> </ul> <p>6. Valid values (defined in sqlenv.h):</p> <ul style="list-style-type: none"> <li>SQL_ALTERNATE_AUTH_ENC_AES (0)</li> <li>SQL_ALTERNATE_AUTH_ENC_AES_CMP (1)</li> <li>SQL_ALTERNATE_AUTH_ENC_NOTSPEC (255)</li> </ul> |             |       |              |       |             |           |                        |

Table 36. Informational Database Manager Configuration Parameters

| Parameter                                                                                                                                                                                                                                                                                     | Token             | Token Value | Data Type | Additional Information                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-------------|-----------|----------------------------------------------------------|
| <b>nodetype<sup>1</sup></b>                                                                                                                                                                                                                                                                   | SQLF_KTN_NODETYPE | 100         | UInt16    | Chapter 17, "nodetype - Machine node type," on page 707  |
| <b>release</b>                                                                                                                                                                                                                                                                                | SQLF_KTN_RELEASE  | 101         | UInt16    | "release - Configuration file release level" on page 838 |
| <p><b>Note:</b></p> <p>1. Valid values (defined in sqlutil.h):</p> <ul style="list-style-type: none"> <li>SQLF_NT_STANDALONE (0)</li> <li>SQLF_NT_SERVER (1)</li> <li>SQLF_NT_REQUESTOR (2)</li> <li>SQLF_NT_STAND_REQ (3)</li> <li>SQLF_NT_MPP (4)</li> <li>SQLF_NT_SATELLITE (5)</li> </ul> |                   |             |           |                                                          |

## Database Configuration Parameter Summary

The following table lists the parameters in the database configuration file. When changing the database configuration parameters, consider the detailed information for the parameter.

For some database configuration parameters, changes only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) The changes take effect at the next connection to the database. Other parameters can be changed online; these are called *configurable online configuration parameters*.

Refer to the Database Manager Configuration Parameter Summary section above for a description of the “Auto.”, “Perf. Impact”, “Token”, “Token Value”, and “Data Type” columns.

The **AUTOMATIC** keyword is also supported on the UPDATE DB CFG command. In the following example, **database\_memory** will be updated to AUTOMATIC and the database manager will use 20000 as a starting value when making further changes to this parameter:

```
db2 update db cfg using for sample using database_memory 20000 automatic
```

Starting with Version 9.5, you can update and reset database configuration parameter values across some or all platforms without having to issue the db2\_all command, or without having to update or reset each partition individually. For details, see Configuring databases across multiple partitions.

Table 37. Configurable Database Configuration Parameters

| Parameter                                                                                                                                                                                                                                        | Cfg. Online | Auto. | Perf. Impact | Token                                                                                                                                                                                                                                                                                                                            | Token Value                                                                                                                                                      | Data Type | Additional Information                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|-------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|-------------------------------------------------------------------------------------------------|
| alt_collate                                                                                                                                                                                                                                      | No          | No    | None         | SQLF_DBTN_ALT_COLLATE                                                                                                                                                                                                                                                                                                            | 809                                                                                                                                                              | UInt32    | “alt_collate - Alternate collating sequence” on page 752                                        |
| applheapsz                                                                                                                                                                                                                                       | Yes         | Yes   | Medium       | SQLF_DBTN_APPLHEAPSZ                                                                                                                                                                                                                                                                                                             | 51                                                                                                                                                               | UInt16    | “applheapsz - Application heap size” on page 754                                                |
| appl_memory                                                                                                                                                                                                                                      | Yes         | Yes   | Medium       | SQLF_DBTN_APPL_MEMORY                                                                                                                                                                                                                                                                                                            | 904                                                                                                                                                              | UInt64    | “appl_memory - Application Memory configuration parameter” on page 753                          |
| archretrydelay                                                                                                                                                                                                                                   | Yes         | No    | None         | SQLF_DBTN_ARCHRETRYDELAY                                                                                                                                                                                                                                                                                                         | 828                                                                                                                                                              | UInt16    | “archretrydelay - Archive retry delay on error” on page 755                                     |
| <ul style="list-style-type: none"> <li>• auto_maint</li> <li>• auto_db_backup</li> <li>• auto_tbl_maint</li> <li>• auto_runstats</li> <li>• auto_stats_prof</li> <li>• auto_stmt_stats</li> <li>• auto_prof_upd</li> <li>• auto_reorg</li> </ul> | Yes         | No    | Medium       | <ul style="list-style-type: none"> <li>• SQLF_DBTN_AUTO_MAINT</li> <li>• SQLF_DBTN_AUTO_DB_BACKUP</li> <li>• SQLF_DBTN_AUTO_TBL_MAINT</li> <li>• SQLF_DBTN_AUTO_RUNSTATS</li> <li>• SQLF_DBTN_AUTO_STATS_PROF</li> <li>• SQLF_DBTN_AUTO_STMT_STATS</li> <li>• SQLF_DBTN_AUTO_PROF_UPD</li> <li>• SQLF_DBTN_AUTO_REORG</li> </ul> | <ul style="list-style-type: none"> <li>• 831</li> <li>• 833</li> <li>• 835</li> <li>• 837</li> <li>• 839</li> <li>• 905</li> <li>• 844</li> <li>• 841</li> </ul> | UInt16    | “auto_maint - Automatic maintenance” on page 757                                                |
| auto_del_rec_obj                                                                                                                                                                                                                                 | Yes         | No    | Medium       | SQLF_DBTN_AUTO_DEL_REC_OBJ                                                                                                                                                                                                                                                                                                       | 912                                                                                                                                                              | UInt16    | “auto_del_rec_obj - Automated deletion of recovery objects configuration parameter” on page 757 |
| autorestart                                                                                                                                                                                                                                      | Yes         | No    | Low          | SQLF_DBTN_AUTO_RESTART                                                                                                                                                                                                                                                                                                           | 25                                                                                                                                                               | UInt16    | Chapter 15, “autorestart - Auto restart enable,” on page 703                                    |
| auto_reval                                                                                                                                                                                                                                       | Yes         | No    | Medium       | SQLF_DBTN_AUTO_REVAL                                                                                                                                                                                                                                                                                                             | 920                                                                                                                                                              | UInt16    | auto_reval - Automatic revalidation and invalidation configuration parameter                    |
| avg_appls                                                                                                                                                                                                                                        | Yes         | Yes   | High         | SQLF_DBTN_AVG_APPLS                                                                                                                                                                                                                                                                                                              | 47                                                                                                                                                               | UInt16    | “avg_appls - Average number of active applications” on page 759                                 |
| blk_log_dsk_ful                                                                                                                                                                                                                                  | Yes         | No    | None         | SQLF_DBTN_BLK_LOG_DSK_FUL                                                                                                                                                                                                                                                                                                        | 804                                                                                                                                                              | UInt16    | “blk_log_dsk_ful - Block on log disk full” on page 760                                          |
| catalogcache_sz                                                                                                                                                                                                                                  | Yes         | No    | Medium       | SQLF_DBTN_CATALOGCACHE_SZ                                                                                                                                                                                                                                                                                                        | 56                                                                                                                                                               | UInt32    | “catalogcache_sz - Catalog cache size” on page 761                                              |
| chnpggs_thresh                                                                                                                                                                                                                                   | No          | No    | High         | SQLF_DBTN_CHNGPGS_THRESH                                                                                                                                                                                                                                                                                                         | 38                                                                                                                                                               | UInt16    | “chnpggs_thresh - Changed pages threshold” on page 762                                          |
| cur_commit                                                                                                                                                                                                                                       | No          | No    | Medium       | SQLF_DBTN_CUR_COMMIT                                                                                                                                                                                                                                                                                                             | 917                                                                                                                                                              | UInt32    | “cur_commit - Currently committed configuration parameter” on page 766                          |

Table 37. Configurable Database Configuration Parameters (continued)

| Parameter        | Cfg. Online | Auto. | Perf. Impact     | Token                      | Token Value                                                                    | Data Type | Additional Information                                                                  |
|------------------|-------------|-------|------------------|----------------------------|--------------------------------------------------------------------------------|-----------|-----------------------------------------------------------------------------------------|
| database_memory  | Yes         | Yes   | Medium           | SQLF_DBTN_DATABASE_MEMORY  | 803                                                                            | UInt64    | "database_memory - Database shared memory size" on page 768                             |
| dbheap           | Yes         | Yes   | Medium           | SQLF_DBTN_DB_HEAP          | 58                                                                             | UInt64    | "dbheap - Database heap" on page 771                                                    |
| db_mem_thresh    | Yes         | No    | Low              | SQLF_DBTN_DB_MEM_THRESH    | 849                                                                            | UInt16    | "db_mem_thresh - Database memory threshold" on page 771                                 |
| decflt_rounding  | No          | No    | None             | SQLF_DBTN_DECFLT_ROUNDING  | 913                                                                            | UInt16    | "decflt_rounding - Decimal floating point rounding configuration parameter" on page 773 |
| dec_to_char_fmt  | Yes         | Yes   | Medium           | SQLF_DBTN_DEC_TO_CHAR_FMT  | <ul style="list-style-type: none"> <li>• 0 (v95)</li> <li>• 1 (NEW)</li> </ul> | UInt16    | dec_to_char_fmt - Decimal to character function configuration parameter                 |
| dft_degree       | Yes         | No    | High             | SQLF_DBTN_DFT_DEGREE       | 301                                                                            | Sint32    | "dft_degree - Default degree" on page 775                                               |
| dft_extent_sz    | Yes         | No    | Medium           | SQLF_DBTN_DFT_EXTENT_SZ    | 54                                                                             | UInt32    | "dft_extent_sz - Default extent size of table spaces" on page 776                       |
| dft_loadrec_ses  | Yes         | No    | Medium           | SQLF_DBTN_DFT_LOADREC_SES  | 42                                                                             | Sint16    | "dft_loadrec_ses - Default number of load recovery sessions" on page 776                |
| dft_mttb_types   | No          | No    | None             | SQLF_DBTN_DFT_MTTB_TYPES   | 843                                                                            | UInt32    | "dft_mttb_types - Default maintained table types for optimization" on page 778          |
| dft_prefetch_sz  | Yes         | Yes   | Medium           | SQLF_DBTN_DFT_PREFETCH_SZ  | 40                                                                             | Sint16    | "dft_prefetch_sz - Default prefetch size" on page 778                                   |
| dft_queryopt     | Yes         | No    | Medium           | SQLF_DBTN_DFT_QUERYOPT     | 57                                                                             | Sint32    | "dft_queryopt - Default query optimization class" on page 779                           |
| dft_refresh_age  | No          | No    | Medium           | SQLF_DBTN_DFT_REFRESH_AGE  | 702                                                                            | char(22)  | "dft_refresh_age - Default refresh age" on page 780                                     |
| dft_sqlmathwarn  | No          | No    | None             | SQLF_DBTN_DFT_SQLMATHWARN  | 309                                                                            | Sint16    | "dft_sqlmathwarn - Continue upon arithmetic exceptions" on page 780                     |
| discover_db      | Yes         | No    | Medium           | SQLF_DBTN_DISCOVER         | 308                                                                            | UInt16    | "discover_db - Discover database" on page 786                                           |
| dlchktime        | Yes         | No    | Medium           | SQLF_DBTN_DLCHKTIME        | 9                                                                              | UInt32    | "dlchktime - Time interval for checking deadlock" on page 686                           |
| dyn_query_mgmt   | No          | No    | Low              | SQLF_DBTN_DYN_QUERY_MGMT   | 604                                                                            | UInt16    | "dyn_query_mgmt - Dynamic SQL and XQuery query management" on page 787                  |
| enable_xmlchar   | Yes         | No    | None             | SQLF_DBTN_ENABLE_XMLCHAR   | 853                                                                            | UInt32    | "enable_xmlchar - Enable conversion to XML configuration parameter" on page 787         |
| failarchpath     | Yes         | No    | None             | SQLF_DBTN_FAILARCHPATH     | 826                                                                            | char(243) | "failarchpath - Failover log archive path" on page 788                                  |
| hadr_local_host  | No          | No    | None             | SQLF_DBTN_HADR_LOCAL_HOST  | 811                                                                            | char(256) | "hadr_local_host - HADR local host name" on page 792                                    |
| hadr_local_svc   | No          | No    | None             | SQLF_DBTN_HADR_LOCAL_SVC   | 812                                                                            | char(41)  | "hadr_local_svc - HADR local service name" on page 792                                  |
| hadr_peer_window | No          | No    | Low (see Note 4) | SQLF_DBTN_HADR_PEER_WINDOW | 914                                                                            | UInt32    | "hadr_peer_window - HADR peer window configuration parameter" on page 793               |
| hadr_remote_host | No          | No    | None             | SQLF_DBTN_HADR_REMOTE_HOST | 813                                                                            | char(256) | "hadr_remote_host - HADR remote host name" on page 793                                  |
| hadr_remote_inst | No          | No    | None             | SQLF_DBTN_HADR_REMOTE_INST | 815                                                                            | char(9)   | "hadr_remote_inst - HADR instance name of the remote server" on page 794                |

Table 37. Configurable Database Configuration Parameters (continued)

| Parameter              | Cfg. Online | Auto. | Perf. Impact                    | Token                     | Token Value | Data Type | Additional Information                                                              |
|------------------------|-------------|-------|---------------------------------|---------------------------|-------------|-----------|-------------------------------------------------------------------------------------|
| hadr_remote_svc        | No          | No    | None                            | SQLF_DBTN_HADR_REMOTE_SVC | 814         | char(41)  | "hadr_remote_svc - HADR remote service name" on page 794                            |
| hadr_syncmode          | No          | No    | None                            | SQLF_DBTN_HADR_SYNCMODE   | 817         | UInt32    | "hadr_syncmode - HADR synchronization mode for log write in peer state" on page 794 |
| hadr_timeout           | No          | No    | None                            | SQLF_DBTN_HADR_TIMEOUT    | 816         | UInt32    | "hadr_timeout - HADR timeout value" on page 795                                     |
| indexrec <sup>2</sup>  | Yes         | No    | Medium                          | SQLF_DBTN_INDEXREC        | 30          | UInt16    | "indexrec - Index re-creation time" on page 796                                     |
| locklist               | Yes         | Yes   | High when it affects escalation | SQLF_DBTN_LOCK_LIST       | 704         | UInt64    | "locklist - Maximum storage for lock list" on page 686                              |
| locktimeout            | No          | No    | Medium                          | SQLF_DBTN_LOCKTIMEOUT     | 34          | Sint16    | "locktimeout - Lock timeout" on page 689                                            |
| logarchmeth1           | Yes         | No    | None                            | SQLF_DBTN_LOGARCHMETH1    | 822         | char(252) | "logarchmeth1 - Primary log archive method" on page 804                             |
| logarchmeth2           | Yes         | No    | None                            | SQLF_DBTN_LOGARCHMETH2    | 823         | char(252) | "logarchmeth2 - Secondary log archive method" on page 806                           |
| logarchopt1            | Yes         | No    | None                            | SQLF_DBTN_LOGARCHOPT1     | 824         | char(243) | "logarchopt1 - Primary log archive options" on page 807                             |
| logarchopt2            | Yes         | No    | None                            | SQLF_DBTN_LOGARCHOPT2     | 825         | char(243) | "logarchopt2 - Secondary log archive options" on page 807                           |
| logbufsz               | No          | No    | High                            | SQLF_DBTN_LOGBUFSZ        | 33          | UInt16    | "logbufsz - Log buffer size" on page 808                                            |
| logfilsiz              | No          | No    | Medium                          | SQLF_DBTN_LOGFIL_SIZ      | 92          | UInt32    | "logfilsiz - Size of log files" on page 808                                         |
| logindexbuild          | Yes         | No    | None                            | SQLF_DBTN_LOGINDEXBUILD   | 818         | UInt32    | "logindexbuild - Log index pages created" on page 809                               |
| logprimary             | No          | No    | Medium                          | SQLF_DBTN_LOGPRIMARY      | 16          | UInt16    | "logprimary - Number of primary log files" on page 810                              |
| logretain <sup>3</sup> | No          | No    | Low                             | SQLF_DBTN_LOG_RETAIN      | 23          | UInt16    | "logretain - Log retain enable" on page 811                                         |
| logsecond              | Yes         | No    | Medium                          | SQLF_DBTN_LOGSECOND       | 17          | UInt16    | "logsecond - Number of secondary log files" on page 812                             |
| max_log                | Yes         | Yes   |                                 | SQLF_DBTN_MAX_LOG         | 807         | UInt16    | "max_log - Maximum log per transaction" on page 815                                 |
| maxappls               | Yes         | Yes   | Medium                          | SQLF_DBTN_MAXAPPLS        | 6           | UInt16    | "maxappls - Maximum number of active applications" on page 816                      |
| maxfilop               | Yes         | No    | Medium                          | SQLF_DBTN_MAXFILOP        | 3           | UInt16    | "maxfilop - Maximum database files open per application" on page 817                |
| maxlocks               | Yes         | Yes   | High when it affects escalation | SQLF_DBTN_MAXLOCKS        | 15          | UInt16    | "maxlocks - Maximum percent of lock list before escalation" on page 690             |
| min_dec_div_3          | No          | No    | High                            | SQLF_DBTN_MIN_DEC_DIV_3   | 605         | Sint32    | "min_dec_div_3 - Decimal division scale to 3" on page 818                           |
| mincommit              | Yes         | No    | High                            | SQLF_DBTN_MINCOMMIT       | 32          | UInt16    | "mincommit - Number of commits to group" on page 819                                |
| mirrorlogpath          | No          | No    | Low                             | SQLF_DBTN_MIRRORLOGPATH   | 806         | char(242) | "mirrorlogpath - Mirror log path" on page 820                                       |
| mon_act_metrics        | Yes         | No    | Medium                          | SQLF_DBTN_MON_ACT_METRICS | 931         | UInt16    | com.ibm.db2.luw.admin.config.doc/doc/r0054936.dita#r                                |
| mon_deadlock           | Yes         | No    | Medium                          | SQLF_DBTN_MON_DEADLOCK    | 934         | UInt16    | com.ibm.db2.luw.admin.config.doc/doc/r0054940.dita#r                                |
| mon_locktimeout        | Yes         | No    | Medium                          | SQLF_DBTN_MON_LOCKTIMEOUT | 933         | UInt16    | com.ibm.db2.luw.admin.config.doc/doc/r0054939.dita#r                                |

Table 37. Configurable Database Configuration Parameters (continued)

| Parameter       | Cfg. Online | Auto. | Perf. Impact | Token                     | Token Value | Data Type | Additional Information                                                |
|-----------------|-------------|-------|--------------|---------------------------|-------------|-----------|-----------------------------------------------------------------------|
| mon_lockwait    | Yes         | No    | Medium       | SQLF_DBTN_MON_LOCKWAIT    | 935         | UInt16    | com.ibm.db2.luw.admin.config.doc/doc/r0054941.di                      |
| mon_lw_thresh   | Yes         | No    | Medium       | SQLF_DBTN_MON_LW_THRESH   | 936         | UInt32    | com.ibm.db2.luw.admin.config.doc/doc/r0054942.di                      |
| mon_obj_metrics | Yes         | No    | Medium       | SQLF_DBTN_MON_OBJ_METRICS | 937         | UInt16    | com.ibm.db2.luw.admin.config.doc/doc/r0054937.di                      |
| mon_req_metrics | Yes         | No    | Medium       | SQLF_DBTN_MON_REQ_METRICS | 930         | UInt16    | com.ibm.db2.luw.admin.config.doc/doc/r0054934.di                      |
| mon_uow_data    | Yes         | No    | Medium       | SQLF_DBTN_MON_UOW_DATA    | 932         | UInt16    | com.ibm.db2.luw.admin.config.doc/doc/r0054938.di                      |
| newlogpath      | No          | No    | Low          | SQLF_DBTN_NEWLOGPATH      | 20          | char(242) | "newlogpath - Change the database log path" on page 822               |
| num_db_backups  | Yes         | No    | None         | SQLF_DBTN_NUM_DB_BACKUPS  | 601         | UInt16    | "num_db_backups - Number of database backups" on page 825             |
| num_freqvalues  | Yes         | No    | Low          | SQLF_DBTN_NUM_FREQVALUES  | 36          | UInt16    | "num_freqvalues - Number of frequent values retained" on page 825     |
| num_iocleaners  | No          | Yes   | High         | SQLF_DBTN_NUM_IOCLEANERS  | 37          | UInt16    | "num_iocleaners - Number of asynchronous page cleaners" on page 827   |
| num_ioservers   | No          | Yes   | High         | SQLF_DBTN_NUM_IOSERVERS   | 39          | UInt16    | "num_ioservers - Number of I/O servers" on page 829                   |
| num_log_span    | Yes         | Yes   |              | SQLF_DBTN_NUM_LOG_SPAN    | 808         | UInt16    | "num_log_span - Number log span" on page 830                          |
| num_quantiles   | Yes         | No    | Low          | SQLF_DBTN_NUM_QUANTILES   | 48          | UInt16    | "num_quantiles - Number of quantiles for columns" on page 831         |
| numarchretry    | Yes         | No    | None         | SQLF_DBTN_NUMARCHRETRY    | 827         | UInt16    | "numarchretry - Number of retries on error" on page 832               |
| overflowlogpath | No          | No    | Medium       | SQLF_DBTN_OVERFLOWLOGPATH | 805         | char(242) | "overflowlogpath - Overflow log path" on page 834                     |
| pckcachesz      | Yes         | Yes   | High         | SQLF_DBTN_PCKCACHE_SZ     | 505         | UInt32    | "pckcachesz - Package cache size" on page 835                         |
| rec_his_retentn | No          | No    | None         | SQLF_DBTN_REC_HIS_RETENTN | 43          | Sint16    | "rec_his_retentn - Recovery history retention period" on page 838     |
| self_tuning_mem | Yes         | No    | High         | SQLF_DBTN_SELF_TUNING_MEM | 848         | UInt16    | "self_tuning_mem- Self-tuning memory" on page 843                     |
| seqdetect       | Yes         | No    | High         | SQLF_DBTN_SEQDETECT       | 41          | UInt16    | "seqdetect - Sequential detection flag" on page 845                   |
| sheapthres_shr  | Yes         | Yes   | High         | SQLF_DBTN_SHEAPTHRES_SHR  | 802         | UInt32    | "sheapthres_shr - Sort heap threshold for shared sorts" on page 847   |
| softmax         | No          | No    | Medium       | SQLF_DBTN_SOFTMAX         | 5           | UInt16    | "softmax - Recovery range and soft checkpoint interval" on page 849   |
| sortheap        | Yes         | Yes   | High         | SQLF_DBTN_SORT_HEAP       | 52          | UInt32    | "sortheap - Sort heap size" on page 850                               |
| stat_heap_sz    | Yes         | Yes   | Low          | SQLF_DBTN_STAT_HEAP_SZ    | 45          | UInt32    | "stat_heap_sz - Statistics heap size" on page 853                     |
| stmt_conc       | Yes         | No    | Medium       | SQLF_DBTN_STMT_CONC       | 919         | UInt32    | stmt_conc - Statement concentrator configuration parameter            |
| stmtheap        | Yes         | Yes   | Medium       | SQLF_DBTN_STMT_HEAP       | 821         | UInt32    | "stmtheap - Statement heap size" on page 854                          |
| trackmod        | No          | No    | Low          | SQLF_DBTN_TRACKMOD        | 703         | UInt16    | "trackmod - Track modified pages enable" on page 858                  |
| tsm_mgmtclass   | Yes         | No    | None         | SQLF_DBTN_TSM_MGMTCLASS   | 307         | char(30)  | "tsm_mgmtclass - Tivoli Storage Manager management class" on page 859 |
| tsm_nodename    | Yes         | No    | None         | SQLF_DBTN_TSM_NODENAME    | 306         | char(64)  | "tsm_nodename - Tivoli Storage Manager node name" on page 859         |



Table 37. Configurable Database Configuration Parameters (continued)

| Parameter       | Cfg. Online | Auto. | Perf. Impact | Token                     | Token Value | Data Type | Additional Information                                                                          |
|-----------------|-------------|-------|--------------|---------------------------|-------------|-----------|-------------------------------------------------------------------------------------------------|
| tsm_owner       | Yes         | No    | None         | SQLF_DBTN_TSM_OWNER       | 305         | char(64)  | "tsm_owner - Tivoli Storage Manager owner name" on page 860                                     |
| tsm_password    | Yes         | No    | None         | SQLF_DBTN_TSM_PASSWORD    | 501         | char(64)  | "tsm_password - Tivoli Storage Manager password" on page 860                                    |
| userexit        | No          | No    | Low          | SQLF_DBTN_USER_EXIT       | 24          | UInt16    | "userexit - User exit enable" on page 861                                                       |
| util_heap_sz    | Yes         | No    | Low          | SQLF_DBTN_UTIL_HEAP_SZ    | 55          | UInt32    | "util_heap_sz - Utility heap size" on page 861                                                  |
| vendoropt       | Yes         | No    | None         | SQLF_DBTN_VENDOROPT       | 829         | char(242) | "vendoropt - Vendor options" on page 863<                                                       |
| wlm_collect_int | Yes         | No    | Low          | SQLF_DBTN_WLM_COLLECT_INT | 907         | Sint32    | "wlm_collect_int - Workload management collection interval configuration parameter" on page 863 |

Note: The bits of SQLF\_DBTN\_AUTONOMIC\_SWITCHES indicate the default settings for a number of auto-maintenance configuration parameters. The individual bits making up this composite parameter are:

1.

```
Default => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint
Bit 2 off (xxxx xxxx xxxx xx0x): auto_db_backup
Bit 3 on (xxxx xxxx xxxx x1xx): auto_tbl_maint
Bit 4 on (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx0 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx xx0x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x0xx xxxx): auto_reorg
Bit 8 off (xxxx xxxx 0xxx xxxx): auto_storage
Bit 9 off (xxxx xxx0 xxxx xxxx): auto_stmt_stats
0 0 0 0
```

```
Maximum => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint
Bit 2 off (xxxx xxxx xxxx xx1x): auto_db_backup
Bit 3 on (xxxx xxxx xxxx x1xx): auto_tbl_maint
Bit 4 on (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx x1x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x1xx xxxx): auto_reorg
Bit 8 off (xxxx xxxx 1xxx xxxx): auto_storage
Bit 9 off (xxxx xxx1 xxxx xxxx): auto_stmt_stats
0 1 F F
```

2. Valid values (defined in sqlutil.h):

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
SQLF_INX_REC_RESTART (2)
```

3. Valid values (defined in sqlutil.h):

```
SQLF_LOGRETAIN_NO (0)
SQLF_LOGRETAIN_RECOVERY (1)
SQLF_LOGRETAIN_CAPTURE (2)
```

4. If you set the **hadr\_peer\_window** parameter to a non-zero time value, then the primary database might seem to hang on transactions when it is in disconnected peer state, because it is waiting for confirmation from the standby database even though it is not connected to the standby database.

Table 38. Informational Database Configuration Parameters

| Parameter      | Token                    | Token Value | Data Type            | Additional Information                                  |
|----------------|--------------------------|-------------|----------------------|---------------------------------------------------------|
| backup_pending | SQLF_DBTN_BACKUP_PENDING | 112         | UInt16               | "backup_pending - Backup pending indicator" on page 760 |
| codepage       | SQLF_DBTN_CODEPAGE       | 101         | UInt16               | "codepage - Code page for the database" on page 763     |
| codeset        | SQLF_DBTN_CODESET        | 120         | char(9) <sup>1</sup> | "codeset - Codeset for the database" on page 763        |
| collate_info   | SQLF_DBTN_COLLATE_INFO   | 44          | char(260)            | "collate_info - Collating information" on page 764      |
| country/region | SQLF_DBTN_COUNTRY        | 100         | UInt16               | "country/region - Database territory code" on page 766  |

Table 38. Informational Database Configuration Parameters (continued)

| Parameter                                                                                                                                                                                           | Token                       | Token Value | Data Type            | Additional Information                                                                              |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|-------------|----------------------|-----------------------------------------------------------------------------------------------------|
| database_consistent                                                                                                                                                                                 | SQLF_DBTN_CONSISTENT        | 111         | Uint16               | Chapter 16, "database_consistent - Database is consistent," on page 705                             |
| database_level                                                                                                                                                                                      | SQLF_DBTN_DATABASE_LEVEL    | 124         | Uint16               | "database_level - Database release level" on page 768                                               |
| hadr_db_role                                                                                                                                                                                        | SQLF_DBTN_HADR_DB_ROLE      | 810         | Uint32               | "hadr_db_role - HADR database role" on page 791                                                     |
| log_retain_status                                                                                                                                                                                   | SQLF_DBTN_LOG_RETAIN_STATUS | 114         | Uint16               | "log_retain_status - Log retain status indicator" on page 804                                       |
| loghead                                                                                                                                                                                             | SQLF_DBTN_LOGHEAD           | 105         | char(12)             | "loghead - First active log file" on page 809                                                       |
| logpath                                                                                                                                                                                             | SQLF_DBTN_LOGPATH           | 103         | char(242)            | "logpath - Location of log files" on page 810                                                       |
| multipage_alloc                                                                                                                                                                                     | SQLF_DBTN_MULTIPAGE_ALLOC   | 506         | Uint16               | "multipage_alloc - Multipage file allocation enabled" on page 822                                   |
| numsegs                                                                                                                                                                                             | SQLF_DBTN_NUMSEGS           | 122         | Uint16               | "numsegs - Default number of SMS containers" on page 833                                            |
| pagesize                                                                                                                                                                                            | SQLF_DBTN_PAGESIZE          | 846         | Uint32               | "pagesize - Database default page size" on page 835                                                 |
| release                                                                                                                                                                                             | SQLF_DBTN_RELEASE           | 102         | Uint16               | "release - Configuration file release level" on page 838                                            |
| restore_pending                                                                                                                                                                                     | SQLF_DBTN_RESTORE_PENDING   | 503         | Uint16               | "restore_pending - Restore pending" on page 839                                                     |
| restrict_access                                                                                                                                                                                     | SQLF_DBTN_RESTRICT_ACCESS   | 852         | Sint32               | Chapter 18, "restrict_access - Database has restricted access configuration parameter," on page 709 |
| rollfwd_pending                                                                                                                                                                                     | SQLF_DBTN_ROLLFWD_PENDING   | 113         | Uint16               | "rollfwd_pending - Roll forward pending indicator" on page 841                                      |
| territory                                                                                                                                                                                           | SQLF_DBTN_TERRITORY         | 121         | char(5) <sup>2</sup> | "territory - Database territory" on page 854                                                        |
| user_exit_status                                                                                                                                                                                    | SQLF_DBTN_USER_EXIT_STATUS  | 115         | Uint16               | "user_exit_status - User exit status indicator" on page 861                                         |
| <p>Note:</p> <ol style="list-style-type: none"> <li>1. char(17) on HP-UX, Linux and the Solaris operating system.</li> <li>2. char(33) on HP-UX, Linux and the Solaris operating system.</li> </ol> |                             |             |                      |                                                                                                     |

## DB2 Administration Server (DAS) Configuration Parameter Summary

Table 39. DAS Configuration Parameters

| Parameter      | Parameter Type      | Additional Information                                                                     |
|----------------|---------------------|--------------------------------------------------------------------------------------------|
| authentication | Configurable        | "authentication - Authentication type DAS" on page 679                                     |
| contact_host   | Configurable Online | "contact_host - Location of contact list" on page 765                                      |
| das_codepage   | Configurable Online | "das_codepage - DAS code page" on page 767                                                 |
| das_territory  | Configurable Online | "das_territory - DAS territory" on page 768                                                |
| dasadm_group   | Configurable        | "dasadm_group - DAS administration authority group name" on page 680                       |
| db2system      | Configurable Online | "db2system - Name of the DB2 server system" on page 770                                    |
| discover       | Configurable Online | "discover - DAS discovery mode" on page 785                                                |
| exec_exp_task  | Configurable        | "exec_exp_task - Execute expired tasks" on page 788                                        |
| jdk_64_path    | Configurable Online | "jdk_64_path - 64-Bit Software Developer's Kit for Java installation path DAS" on page 802 |
| jdk_path       | Configurable Online | "jdk_path - Software Developer's Kit for Java installation path DAS" on page 802           |
| sched_enable   | Configurable        | "sched_enable - Scheduler mode" on page 843                                                |
| sched_userid   | Informational       | "sched_userid - Scheduler user ID" on page 843                                             |
| smtp_server    | Configurable Online | "smtp_server - SMTP server" on page 848                                                    |

Table 39. DAS Configuration Parameters (continued)

| Parameter       | Parameter Type | Additional Information                                        |
|-----------------|----------------|---------------------------------------------------------------|
| toolscat_db     | Configurable   | "toolscat_db - Tools catalog database" on page 855            |
| toolscat_inst   | Configurable   | "toolscat_inst - Tools catalog database instance" on page 856 |
| toolscat_schema | Configurable   | "toolscat_schema - Tools catalog database schema" on page 856 |

## Configuration parameter section headings

Each of the configuration parameter descriptions contain some or all of the following section headings, as applicable. In some cases they are mutually exclusive, for example, valid values are not needed if the [range] is specified. In most cases, these headings are self-explanatory.

Table 40.

| Section heading    | Description and possible values                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Configuration type | Possible values are: <ul style="list-style-type: none"> <li>• Database manager</li> <li>• Database</li> <li>• DB2 Administration Server</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Applies to         | If applicable, lists the data server types that the configuration parameter applies to. Possible values are: <ul style="list-style-type: none"> <li>• Client</li> <li>• Database server with local and remote clients</li> <li>• Database server with local clients</li> <li>• DB2 Administration Server</li> <li>• OLAP functions</li> <li>• Partitioned database server with local and remote clients</li> <li>• Partitioned database server with local and remote clients when federation is enabled.</li> <li>• Satellite database server with local clients</li> </ul> |
| Parameter type     | Possible values are: <ul style="list-style-type: none"> <li>• Configurable (the database manager must be restarted to have the changes take effect)</li> <li>• Configurable online (can be dynamically updated online without having to restart the database manager)</li> <li>• Informational (values are for your information only and cannot be updated)</li> </ul>                                                                                                                                                                                                      |
| Default [range]    | If applicable, lists the default value and the possible ranges, including NULL values or automatic settings. If the range differs by platform, then the values are listed by platform or platform type, for example, 32-bit or 64-bit platforms. Note that in most cases the default value is not listed as part of the range.                                                                                                                                                                                                                                              |
| Unit of measure    | If applicable, lists the unit of measure. Possible values are: <ul style="list-style-type: none"> <li>• Bytes</li> <li>• Counter</li> <li>• Megabytes per second</li> <li>• Milliseconds</li> <li>• Minutes</li> <li>• Pages (4 KB)</li> <li>• Percentage</li> <li>• Seconds</li> </ul>                                                                                                                                                                                                                                                                                     |
| Valid values       | If applicable, lists the valid value. This heading is mutually exclusive with the default [range] heading.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Examples           | If applicable, lists examples.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Propagation class  | If applicable, possible values are: <ul style="list-style-type: none"> <li>• Immediate</li> <li>• Statement boundary</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| When allocated     | If applicable, indicates when the configuration parameter is allocated by the database manager.                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

Table 40. (continued)

| Section heading | Description and possible values                                                             |
|-----------------|---------------------------------------------------------------------------------------------|
| When freed      | If applicable, indicates when the configuration parameter is freed by the database manager. |
| Restrictions    | If applicable, lists any restrictions that apply to the configuration parameter.            |
| Limitations     | If applicable, lists any limitations that apply to the configuration parameter.             |
| Recommendations | If applicable, lists any recommendations that apply to the configuration parameter.         |
| Usage notes     | If applicable, lists any usage notes that apply to the configuration parameter.             |

---

## Changing the database configuration across multiple database partitions

When you have a database that is distributed across more than one database partition, the database configuration file should be the same on all database partitions.

Consistency is required since the SQL compiler compiles distributed SQL statements based on information in the node configuration file and creates an access plan to satisfy the needs of the SQL statement. Maintaining different configuration files on database partitions could lead to different access plans, depending on which database partition the statement is prepared. Use `db2_all` to maintain the configuration files across all database partitions.

---

## Security-Related Configuration Parameters

### **audit\_buf\_sz - Audit buffer size**

This parameter specifies the size of the buffer used when auditing the database.

#### **Configuration type**

Database manager

#### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

#### **Parameter type**

Configurable

#### **Default [range]**

0 [0 - 65 000 ]

#### **Unit of measure**

Pages (4 KB)

#### **When allocated**

When DB2 is started

#### **When freed**

When DB2 is stopped

The default value for this parameter is zero (0). If the value is zero (0), the audit buffer is not used. If the value is greater than zero (0), space is allocated for the audit buffer where the audit records will be placed when they are generated by the audit facility. The value times 4 KB pages is the amount of space allocated for the

audit buffer. The audit buffer cannot be allocated dynamically; DB2 must be stopped and then restarted before the new value for this parameter takes effect.

By changing this parameter from the default to some value larger than zero (0), the audit facility writes records to disk asynchronously compared to the execution of the statements generating the audit records. This improves DB2 performance over leaving the parameter value at zero (0). The value of zero (0) means the audit facility writes records to disk synchronously with (at the same time as) the execution of the statements generating the audit records. The synchronous operation during auditing decreases the performance of applications running in DB2.

## authentication - Authentication type

This parameter specifies and determines how and where authentication of a user takes place.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default [range]

SERVER [CLIENT; SERVER; SERVER\_ENCRYPT; DATA\_ENCRYPT;  
DATA\_ENCRYPT\_CMP; KERBEROS; KRB\_SERVER\_ENCRYPT;  
GSSPLUGIN; GSS\_SERVER\_ENCRYPT ]

If **authentication** is SERVER, the user ID and password are sent from the client to the server so that authentication can take place on the server. The value SERVER\_ENCRYPT provides the same behavior as SERVER, except that any user IDs and passwords sent over the network are encrypted.

A value of DATA\_ENCRYPT means the server accepts encrypted SERVER authentication schemes and the encryption of user data. The authentication works exactly the same way as SERVER\_ENCRYPT.

The following user data are encrypted when using this authentication type:

- SQL statements
- SQL program variable data
- Output data from the server processing an SQL statement and including a description of the data
- Some or all of the answer set data resulting from a query
- Large object (LOB) streaming
- SQLDA descriptors

A value of DATA\_ENCRYPT\_CMP means the server accepts encrypted SERVER authentication schemes and the encryption of user data. In addition, this authentication type allows compatibility with earlier products that do not support DATA\_ENCRYPT authentication type. These products are permitted to connect

with the `SERVER_ENCRYPT` authentication type and without encrypting user data. Products supporting the new authentication type must use it. This authentication type is only valid in the server's database manager configuration file and is not valid when used on the `CATALOG DATABASE` command.

**Note:** For a standards compliance (defined in the "Standards compliance" topic) configuration, `SERVER` is the only supported value.

A value of `CLIENT` indicates that all authentication takes place at the client. No authentication needs to be performed at the server.

A value of `KERBEROS` means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication. With an authentication type of `KRB_SERVER_ENCRYPT` at the server and clients that support the Kerberos security system, the effective system authentication type is `KERBEROS`. If the clients do not support the Kerberos security system, the system authentication type is effectively equivalent to `SERVER_ENCRYPT`.

A value of `GSSPLUGIN` means that authentication is performed using an external GSSAPI-based security mechanism. With an authentication type of `GSS_SERVER_ENCRYPT` at the server and clients that support the `GSSPLUGIN` security mechanism, the effective system authentication type is `GSSPLUGIN` (that is, if the clients support one of the server's plug-ins). If the clients do not support the `GSSPLUGIN` security mechanism, the system authentication type is effectively equivalent to `SERVER_ENCRYPT`.

**Recommendation:** Typically, the default value (`SERVER`) is adequate for local clients. If remote clients are connecting to the database server then `SERVER_ENCRYPT` is the suggested value to protect the user ID and password.

## authentication - Authentication type DAS

This parameter determines how and where authentication of a user takes place.

### Configuration type

DB2 Administration Server

### Applies to

DB2 Administration Server

### Parameter type

Configurable

### Default [range]

`SERVER_ENCRYPT [SERVER_ENCRYPT; KERBEROS_ENCRYPT ]`

If **authentication** is `SERVER_ENCRYPT`, then the user ID and password are sent from the client to the server so authentication can take place on the server. User IDs and passwords sent over the network are encrypted.

A value of `KERBEROS_ENCRYPT` means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication.

**Note:** The `KERBEROS_ENCRYPT` authentication type is only supported on servers running Windows.

This parameter can only be updated from a Version 9 command line processor (CLP).

## catalog\_noauth - Cataloging allowed without authority

This parameter specifies whether users are able to catalog and uncatalog databases and nodes, or DCS and ODBC directories, without SYSADM authority.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable Online

### Propagation class

Immediate

### Default [range]

#### Database server with local and remote clients

NO [ NO (0) — YES (1) ]

#### Client; Database server with local clients

YES [ NO (0) — YES (1) ]

The default value (0) for this parameter indicates that SYSADM authority is required. When this parameter is set to 1 (yes), SYSADM authority is not required.

## dasadm\_group - DAS administration authority group name

This parameter defines the group name with DAS Administration (DASADM) authority for the DAS.

### Configuration type

DB2 Administration Server

### Applies to

DB2 Administration Server

### Parameter type

Configurable

### Default [range]

Null [any valid group name ]

DASADM authority is the highest level of authority within the DAS.

DASADM authority is determined by the security facilities used in a specific operating environment.

- For the Windows operating systems, this parameter can be set to any local group that is defined in the Windows security database. Group names are accepted as long as they are 30 bytes or less in length. If “NULL” is specified for this parameter, all members of the Administrators group have DASADM authority.
- For Linux and UNIX systems, if “NULL” is specified as the value of this parameter, the DASADM group defaults to the primary group of the instance owner.

If the value is not “NULL”, the DASADM group can be any valid UNIX group name.



This parameter can only be updated from a Version 8 command line processor (CLP).

## **dftdbpath - Default database path**

This parameter contains the default file path used to create databases under the database manager. If no path is specified when a database is created, the database is created under the path specified by the *dftdbpath* parameter.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable Online

### **Propagation class**

Immediate

### **Default [range]**

UNIX Home directory of instance owner [any existing path ]

### **Windows**

Drive on which DB2 is installed [any existing path ]

In a partitioned database environment, you should ensure that the path on which the database is being created is not an NFS-mounted path (on Linux and UNIX platforms), or a network drive (in a Windows environment). The specified path must physically exist on each database partition server. To avoid confusion, it is best to specify a path that is locally mounted on each database partition server. The maximum length of the path is 205 characters. The system appends the database partition name to the end of the path.

Given that databases can grow to a large size and that many users could be creating databases (depending on your environment and intentions), it is often convenient to be able to have all databases created and stored in a specified location. It is also good to be able to isolate databases from other applications and data both for integrity reasons and for ease of backup and recovery.

For Linux and UNIX environments, the length of the *dftdbpath* name cannot exceed 215 characters and must be a valid, absolute, path name. For Windows, the *dftdbpath* can be a drive letter, optionally followed by a colon.

**Recommendation:** If possible, put high volume databases on a different disk than other frequently accessed data, such as the operating system files and the database logs.

## **svcname - TCP/IP service name**

This parameter contains the name of the TCP/IP port which a database server will use to await communications from remote client nodes. This name must be the reserved for use by the database manager.

### **Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default**

Null

In order to accept connection requests from a Data Server Runtime Client using TCP/IP, the database server must be listening on a port designated to that server. The system administrator for the database server must reserve a port (number *n*) and define its associated TCP/IP service name in the services file at the server.

The database server port (number *n*) and its TCP/IP service name need to be defined in the services file on the database client.

On Linux and UNIX systems, the services file is located in: `/etc/services`

The *svcname* parameter should be set to the service name associated with the main connection port so that when the database server is started, it can determine on which port to listen for incoming connection requests.

## **sysadm\_group - System administration authority group name**

This parameter defines the group name with SYSADM authority for the database manager instance.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default**

NULL

The SYSADM authority level is the highest level of administrative authority at the instance level. Users with SYSADM authority can run some utilities and issue some database and database manager commands within the instance.

SYSADM authority is determined by the security facilities used in a specific operating environment.

- For the Windows operating system, this parameter can be set to local or domain group. Group names must adhere to the length limits specified in SQL and XML limits. The following users have SYSADM authority if "NULL" is specified for **sysadm\_group** database manager configuration parameter:
  - Members of the local Administrators group

- Members of the Administrators group at the Domain Controller if DB2\_GRP\_LOOKUP is not set or set to DOMAIN
- Members of DB2ADMNS group if Extended Security feature is enabled. The location of the DB2ADMNS group was decided during installation
- The LocalSystem account
- For Linux and UNIX systems, if “NULL” is specified as the value of this parameter, the SYSADM group defaults to the primary group of the instance owner.  
If the value is not “NULL”, the SYSADM group can be any valid UNIX group name.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSADM\_GROUP NULL. You must specify the keyword “NULL” in uppercase.

## sysctrl\_group - System control authority group name

This parameter defines the group name with system control (SYSCTRL) authority. SYSCTRL has privileges allowing operations affecting system resources, but does not allow direct access to data.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default

Null

Group names on all platforms are accepted as long as they adhere to the length limits specified in SQL and XML limits.

**Attention:** This parameter must be NULL for Windows clients when system security is used (that is, **authentication** is CLIENT, SERVER or any other valid authentication). This is because the Windows operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSCTRL group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSCTRL\_GROUP NULL. You must specify the keyword NULL in uppercase.

## sysmaint\_group - System maintenance authority group name

This parameter defines the group name with system maintenance (SYSMAINT) authority.

### Configuration type

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default**

Null

SYSMAINT has privileges to perform maintenance operations on all databases associated with an instance without having direct access to data.

Group names on all platforms are accepted as long as they adhere to the length limits specified in SQL and XML limits.

**Attention:** This parameter must be NULL for Windows clients when system security is used (that is, **authentication** is CLIENT, SERVER, or any other valid authentication). This is because the Windows operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSMAINT group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMAINT\_GROUP NULL. You must specify the keyword NULL in uppercase.

## **sysmon\_group - System monitor authority group name**

This parameter defines the group name with system monitor (SYSMON) authority.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default**

Null

Users having SYSMON authority at the instance level have the ability to take database system monitor snapshots of a database manager instance or its databases. SYSMON authority includes the ability to use the following commands:

- GET DATABASE MANAGER MONITOR SWITCHES
- GET MONITOR SWITCHES
- GET SNAPSHOT
- LIST ACTIVE DATABASES
- LIST APPLICATIONS

- LIST DCS APPLICATIONS
- RESET MONITOR
- UPDATE MONITOR SWITCHES

Users with SYSADM, SYSCTRL, or SYSMAINT authority automatically have the ability to take database system monitor snapshots and to use these commands.

Group names on all platforms are accepted as long as they adhere to the length limits specified in SQL and XML limits.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMON\_GROUP NULL. You must specify the keyword NULL in uppercase.

## **trust\_allclnts - Trust all clients**

This parameter and *trust\_clntauth* are used to determine where users are validated to the database environment.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable

### **Default [range]**

YES [NO, YES, DRDAONLY]

This parameter is only active when the *authentication* parameter is set to CLIENT.

By accepting the default of “YES” for this parameter, all clients are treated as trusted clients. This means that the server assumes that a level of security is available at the client and the possibility that users can be validated at the client.

This parameter can only be changed to “NO” if the *authentication* parameter is set to CLIENT. If this parameter is set to “NO”, the untrusted clients must provide a userid and password combination when they connect to the server. Untrusted clients are operating system platforms that do not have a security subsystem for authenticating users.

Setting this parameter to “DRDAONLY” protects against all clients except clients from DB2 for OS/390® and z/OS, DB2 for VM and VSE, and DB2 for OS/400®. Only these clients can be trusted to perform client-side authentication. All other clients must provide a user ID and password to be authenticated by the server.

When *trust\_allclnts* is set to “DRDAONLY”, the *trust\_clntauth* parameter is used to determine where the clients are authenticated. If *trust\_clntauth* is set to “CLIENT”, authentication occurs at the client. If *trust\_clntauth* is set to “SERVER”, authentication occurs at the client if no password is provided, and at the server if a password is provided.

---

## Locking Configuration Parameters

### dlchktime - Time interval for checking deadlock

This parameter defines the frequency at which the database manager checks for deadlocks among all the applications connected to a database.

**Configuration type**

Database

**Parameter type**

Configurable online

**Propagation class**

Immediate

**Default [range]**

10 000 (10 seconds) [1 000 - 600 000]

**Unit of measure**

Milliseconds

A deadlock occurs when two or more applications connected to the same database wait indefinitely for a resource. The waiting is never resolved because each application is holding a resource that the other needs to continue.

**Note:**

1. In a partitioned database environment, this parameter applies to the catalog node only.
2. In a partitioned database environment, a deadlock is not flagged until after the second iteration.

**Recommendation:** Increasing this parameter decreases the frequency of checking for deadlocks, thereby increasing the time that application programs must wait for the deadlock to be resolved.

Decreasing this parameter increases the frequency of checking for deadlocks, thereby decreasing the time that application programs must wait for the deadlock to be resolved but increasing the time that the database manager takes to check for deadlocks. If the deadlock interval is too small, it can decrease runtime performance, because the database manager is frequently performing deadlock detection. If this parameter is set lower to improve concurrency, you should ensure that *maxlocks* and *locklist* are set appropriately to avoid unnecessary lock escalation, which can result in more lock contention and as a result, more deadlock situations.

### locklist - Maximum storage for lock list

This parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database and it contains the locks held by all applications concurrently connected to the database.

**Configuration type**

Database

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

UNIX Automatic [4 - 524 288]

**Windows Database server with local and remote clients**

Automatic [4 - 524 288]

**Windows 64-bit Database server with local clients**

Automatic [4 - 524 288]

**Windows 32-bit Database server with local clients**

Automatic [4 - 524 288]

**Unit of measure**

Pages (4 KB)

**When allocated**

When the first application connects to the database

**When freed**

When last application disconnects from the database

Locking is the mechanism that the database manager uses to control concurrent access to data in the database by multiple applications. Both rows and tables can be locked. The database manager can also acquire locks for internal use.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active

The value of *locklist* is tuned together with the *maxlocks* parameter, therefore disabling self tuning of the *locklist* parameter automatically disables self tuning of the *maxlocks* parameter. Enabling self tuning of the *locklist* parameter automatically enables self tuning of the *maxlocks* parameter.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the *self\_tuning\_mem* configuration parameter is set to "ON.")

On 32-bit platforms, each lock requires 48 or 96 bytes of the lock list, depending on whether other locks are held on the object:

- 96 bytes are required to hold a lock on an object that has no other locks held on it
- 48 bytes are required to record a lock on an object that has an existing lock held on it.

On 64-bit platforms (except HP-UX/PA-RISC), each lock requires 64 or 128 bytes of the lock list, depending on whether other locks are held on the object:

- 128 bytes are required to hold a lock on an object that has no other locks held on it
- 64 bytes are required to record a lock on an object that has an existing lock held on it.

On 64-bit HP-UX/PA-RISC, each lock requires 80 or 160 bytes of the lock list, depending on whether or not other locks are held on the object.



When the percentage of the lock list used by one application reaches *maxlocks*, the database manager will perform lock escalation, from row to table, for the locks held by the application. Although the escalation process itself does not take much time, locking entire tables (versus individual rows) decreases concurrency, and overall database performance might decrease for subsequent accesses against the affected tables. Suggestions of how to control the size of the lock list are:

- Perform frequent COMMITs to release locks.
- When performing many updates, lock the entire table before updating (using the SQL LOCK TABLE statement). This will use only one lock, keeps others from interfering with the updates, but does reduce concurrency of the data.  
You can also use the LOCKSIZE option of the ALTER TABLE statement to control how locking is done for a specific table.  
Use of the Repeatable Read isolation level might result in an automatic table lock.
- Use the Cursor Stability isolation level when possible to decrease the number of share locks held. If application integrity requirements are not compromised use Uncommitted Read instead of Cursor Stability to further decrease the amount of locking.
- Set *locklist* to AUTOMATIC. The lock list will increase synchronously to avoid lock escalation or a lock list full situation.

Once the lock list is full, performance can degrade since lock escalation will generate more table locks and fewer row locks, thus reducing concurrency on shared objects in the database. Additionally there might be more deadlocks between applications (since they are all waiting on a limited number of table locks), which will result in transactions being rolled back. Your application will receive an SQLCODE of -912 when the maximum number of lock requests has been reached for the database.

**Recommendation:** If lock escalations are causing performance concerns you might need to increase the value of this parameter or the *maxlocks* parameter. You can use the database system monitor to determine if lock escalations are occurring. Refer to the *lock\_escals* (*lock escalations*) monitor element.

The following steps might help in determining the number of pages required for your lock list:

1. Calculate a lower bound for the size of your lock list, using *one* of the following calculations, depending on your environment:

a.

$$(512 * x * \text{maxapps}) / 4096$$

b. with Concentrator enabled:

$$(512 * x * \text{max\_coordagents}) / 4096$$

c. in a partitioned database with Concentrator enabled:

$$(512 * x * \text{max\_coordagents} * \text{number of database partitions}) / 4096$$

where 512 is an estimate of the average number of locks per application and x is the number of bytes required for each lock against an object that has an existing lock (40 bytes on 32-bit platforms, 64 bytes on 64-bit platforms).

2. Calculate an upper bound for the size of your lock list:

$$(512 * y * \text{maxapps}) / 4096$$

where y is the number of bytes required for the first lock against an object (80 bytes on 32-bit platforms, 128 bytes on 64-bit platforms).

3. Estimate the amount of concurrency you will have against your data and based on your expectations, choose an initial value for *locklist* that falls between the upper and lower bounds that you have calculated.
4. Using the database system monitor, as described below, tune the value of this parameter.

If *maxappls* or *max\_coordagents* are set to AUTOMATIC in your applicable scenario, you should also set *locklist* to AUTOMATIC.

You can use the database system monitor to determine the maximum number of locks held by a given transaction. Refer to the *locks\_held\_top* (*maximum number of locks held*) monitor element.

This information can help you validate or adjust the estimated number of locks per application. In order to perform this validation, you will have to sample several applications, noting that the monitor information is provided at a transaction level, not an application level.

You might also want to increase *locklist* if *maxappls* is increased, or if the applications being run perform infrequent commits.

You should consider rebinding applications (using the REBIND command) after changing this parameter.

## locktimeout - Lock timeout

This parameter specifies the number of seconds that an application will wait to obtain a lock, helping avoid global deadlocks for applications.

### Configuration type

Database

### Parameter type

Configurable

### Default [range]

-1 [-1; 0 - 32 767 ]

### Unit of measure

Seconds

If you set this parameter to 0, locks are not waited for. In this situation, if no lock is available at the time of the request, the application immediately receives a -911.

If you set this parameter to -1, lock timeout detection is turned off. In this situation a lock will be waited for (if one is not available at the time of the request) until either of the following:

- The lock is granted
- A deadlock occurs.

**Recommendation:** In a transaction processing (OLTP) environment, you can use an initial starting value of 30 seconds. In a query-only environment you could start with a higher value. In both cases, you should use benchmarking techniques to tune this parameter.

The value should be set to quickly detect waits that are occurring because of an abnormal situation, such as a transaction that is stalled (possibly as a result of a

user leaving their workstation). You should set it high enough so valid lock requests do not time out because of peak workloads, during which time, there is more waiting for locks.

You can use the database system monitor to help you track the number of times an application (connection) experienced a lock timeout or that a database detected a timeout situation for all applications that were connected.

High values of the *lock\_timeout* (number of lock timeouts) monitor element can be caused by:

- Too low a value for this configuration parameter.
- An application (transaction) that is holding locks for an extended period. You can use the database system monitor to further investigate these applications.
- A concurrency problem, that could be caused by lock escalations (from row-level to a table-level lock).

## **maxlocks - Maximum percent of lock list before escalation**

This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs lock escalation.

### **Configuration type**

Database

### **Parameter type**

Configurable online

### **Propagation class**

Immediate

### **Default [range]**

Automatic [1 - 100 ]

### **Unit of measure**

Percentage

Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation will occur for the locks held by that application. Lock escalation also occurs if the lock list runs out of space.

The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the **maxlocks** value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage of the lock list held is below the value of **maxlocks**. The **maxlocks** parameter multiplied by the **maxappls** parameter cannot be less than 100.

When this parameter is set to **AUTOMATIC**, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active.

The value of **locklist** is tuned together with the **maxlocks** parameter, therefore disabling self tuning of the **locklist** parameter automatically disables self tuning of

the **maxlocks** parameter. Enabling self tuning of the **locklist** parameter automatically enables self tuning of the **maxlocks** parameter.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the **self\_tuning\_mem** configuration parameter is set to ON).

On 32-bit platforms, each lock requires 48 or 96 bytes of the lock list, depending on whether other locks are held on the object:

- 96 bytes are required to hold a lock on an object that has no other locks held on it.
- 48 bytes are required to record a lock on an object that has an existing lock held on it.

On 64-bit platforms (except HP-UX/PA-RISC), each lock requires 64 or 128 bytes of the lock list, depending on whether other locks are held on the object:

- 128 bytes are required to hold a lock on an object that has no other locks held on it.
- 64 bytes are required to record a lock on an object that has an existing lock held on it.

On 64-bit HP-UX/PA-RISC, each lock requires 80 or 160 bytes of the lock list, depending on whether or not other locks are held on the object.

**Recommendation:** The following formula allows you to set **maxlocks** to allow an application to hold twice the average number of locks:

$$\text{maxlocks} = 2 * 100 / \text{maxappls}$$

Where 2 is used to achieve twice the average and 100 represents the largest percentage value allowed. If you have only a few applications that run concurrently, you could use the following formula as an alternative to the first formula:

$$\text{maxlocks} = 2 * 100 / (\text{average number of applications running concurrently})$$

One of the considerations when setting **maxlocks** is to use it in conjunction with the size of the lock list (**locklist**). The actual limit of the number of locks held by an application before lock escalation occurs is:

- $\text{maxlocks} * \text{locklist} * 4096 / (100 * 48)$  on a 32-bit system
- $\text{maxlocks} * \text{locklist} * 4096 / (100 * 80)$  on a 64-bit system  
HP-UX/PA-RISC environment
- $\text{maxlocks} * \text{locklist} * 4096 / (100 * 64)$  on other 64-bit systems

Where 4096 is the number of bytes in a page, 100 is the largest percentage value allowed for **maxlocks**, and 48 is the number of bytes per lock on a 32-bit system, 80 is the number of bytes per lock on a HP-UX/PA-RISC 64-bit system, and 64 is the number of bytes per lock on other 64-bit systems. If you know that one of your applications requires 1000 locks, and you do not want lock escalation to occur, then you should choose values for **maxlocks** and **locklist** in this formula so that the result is greater than 1000. (Using 10 for **maxlocks** and 100 for **locklist**, this formula results in greater than the 1000 locks needed.)

If **maxlocks** is set too low, lock escalation happens when there is still enough lock space for other concurrent applications. If **maxlocks** is set too high, a few

applications can consume most of the lock space, and other applications will have to perform lock escalation. The need for lock escalation in this case results in poor concurrency.

You can use the database system monitor to help you track and tune this configuration parameter.

---

## SSL Configuration Parameters

### **ssl\_svr\_keydb - SSL key file path for incoming SSL connections at the server configuration parameter**

This configuration parameter specifies a fully qualified file path of the key file to be used for SSL setup at server-side.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

Null

The SSL key file has extension **.kdb** by default, and stores personal certificates, personal certificate requests and signer certificates. This key file is accessed during the instance startup and the servers personal certificate is sent to the client for server authentication during SSL handshake.

By default, the value is Null. During the instance start up, you must define if the DB2COMM registry variable contains SSL. Otherwise, the instance starts up without SSL protocol support.

### **ssl\_svr\_stash - SSL stash file path for incoming SSL connections at the server configuration parameter**

This configuration parameter specifies a fully qualified file path of the stash file to be used for SSL setup at server-side.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

Null

The SSL stash file has extension **.sth** by default, and stores an encrypted version of the key database password. The password held in the stash file is used to access the SSL key file during the instance startup.

By default, the value is Null. During the instance start up, you must define if the DB2COMM registry variable contains SSL. Otherwise, the instance starts up without SSL protocol support.

## **ssl\_svr\_label - Label in the key file for incoming SSL connections at the server configuration parameter**

This configuration parameter specifies a label of the personal certificate of the server in the key database.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable

### **Default [range]**

Null

By default, the value is null. When establishing a SSL connection, the server certificate specified by this configuration parameter is sent to the client for server authentication. If the value is null, the default certificate defined in the key file is used. If the default does not exist, the connection fails.

## **ssl\_svcname - SSL service name configuration parameter**

This configuration parameter specifies the name of the port that a database server uses to await communications from remote client nodes using SSL protocol.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable

### **Default [range]**

Null

This configuration parameter contains the port that a database server uses to await communications from remote client nodes through SSL protocol. This service name must be reserved for use by the database manager. During instance startup, you must define if the DB2COMM registry variable contains SSL. Otherwise the instance starts up without SSL protocol support.

If DB2COMM contains both TCP/IP and SSL, the port specified by **ssl\_svcsname** must not be the same as the **svcsname**. Otherwise, the instance starts up without either SSL or TCP/IP protocol support.

On UNIX systems, the services file is located in: `/etc/services`

The database server SSL port (number *n*) and its service name needs to be defined in the services file on the database client.

## **ssl\_versions - Supported SSL versions at the server configuration parameter**

This configuration parameter specifies Secure Sockets Layer (SSL) and Transport Layer Security (TLS) versions that the server supports for incoming connection requests.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable

### **Default**

Null [TLSv1]

If you set the parameter to Null or TLSv1, the parameter enables support for TLS version 1.0 (RFC2246) and TLS version 1.1 (RFC4346).

During SSL handshake, the client and the server negotiate and find the most secure version to use either TLS version 1.0 or TLS version 1.1. If there is no compatible version between the client and the server, the connection fails. If the client supports TLS version 1.0 and TLS version 1.1, but the server support TLS version 1.0 only, then TLS version 1.0 is used.

## **ssl\_cipherspecs - Supported cipher specifications at the server configuration parameter**

This configuration parameter specifies the cipher suites that the server allows for incoming connection requests when using SSL protocol.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable

### **Default [range]**

Null [TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA;



```
TLS_RSA_WITH_AES_128_CBC_SHA ;
TLS_RSA_WITH_3DES_EDE_CBC_SHA]
```

You can specify multiple cipher specifications, such as `TLS_RSA_WITH_AES_256_CBC_SHA` or `TLS_RSA_WITH_AES_128_CBC_SHA` or `TLS_RSA_WITH_3DES_EDE_CBC_SHA` they must be separated by a comma (,) with no space either before or after the comma.

During SSL handshake, if Null or multiple values are specified, the client and the server negotiate and find the most secure cipher suites to use. If no compatible cipher suites is found, the connection fails. You cannot prioritize the cipher suites by specifying one before the another.

## **ssl\_clnt\_keydb - SSL key file path for outbound SSL connections at the client configuration parameter**

This configuration parameter specifies the fully qualified file path of the key file to be used for SSL connection at the client-side.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable

### **Default [range]**

Null

The SSL key file has extension `.kdb` by default, and stores the signer certificate from the servers personal certificate. For a self-signed server personal certificate, the signer certificate is the public key. For a certificate authority signed server personal certificate, the signer certificate is the root CA certificate. The key file is accessed by the client to verify the servers personal certificate during the SSL handshake.

By default, the value is Null. Depending on your application type, you should specify the client SSL key file path by the database manager configuration parameter `ssl_clnt_keydb`, the connection string `ssl_clnt_keydb`, or the `db2cli.ini` keyword `ssl_clnt_keydb` for a SSL connection request. If none of them is specified, the SSL connection fails.

## **ssl\_clnt\_stash - SSL stash file path for outbound SSL connections at the client configuration parameter**

This configuration parameter specifies the fully qualified file path of the stash file to be used for SSL connections at the client-side.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients

- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

Null

The SSL stash file has extension `.sth` by default, and stores an encrypted version of the key database password. The password held in the stash file is used to access the SSL key file during an SSL connection request.

By default the value is Null. Depending on your application type, you can specify the client SSL stash file path by the database manager configuration parameter `ssl_clnt_stash`, the connection string `ssl_clnt_stash`, or the `db2cli.ini` keyword `ssl_clnt_stash` for a SSL connection request. If none of them is specified, the SSL connection fails.

---

## Chapter 14. Communications in a partitioned database environment

The following parameters provide information about communications in the partitioned database environment.

---

### conn\_elapse - Connection elapse time

This parameter specifies the number of seconds within which a TCP/IP connection is to be established between two database partition servers.

**Configuration type**

Database manager

**Applies to**

Partitioned database server with local and remote clients

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

10 [0–100]

**Unit of measure**

Seconds

If the attempt to connect succeeds within the time specified by this parameter, communications are established. If it fails, another attempt is made to establish communications. If the connection is attempted the number of times specified by the *max\_connretries* parameter and always times out, an error is issued.

---

### fcm\_num\_buffers - Number of FCM buffers

This parameter specifies the number of 4 KB buffers that are used for internal communications (messages) both among and within database servers.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

**32-bit platforms**  
Automatic [128 - 65 300]

### 64-bit platforms

Automatic [128 - 524 288]

- Database server with local and remote clients: the default is 1024
- Database server with local clients: the default is 512
- Partitioned database server with local and remote clients: the default is 4096

On single-partition database systems, this parameter is used only if intra-partition parallelism is enabled by changing the *intra\_parallel* parameter from its default value of NO to YES.

It is possible to set both an initial value and the AUTOMATIC attribute.

When set to AUTOMATIC, FCM monitors resource usage and can either increase or decrease resources, if they are not used within 30 minutes. The amount by which resources can be increased or decreased depends on the platform, in particular, that on Linux it can only be increased 25% above the starting value. If the database manager cannot allocate the number of resources specified when an instance is started, it scales back the configuration values incrementally until it can start the instance.

If you have multiple logical nodes on the same machine, you might find it necessary to increase the value of this parameter. You might also find it necessary to increase the value of this parameter if you run out of message buffers because of the number of users on the system, the number of database partition servers on the system, or the complexity of the applications.

If you are using multiple logical nodes, one pool of *fcm\_num\_buffers* buffers is shared by all the multiple logical nodes on the same machine. The size of the pool will be determined by multiplying the *fcm\_num\_buffers* value times the number of logical nodes on that physical machine. Re-examine the value you are using; consider how many FCM buffers in total will be allocated on the machine (or machines) where the multiple logical nodes reside.

---

## fcm\_num\_channels - Number of FCM channels

This parameter specifies the number of FCM channels for each database partition.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

### Parameter type

Configurable Online

### Propagation class

Immediate

### Default [range]

#### UNIX 32-bit platforms

Automatic, with starting values of 256, 512, 2 048 [128 - 120 000 ]

### UNIX 64-bit platforms

Automatic, with starting values of 256, 512, 2 048 [128 - 524 288 ]

### Windows 32-bit

Automatic, with a starting value 10 000 [128 - 120 000 ]

### Windows 64-bit

Automatic, with starting values of 256, 512, 2 048 [128 - 524 288 ]

- For database server with local and remote clients, the starting value is 512.
- For database server with local clients, the starting value is 256.
- For partitioned database environment servers with local and remote clients, the starting value is 2 048.

On non-partitioned database environments, the *intra\_parallel* parameter must be active before *fcm\_num\_channels* can be used.

An FCM channel represents a logical communication end point between EDUs running in the DB2 engine. Both control flows (request and reply) and data flows (table queue data) rely on channels to transfer data between partitions.

When set to AUTOMATIC, FCM monitors channel usage, incrementally allocating and releasing resources as requirements change.

---

## max\_connretries - Node connection retries

This parameter specifies the maximum number of times an attempt will be made to establish a TCP/IP connection between two database partition servers.

### Configuration type

Database manager

### Applies to

Partitioned database server with local and remote clients

### Parameter type

Configurable Online

### Propagation class

Immediate

### Default [range]

5 [0–100]

If the attempt to establish communication between two database partition servers fails (for example, the value specified by the *conn\_elapse* parameter is reached), *max\_connretries* specifies the number of connection retries that can be made to a database partition server. If the value specified for this parameter is exceeded, an error is returned.

---

## max\_time\_diff - Maximum time difference among nodes

This parameter specifies the maximum time difference, in minutes, that is permitted among the database partition servers listed in the node configuration file.

### Configuration type

Database manager

**Applies to**

Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

60 [1 - 1 440]

**Unit of measure**

Minutes

Each database partition server has its own system clock. If two or more database partition servers are associated with a transaction, and the time difference between their clocks is more than the amount specified by the `MAX_TIME_DIFF` parameter, the transaction is rejected and an `SQLCODE` is returned. (The transaction is rejected only if data modification is associated with it.)

A `SQLCODE` may also be returned in database partitioned environments where DB2 compares the system clock to the virtual timestamp (VTS) saved to the `SQLLOGCTL.LFH` log control file. If the timestamp in the `.LFH` file is less than the system time, the time in the database log is set to the VTS until the system clock matches this time. The `SQL1473N` error message will also be returned, despite the time difference between multiple nodes being smaller than `MAX_TIME_DIFF` parameter value.

DB2 uses *Coordinated Universal Time* (UTC), so different time zones are not a consideration when you set this parameter. The Coordinated Universal Time is the same as Greenwich Mean Time.

---

## start\_stop\_time - Start and stop timeout

This parameter specifies the time, in minutes, within which all database partition servers must respond to a `START DBM` or a `STOP DBM` command. It is also used as the timeout value during an `ADD DBPARTITIONNUM` operation.

**Configuration type**

Database manager

**Applies to**

Database server with local and remote clients

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

10 [1 - 1 440]

**Unit of measure**

Minutes

Database partition servers that do not respond to a `DB2START` command within the specified time send a message to the `db2start` error log in the `log` subdirectory of the `sql1ib` subdirectory of the home directory for the instance. You should issue a `DB2STOP` on these nodes before restarting them.

Database partition servers that do not respond to a DB2STOP command within the specified time send a message to the db2stop error log in the log subdirectory of the sql1ib subdirectory of the home directory for the instance. You can either issue db2stop for each database partition server that does not respond, or for all of them. (Those that are already stopped will return stating that they are stopped.)

If a db2start or db2stop operation in a multi-partition database is not completed within the value specified by the *start\_stop\_time* database manager configuration parameter, the database partitions that have timed out will be killed internally. Environments with many database partitions with a low value for *start\_stop\_time* might experience this behavior. To resolve this behavior, increase the value of *start\_stop\_time*.

When adding a new database partition using one of the DB2START, START DATABASE MANAGER, or ADD DBPARTITIONNUM commands, the add database partition operation must determine whether or not each database in the instance is enabled for automatic storage. This is done by communicating with the catalog partition for each database. If automatic storage is enabled, the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, the operation might have to communicate with another database partition server to retrieve the table space definitions for the database partitions that reside on that server. These factors should be considered when determining the value of the *start\_stop\_time* parameter.





---

## Chapter 15. autorestart - Auto restart enable

This parameter determines whether the database manager can, in the event of an abnormal termination of the database, automatically call the restart database utility when an application connects to a database.

**Configuration type**

Database

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

On [ On; Off ]

The restart database utility performs a *Crash recovery* if the database terminated abnormally (because, for example, of a power failure or a system software failure) while applications were connected to it. It applies any committed transactions that were in the database buffer pool but were not written to disk at the time of the failure. It also backs out any uncommitted transactions that might have been written to disk.

If *autorestart* is not enabled, then an application that attempts to connect to a database which needs to have crash recovery performed (needs to be restarted) will receive a SQL1015N error. In this case, the application can call the restart database utility, or you can restart the database by selecting the restart operation of the recovery tool.



---

## Chapter 16. database\_consistent - Database is consistent

This parameter indicates whether the database is in a consistent state.

**Configuration type**

Database

**Parameter type**

Informational

**YES** indicates that all transactions have been committed or rolled back so that the data is consistent. If the system “crashes” while the database is consistent, you do not need to take any special action to make the database usable.

**NO** indicates that a transaction is pending or some other task is pending on the database and the data is not consistent at this point. If the system “crashes” while the database is not consistent, you will need to restart the database using the `RESTART DATABASE` command to make the database usable.



---

## Chapter 17. nodetype - Machine node type

This parameter provides information about the DB2 products which you have installed on your machine and, as a result, information about the type of database manager configuration.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Informational

The following are the possible values returned by this parameter and the products associated with that node type:

- **Database server with local and remote clients** – a DB2 server product, supporting local and remote Data Server Runtime Clients, and capable of accessing other remote database servers.
- **Client** – a Data Server Runtime Client capable of accessing remote database servers.
- **Database server with local clients** – a DB2 relational database management system, supporting local Data Server Runtime Clients and capable of accessing other, remote database servers.
- **Partitioned database server with local and remote clients** – a DB2 server product, supporting local and remote Data Server Runtime Clients, and capable of accessing other remote database servers, and capable of parallelism.



---

## Chapter 18. restrict\_access - Database has restricted access configuration parameter

This parameter indicates whether the database was created using the restrictive set of default actions. In other words, if it was created with the RESTRICTIVE clause in the CREATE DATABASE command.

**Configuration type**

Database

**Parameter type**

Informational

**YES** The RESTRICTIVE clause was used in the CREATE DATABASE command when this database was created.

**NO** The RESTRICTIVE clause was not used in the CREATE DATABASE command when this database was created.





---

## **Part 6. Recovery considerations**



---

## Chapter 19. Crash Recovery and Database Logs

---

### Crash recovery

Transactions (or units of work) against a database can be interrupted unexpectedly. If a failure occurs before all of the changes that are part of the unit of work are completed and committed, the database is left in an inconsistent and unusable state. *Crash recovery* is the process by which the database is moved back to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred (Figure 10). When a database is in a consistent and usable state, it has attained what is known as a "point of consistency".

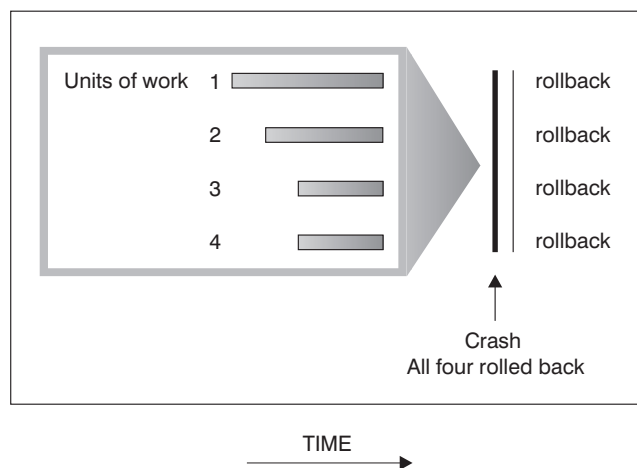


Figure 10. Rolling Back Units of Work (Crash Recovery)

A *transaction failure* results from a severe error or condition that causes the database or the database manager to end abnormally. Partially completed units of work, or UOW that have not been flushed to disk at the time of failure, leave the database in an inconsistent state. Following a transaction failure, the database must be recovered. Conditions that can result in transaction failure include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down
- A hardware failure such as memory corruption, or disk, CPU, or network failure.
- A serious operating system error that causes DB2 to go down
- An application terminating abnormally.

If you want the rollback of incomplete units of work to be done automatically by the database manager, enable the automatic restart (*autorestart*) database configuration parameter by setting it to ON. (This is the default value.) If you do not want automatic restart behavior, set the *autorestart* database configuration parameter to OFF. As a result, you will need to issue the RESTART DATABASE command when a database failure occurs. If the database I/O was suspended before the crash occurred, you must specify the WRITE RESUME option of the RESTART DATABASE command in order for the crash recovery to continue. The administration notification log records when the database restart operation begins.

If crash recovery is applied to a database that is enabled for forward recovery (that is, the *logarchmeth1* configuration parameter is not set to OFF), and an error occurs during crash recovery that is attributable to an individual table space, that table space will be taken offline, and cannot be accessed until it is repaired. Crash recovery continues. At the completion of crash recovery, the other table spaces in the database will be accessible, and connections to the database can be established. However, if the table space that is taken offline is the table space that contains the system catalogs, it must be repaired before any connections will be permitted.

---

## Chapter 20. Application processes, concurrency, and recovery

All SQL programs execute as part of an *application process* or agent. An application process involves the execution of one or more programs, and is the unit to which the database manager allocates resources and locks. Different application processes might involve the execution of different programs, or different executions of the same program.

More than one application process can request access to the same data at the same time. *Locking* is the mechanism that is used to maintain data integrity under such conditions, preventing, for example, two application processes from updating the same row of data simultaneously.

The database manager acquires locks to prevent uncommitted changes made by one application process from being accidentally perceived by any other process. The database manager releases all locks it has acquired and retained on behalf of an application process when that process ends. However, an application process can explicitly request that locks be released sooner. This is done using a *commit* operation, which releases locks that were acquired during a unit of work and also commits database changes that were made during the unit of work.

A *unit of work* (UOW) is a recoverable sequence of operations within an application process. A unit of work is initiated when an application process starts, or when the previous UOW ends because of something other than the termination of the application process. A unit of work ends with a commit operation, a rollback operation, or the end of an application process. A commit or rollback operation affects only the database changes that were made within the UOW that is ending.

The database manager provides a means of backing out of uncommitted changes that were made by an application process. This might be necessary in the event of a failure on the part of an application process, or in the case of a deadlock or lock timeout situation. An application process can explicitly request that its database changes be cancelled. This is done using a *rollback* operation.

As long as these changes remain uncommitted, other application processes are unable to see them, and the changes can be rolled back. This is not true, however, if the prevailing isolation level is uncommitted read (UR). After they are committed, these database changes are accessible to other application processes and can no longer be rolled back.

Both DB2 call level interface (CLI) and embedded SQL allow for a connection mode called *concurrent transactions*, which supports multiple connections, each of which is an independent transaction. An application can have multiple concurrent connections to the same database.

Locks that are acquired by the database manager on behalf of an application process are held until the end of a UOW, except when the isolation level is cursor stability (CS, in which the lock is released as the cursor moves from row to row) or uncommitted read (UR).

An application process is never prevented from performing operations because of its own locks. However, if an application uses concurrent transactions, the locks from one transaction might affect the operation of a concurrent transaction.

The initiation and the termination of a UOW define *points of consistency* within an application process. For example, a banking transaction might involve the transfer of funds from one account to another. Such a transaction would require that these funds be subtracted from the first account, and then added to the second account. Following the subtraction step, the data is inconsistent. Only after the funds have been added to the second account is consistency reestablished. When both steps are complete, the commit operation can be used to end the UOW, thereby making the changes available to other application processes. If a failure occurs before the UOW ends, the database manager will roll back any uncommitted changes to restore data consistency.

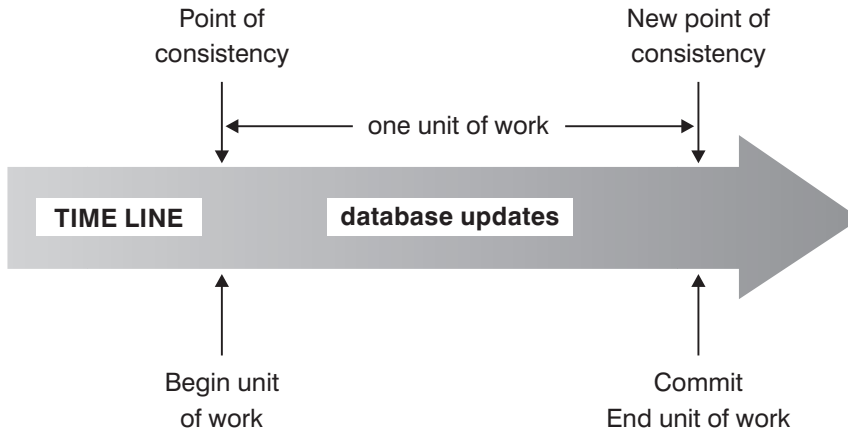


Figure 11. Unit of work with a COMMIT statement

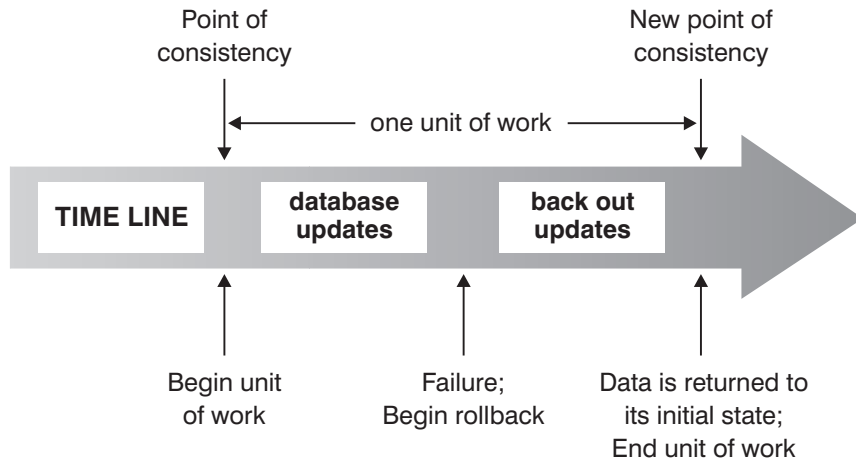


Figure 12. Unit of work with a ROLLBACK statement

---

## Chapter 21. Recovering from transaction failures in a partitioned database environment

If a transaction failure occurs in a partitioned database environment, database recovery is usually necessary on both the failed database partition server and any other database partition server that was participating in the transaction:

- Crash recovery occurs on the failed database partition server after the failure condition is corrected.
- *Database partition failure recovery* on the other (still active) database partition servers occurs immediately after the failure has been detected.

In a partitioned database environment, the database partition server on which a transaction is submitted is the coordinator partition, and the first agent that processes the transaction is the coordinator agent. The coordinator agent is responsible for distributing work to other database partition servers, and it keeps track of which ones are involved in the transaction. When the application issues a COMMIT statement for a transaction, the coordinator agent commits the transaction by using the two-phase commit protocol. During the first phase, the coordinator partition distributes a PREPARE request to all the other database partition servers that are participating in the transaction. These servers then respond with one of the following:

### READ-ONLY

No data change occurred at this server

**YES** Data change occurred at this server

**NO** Because of an error, the server is not prepared to commit

If one of the servers responds with a NO, the transaction is rolled back. Otherwise, the coordinator partition begins the second phase.

During the second phase, the coordinator partition writes a COMMIT log record, then distributes a COMMIT request to all the servers that responded with a YES. After all the other database partition servers have committed, they send an acknowledgement of the COMMIT to the coordinator partition. The transaction is complete when the coordinator agent has received all COMMIT acknowledgments from all the participating servers. At this point, the coordinator agent writes a FORGET log record.

### Transaction failure recovery on an active database partition server

If any database partition server detects that another server is down, all work that is associated with the failed database partition server is stopped:

- If the still active database partition server is the coordinator partition for an application, and the application was running on the failed database partition server (and not ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, SQL0279N is returned to the application, which in turn loses its database connection. Otherwise, the coordinator agent distributes a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.



- If the failed database partition server was the coordinator partition for the application, then agents that are still working for the application on the active servers are interrupted to do failure recovery. The transaction is rolled back locally on each database partition where the transaction is not in prepared state. On those database partitions where the transaction is in prepared state, the transaction becomes in doubt. The coordinator database partition is not aware that the transaction is in doubt on some database partitions because the coordinator database partition is not available.
- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator partition, agents working for this application are interrupted. The coordinator partition will either send a ROLLBACK or a DISCONNECT message to the other database partition servers. The transaction will only be in doubt on database partition servers that are still active if the coordinator partition returns SQL0279.

Any process (such as an agent or deadlock detector) that attempts to send a request to the failed server is informed that it cannot send the request.

### **Transaction failure recovery on the failed database partition server**

If the transaction failure causes the database manager to end abnormally, you can issue the db2start command with the RESTART option to restart the database manager once the database partition has been restarted. If you cannot restart the database partition, you can issue db2start to restart the database manager on a different database partition.

If the database manager ends abnormally, database partitions on the server can be left in an inconsistent state. To make them usable, crash recovery can be triggered on a database partition server:

- Explicitly, through the RESTART DATABASE command
- Implicitly, through a CONNECT request when the *autorestart* database configuration parameter has been set to ON

Crash recovery reapplies the log records in the active log files to ensure that the effects of all complete transactions are in the database. After the changes have been reapplied, all uncommitted transactions are rolled back locally, *except* for indoubt transactions. There are two types of indoubt transaction in a partitioned database environment:

- On a database partition server that is not the coordinator partition, a transaction is in doubt if it is prepared but not yet committed.
- On the coordinator partition, a transaction is in doubt if it is committed but not yet logged as complete (that is, the FORGET record is not yet written). This situation occurs when the coordinator agent has not received all the COMMIT acknowledgments from all the servers that worked for the application.

Crash recovery attempts to resolve all the indoubt transactions by doing one of the following. The action that is taken depends on whether the database partition server was the coordinator partition for an application:

- If the server that restarted is not the coordinator partition for the application, it sends a query message to the coordinator agent to discover the outcome of the transaction.

- If the server that restarted *is* the coordinator partition for the application, it sends a message to all the other agents (subordinate agents) that the coordinator agent is still waiting for COMMIT acknowledgments.

It is possible that crash recovery might not be able to resolve all the indoubt transactions. For example, some of the database partition servers might not be available. If the coordinator partition completes crash recovery before other database partitions involved in the transaction, crash recovery will not be able to resolve the indoubt transaction. This is expected because crash recovery is performed by each database partition independently. In this situation, the SQL warning message SQL1061W is returned. Because indoubt transactions hold resources, such as locks and active log space, it is possible to get to a point where no changes can be made to the database because the active log space is being held up by indoubt transactions. For this reason, you should determine whether indoubt transactions remain after crash recovery, and recover all database partition servers that are required to resolve the indoubt transactions as quickly as possible.

**Note:** In a partitioned database server environment, the RESTART database command is run on a per-node basis. In order to ensure that the database is restarted on all nodes, use the following recommended command:

```
db2_all "db2 restart database <database_name>"
```

If one or more servers that are required to resolve an indoubt transaction cannot be recovered in time, and access is required to database partitions on other servers, you can manually resolve the indoubt transaction by making an heuristic decision. You can use the LIST INDOUBT TRANSACTIONS command to query, commit, and roll back the indoubt transaction on the server.

**Note:** The LIST INDOUBT TRANSACTIONS command is also used in a distributed transaction environment. To distinguish between the two types of indoubt transactions, the *originator* field in the output that is returned by the LIST INDOUBT TRANSACTIONS command displays one of the following:

- DB2 Enterprise Server Edition, which indicates that the transaction originated in a partitioned database environment.
- XA, which indicates that the transaction originated in a distributed environment.

## Identifying the failed database partition server

When a database partition server fails, the application will typically receive one of the following SQLCODEs. The method for detecting which database manager failed depends on the SQLCODE received:

### SQL0279N

This SQLCODE is received when a database partition server involved in a transaction is terminated during COMMIT processing.

### SQL1224N

This SQLCODE is received when the database partition server that failed is the coordinator partition for the transaction.

### SQL1229N

This SQLCODE is received when the database partition server that failed is not the coordinator partition for the transaction.

Determining which database partition server failed is a two-step process.

1. Find the partition server that detected the failure by examining the SQLCA. The SQLCA associated with SQLCODE SQL1229N contains the node number of

the server that detected the error in the sixth array position of the *sqlerrd* field. (The node number that is written for the server corresponds to the node number in the *db2nodes.cfg* file.)

2. Examine the administration notification log on the server found in step one for the node number of the failed server.

**Note:** If multiple logical nodes are being used on a processor, the failure of one logical node can cause other logical nodes on the same processor to fail.

---

## Part 7. Appendixes



## Appendix A. Related topics (linked to from topics in this book)

### SQL Reference topics

#### Assignments and comparisons

The basic operations of SQL are assignment and comparison. Assignment operations are performed during the execution of INSERT, UPDATE, FETCH, SELECT INTO, VALUES INTO and SET transition-variable statements. Arguments of functions are also assigned when invoking a function. Comparison operations are performed during the execution of statements that include predicates and other language elements such as MAX, MIN, DISTINCT, GROUP BY, and ORDER BY.

One basic rule for both operations is that the data type of the operands involved must be compatible. The compatibility rule also applies to set operations.

Another basic rule for assignment operations is that a null value cannot be assigned to a column that cannot contain null values, nor to a host variable that does not have an associated indicator variable.

Following is a compatibility matrix showing the system-defined data type compatibilities for assignment and comparison operations.

Table 41. Data Type Compatibility for Assignments and Comparisons

| Operands               | Binary Integer   | Decimal Number   | Floating-point   | Decimal Floating-point | Character String   | Graphic String     | Date             | Time             | Time-stamp       | Binary String   | Boolean          | UDT          |
|------------------------|------------------|------------------|------------------|------------------------|--------------------|--------------------|------------------|------------------|------------------|-----------------|------------------|--------------|
| Binary Integer         | Yes              | Yes              | Yes              | Yes                    | Yes                | Yes <sup>5</sup>   | No               | No               | No               | No              | No               | <sup>2</sup> |
| Decimal Number         | Yes              | Yes              | Yes              | Yes                    | Yes                | Yes <sup>5</sup>   | No               | No               | No               | No              | No               | <sup>2</sup> |
| Floating point         | Yes              | Yes              | Yes              | Yes                    | Yes                | Yes <sup>5</sup>   | No               | No               | No               | No              | No               | <sup>2</sup> |
| Decimal Floating point | Yes              | Yes              | Yes              | Yes                    | Yes                | Yes <sup>5</sup>   | No               | No               | No               | No              | No               | <sup>2</sup> |
| Character String       | Yes              | Yes              | Yes              | Yes                    | Yes                | Yes <sup>5,6</sup> | Yes              | Yes              | Yes              | No <sup>3</sup> | No               | <sup>2</sup> |
| Graphic string         | Yes <sup>5</sup> | Yes <sup>5</sup> | Yes <sup>5</sup> | Yes <sup>5</sup>       | Yes <sup>5,6</sup> | Yes                | Yes <sup>5</sup> | Yes <sup>5</sup> | Yes <sup>5</sup> | No              | No               | <sup>2</sup> |
| Date                   | No               | No               | No               | No                     | Yes                | Yes <sup>5</sup>   | Yes              | No               | Yes              | No              | No               | <sup>2</sup> |
| Time                   | No               | No               | No               | No                     | Yes                | Yes <sup>5</sup>   | No               | Yes <sup>1</sup> |                  | No              | No               | <sup>2</sup> |
| Time-stamp             | No               | No               | No               | No                     | Yes                | Yes <sup>5</sup>   | Yes              | <sup>1</sup>     | Yes              | No              | No               | <sup>2</sup> |
| Binary string          | No               | No               | No               | No                     | Yes <sup>3</sup>   | No                 | No               | No               | No               | Yes             | No               | <sup>2</sup> |
| Boolean                | No               | No               | No               | No                     | No                 | No                 | No               | No               | No               | No              | Yes <sup>7</sup> | <sup>2</sup> |
| UDT                    | <sup>2</sup>     | <sup>2</sup>     | <sup>2</sup>     | <sup>2</sup>           | <sup>2</sup>       | <sup>2</sup>       | <sup>2</sup>     | <sup>2</sup>     | <sup>2</sup>     | <sup>2</sup>    | <sup>2</sup>     | Yes          |

Table 41. Data Type Compatibility for Assignments and Comparisons (continued)

| Operands                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Binary Integer | Decimal Number | Floating-point | Decimal Floating-point | Character String | Graphic String | Date | Time | Time-stamp | Binary String | Boolean | UDT |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|----------------|----------------|------------------------|------------------|----------------|------|------|------------|---------------|---------|-----|
| <sup>1</sup> A TIMESTAMP value can be assigned to a TIME value; however, a TIME value cannot be assigned to a TIMESTAMP value and a TIMESTAMP value cannot be compared with a TIME value.                                                                                                                                                                                                                                                                                                                                       |                |                |                |                        |                  |                |      |      |            |               |         |     |
| <sup>2</sup> A user-defined distinct type value is only comparable to a value defined with the same user-defined distinct type. In general, assignments are supported between a distinct type value and its source data type. A user-defined structured type is not comparable and can only be assigned to an operand of the same structured type or one of its supertypes. Some additional assignment rules apply to row, cursor, and array types. For additional information see "User-defined type assignments" on page 730. |                |                |                |                        |                  |                |      |      |            |               |         |     |
| <sup>3</sup> Support for assignment only (not comparison) and only for character strings defined as FOR BIT DATA.                                                                                                                                                                                                                                                                                                                                                                                                               |                |                |                |                        |                  |                |      |      |            |               |         |     |
| <sup>4</sup> For information on assignment and comparison of reference types, see "Reference type assignments" on page 732 and "Reference type comparisons" on page 738.                                                                                                                                                                                                                                                                                                                                                        |                |                |                |                        |                  |                |      |      |            |               |         |     |
| <sup>5</sup> Only supported for Unicode databases.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                |                |                |                        |                  |                |      |      |            |               |         |     |
| <sup>6</sup> Bit data and graphic strings are not compatible.                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                |                |                |                        |                  |                |      |      |            |               |         |     |
| <sup>7</sup> Variables of Boolean data type cannot be directly compared; comparison can only be done with the literal values TRUE, FALSE, NULL.                                                                                                                                                                                                                                                                                                                                                                                 |                |                |                |                        |                  |                |      |      |            |               |         |     |

## Numeric assignments

For numeric assignments, overflow is not allowed.

- When assigning to an exact numeric data type, overflow occurs if any digit of the whole part of the number would be eliminated. If necessary, the fractional part of a number is truncated.
- When assigning to an approximate numeric data type or decimal floating-point, overflow occurs if the most significant digit of the whole part of the number is eliminated. For floating-point and decimal floating-point numbers, the whole part of the number is the number that would result if the floating-point or decimal floating-point number were converted to a decimal number with unlimited precision. If necessary, rounding may cause the least significant digits of the number to be eliminated.

For decimal floating-point, truncation of the whole part of the number is not allowed and results in an error.

For floating-point numbers, underflow is also not allowed. Underflow occurs for numbers between 1 and -1 if the most significant digit other than zero would be eliminated. For decimal floating-point, underflow is allowed and depending on the rounding mode, results in zero or the smallest positive number or the largest negative number that can be represented along with a warning.

An overflow or underflow warning is returned instead of an error if an overflow or underflow occurs on assignment to a host variable with an indicator variable. In this case, the number is not assigned to the host variable and the indicator variable is set to negative 2.

For decimal floating-point numbers, the CURRENT DECFLOAT ROUNDING MODE special register indicates the rounding mode in effect.

## Assignments to integer

When a decimal, floating-point, or decimal floating-point number is assigned to an integer column or variable, the fractional part of the number is eliminated. As a result, a number between 1 and -1 is reduced to 0.

## Assignments to decimal

When an integer is assigned to a decimal column or variable, the number is first converted to a temporary decimal number and then, if necessary, to the precision and scale of the target. The precision and scale of the temporary decimal number is 5,0 for a small integer, 11,0 for a large integer, or 19,0 for a big integer.

When a decimal number is assigned to a decimal column or variable, the number is converted, if necessary, to the precision and the scale of the target. The necessary number of leading zeros is added, and in the fractional part of the decimal number the necessary number of trailing zeros is added, or the necessary number of trailing digits is eliminated.

When a floating-point number is assigned to a decimal column or variable, the number is first converted to a temporary decimal number of precision 31, and then, if necessary, truncated to the precision and scale of the target. In this conversion, the number is rounded (using floating-point arithmetic) to a precision of 31 decimal digits. As a result, a number between 1 and -1 that is less than the smallest positive number or greater than the largest negative number that can be represented in the decimal column or variable is reduced to 0. The scale is given the largest possible value that allows the whole part of the number to be represented without loss of significance.

When a decimal floating-point number is assigned to a decimal column or variable, the number is rounded to the precision and scale of the decimal column or variable. As a result, a number between 1 and -1 that is less than the smallest positive number or greater than the largest negative number that can be represented in the decimal column or variable is reduced to 0 or rounded to the smallest positive or largest negative value that can be represented in the decimal column or variable, depending on the rounding mode.

## Assignments to floating-point

Floating-point numbers are approximations of real numbers. Hence, when an integer, decimal, floating-point, or decimal floating-point number is assigned to a floating-point column or variable, the result may not be identical to the original number. The number is rounded to the precision of the floating-point column or variable using floating-point arithmetic. A decimal floating-point value is first converted to a string representation, and is then converted to a floating-point number.

## Assignments to decimal floating-point

When an integer number is assigned to a decimal floating-point column or variable, the number is first converted to a temporary decimal number and then to a decimal floating-point number. The precision and scale of the temporary decimal number is 5,0 for a small integer, 11,0 for a large integer, or 19,0 for a big integer. Rounding may occur when assigning a BIGINT to a DECFLOAT(16) column or variable.

When a decimal number is assigned to a decimal floating-point column or variable, the number is converted to the precision (16 or 34) of the target. Leading zeros are eliminated. Depending on the precision and scale of the decimal number and the precision of the target, the value might be rounded.



When a floating-point number is assigned to a decimal floating-point column or variable, the number is first converted to a temporary string representation of the floating-point number. The string representation of the number is then converted to decimal floating-point.

When a DECFLOAT(16) number is assigned to a DECFLOAT(34) column or variable, the resulting value is identical to the DECFLOAT(16) number.

When a DECFLOAT(34) number is assigned to a DECFLOAT(16) column or variable, the exponent of the source is converted to the corresponding exponent in the result format. The mantissa of the DECFLOAT(34) number is rounded to the precision of the target.

## Assignments from strings to numeric

When a string is assigned to a numeric data type, it is converted to the target numeric data type using the rules for a CAST specification. For more information, see “CAST specification” in the *SQL Reference, Volume 1*.

## String assignments

There are two types of assignments:

- In *storage assignment*, a value is assigned and truncation of significant data is not desirable; for example, when assigning a value to a column
- In *retrieval assignment*, a value is assigned and truncation is allowed; for example, when retrieving data from the database

The rules for string assignment differ based on the assignment type.

## Storage assignment

The basic rule is that the length of the string assigned to the target must not be greater than the length attribute of the target. If the length of the string is greater than the length attribute of the target, the following actions might occur:

- The string is assigned with trailing blanks truncated (from all string types except LOB strings) to fit the length attribute of the target
- An error is returned (SQLSTATE 22001) when:
  - Non-blank characters would be truncated from other than a LOB string
  - Any character (or byte) would be truncated from a LOB string

If a string is assigned to a fixed-length target, and the length of the string is less than the length attribute of the target, the string is padded to the right with the necessary number of single-byte, double-byte, or UCS-2 blanks. The pad character is always a blank, even for columns defined with the FOR BIT DATA attribute. (UCS-2 defines several SPACE characters with different properties. For a Unicode database, the database manager always uses the ASCII SPACE at position x'0020' as UCS-2 blank. For an EUC database, the IDEOGRAPHIC SPACE at position x'3000' is used for padding GRAPHIC strings.)

## Retrieval assignment

The length of a string that is assigned to a target can be longer than the length attribute of the target. When a string is assigned to a target, and the length of the string is longer than the length attribute of the target, the string is truncated on the

right by the necessary number of characters (or bytes). When this occurs, a warning is returned (SQLSTATE 01004), and the value 'W' is assigned to the SQLWARN1 field of the SQLCA.

Furthermore, if an indicator variable is provided, and the source of the value is not a LOB, the indicator variable is set to the original length of the string.

If a character string is assigned to a fixed-length target, and the length of the string is less than the length attribute of the target, the string is padded to the right with the necessary number of single-byte, double-byte, or UCS-2 blanks. The pad character is always a blank, even for strings defined with the FOR BIT DATA attribute. (UCS-2 defines several SPACE characters with different properties. For a Unicode database, the database manager always uses the ASCII SPACE at position x'0020' as UCS-2 blank. For an EUC database, the IDEOGRAPHIC SPACE at position x'3000' is used for padding GRAPHIC strings.)

Retrieval assignment of C NUL-terminated host variables is handled on the basis of options that are specified with the PREP or BIND command.

### **Conversion rules for string assignments**

A character string or graphic string assigned to a column or host variable is first converted, if necessary, to the code page of the target. Character conversion is necessary only if all of the following are true:

- The code pages are different.
- The string is neither null nor empty.
- Neither string has a code page value of 0 (FOR BIT DATA).

For Unicode databases, character strings can be assigned to a graphic column, and graphic strings can be assigned to a character column.

### **MBCS considerations for character string assignments**

There are several considerations when assigning character strings that could contain both single and multi-byte characters. These considerations apply to all character strings, including those defined as FOR BIT DATA.

- Blank padding is always done using the single-byte blank character (X'20').
- Blank truncation is always done based on the single-byte blank character (X'20'). The double-byte blank character is treated like any other character with respect to truncation.
- Assignment of a character string to a host variable may result in fragmentation of MBCS characters if the target host variable is not large enough to contain the entire source string. If an MBCS character is fragmented, each byte of the MBCS character fragment in the target is set to a single-byte blank character (X'20'), no further bytes are moved from the source, and SQLWARN1 is set to 'W' to indicate truncation. Note that the same MBCS character fragment handling applies even when the character string is defined as FOR BIT DATA.

### **DBCS considerations for graphic string assignments**

Graphic string assignments are processed in a manner analogous to that for character strings. For non-Unicode databases, graphic string data types are compatible only with other graphic string data types, and never with numeric, character string, or datetime data types. For Unicode databases, graphic string data

types are compatible with character string data types. However, graphic and character string data types cannot be used interchangeably in the SELECT INTO or the VALUES INTO statement.

If a graphic string value is assigned to a graphic string column, the length of the value must not be greater than the length of the column.

If a graphic string value (the 'source' string) is assigned to a fixed length graphic string data type (the 'target', which can be a column or host variable), and the length of the source string is less than that of the target, the target will contain a copy of the source string which has been padded on the right with the necessary number of double-byte blank characters to create a value whose length equals that of the target.

If a graphic string value is assigned to a graphic string host variable and the length of the source string is greater than the length of the host variable, the host variable will contain a copy of the source string which has been truncated on the right by the necessary number of double-byte characters to create a value whose length equals that of the host variable. (Note that for this scenario, truncation need not be concerned with bisection of a double-byte character; if bisection were to occur, either the source value or target host variable would be an ill-defined graphic string data type.) The warning flag SQLWARN1 in the SQLCA will be set to 'W'. The indicator variable, if specified, will contain the original length (in double-byte characters) of the source string. In the case of DBCLOB, however, the indicator variable does not contain the original length.

Retrieval assignment of C NUL-terminated host variables (declared using wchar\_t) is handled based on options specified with the PREP or BIND command.

## Assignments from numeric to strings

When a number is assigned to a string data type, it is converted to the target string data type using the rules for a CAST specification. For more information, see "CAST specification" in the *SQL Reference, Volume 1*.

If a nonblank character is truncated during the cast of a numeric value to a character or graphic data type, a warning is returned. This truncation behavior is unlike the assignment to a character or graphic data type that follows storage assignment rules, where if a nonblank character is truncated during assignment, an error is returned.

## Datetime assignments

A TIME value can be assigned only to a TIME column or to a string variable or string column.

A DATE can be assigned to a DATE, TIMESTAMP or string data type. When a DATE value is assigned to a TIMESTAMP data type, the missing time information is assumed to be all zeros.

A TIMESTAMP value can be assigned to a DATE, TIME, TIMESTAMP or string data type. When a TIMESTAMP value is assigned to a DATE data type, the date portion is extracted and the time portion is truncated. When a TIMESTAMP value is assigned to a TIME data type, the date portion is ignored and the time portion is extracted, but with the fractional seconds truncated. When a TIMESTAMP value is assigned to a TIMESTAMP with lower precision, the excess fractional seconds are

truncated. When a `TIMESTAMP` value is assigned to a `TIMESTAMP` with higher precision, missing digits are assumed to be zeros.

The assignment must not be to a `CLOB`, `DBCLOB`, or `BLOB` variable or column.

When a datetime value is assigned to a string variable or string column, conversion to a string representation is automatic. Leading zeros are not omitted from any part of the date, time, or timestamp. The required length of the target will vary, depending on the format of the string representation. If the length of the target is greater than required, and the target is a fixed-length string, it is padded on the right with blanks. If the length of the target is less than required, the result depends on the type of datetime value involved, and on the type of target.

When the target is not a host variable and has a character data type, truncation is not allowed. The length attribute of the column must be at least 10 for a date, 8 for a time, 19 for a `TIMESTAMP(0)`, and  $20+p$  for `TIMESTAMP(p)`.

When the target is a string host variable, the following rules apply:

- **For a DATE:** If the length of the host variable is less than 10 characters, an error is returned.
- **For a TIME:** If the USA format is used, the length of the host variable must not be less than 8 characters; in other formats the length must not be less than 5 characters.  
If ISO or JIS formats are used, and if the length of the host variable is less than 8 characters, the seconds part of the time is omitted from the result and assigned to the indicator variable, if provided. The `SQLWARN1` field of the `SQLCA` is set to indicate the omission.
- **For a TIMESTAMP:** If the length of the host variable is less than 19 characters, an error is returned. If the length is less than 32 characters, but greater than or equal to 19 characters, trailing digits of the fractional seconds part of the value are omitted. The `SQLWARN1` field of the `SQLCA` is set to indicate the omission.

When a `DATE` is assigned to a `TIMESTAMP`, the time and fractional components of the timestamp are set to midnight and 0, respectively. When a `TIMESTAMP` is assigned to a `DATE`, the date portion is extracted and the time and fractional components are truncated.

When a `TIMESTAMP` is assigned to a `TIME`, the `DATE` portion is ignored and the fractional components are truncated.

## XML assignments

The general rule for XML assignments is that only an XML value can be assigned to XML columns or to XML variables. There are exceptions to this rule, as follows.

- **Processing of input XML host variables:** This is a special case of the XML assignment rule, because the host variable is based on a string value. To make the assignment to XML within SQL, the string value is implicitly parsed into an XML value using the setting of the `CURRENT IMPLICIT XMLPARSE OPTION` special register. This determines whether to preserve or to strip whitespace, unless the host variable is an argument of the `XMLVALIDATE` function, which always strips unnecessary whitespace.
- **Assigning strings to input parameter markers of data type XML:** If an input parameter marker has an implicit or explicit data type of XML, the value bound (assigned) to that parameter marker could be a character string variable, graphic

string variable, or binary string variable. In this case, the string value is implicitly parsed into an XML value using the setting of the CURRENT IMPLICIT XMLPARSE OPTION special register to determine whether to preserve or to strip whitespace, unless the parameter marker is an argument of the XMLVALIDATE function, which always strips unnecessary whitespace.

- **Assigning strings directly to XML columns in data change statements:** If assigning directly to a column of type XML in a data change statement, the assigned expression can also be a character string or a binary string. In this case, the result of XMLPARSE (DOCUMENT *expression* STRIP WHITESPACE) is assigned to the target column. The supported string data types are defined by the supported arguments for the XMLPARSE function. Note that this XML assignment exception does not allow character or binary string values to be assigned to SQL variables or to SQL parameters of data type XML.
- **Assigning XML to strings on retrieval:** If retrieving XML values into host variables using a FETCH INTO statement or an EXECUTE INTO statement in embedded SQL, the data type of the host variable can be CLOB, DBCLOB, or BLOB. If using other application programming interfaces (such as CLI, JDBC, or .NET), XML values can be retrieved into the character, graphic, or binary string types that are supported by the application programming interface. In all of these cases, the XML value is implicitly serialized to a string encoded in UTF-8 and, for character or graphic string variables, converted into the client code page.

Character string or binary string values cannot be retrieved into XML host variables. Values in XML host variables cannot be assigned to columns, SQL variables, or SQL parameters of a character string data type or a binary string data type.

Assignment to XML parameters and variables in inlined SQL bodied UDFs and SQL procedures is done by reference. Passing parameters of data type XML to invoke an inlined SQL UDF or SQL procedure is also done by reference. When XML values are passed by reference, any input node trees are used directly. This direct usage preserves all properties, including document order, the original node identities, and all parent properties.

## User-defined type assignments

For distinct types and structured types, different rules are applied for assignments to host variables than are used for all other assignments.

**Distinct Types:** Assignment to host variables is done based on the source type of the distinct type. That is, it follows the rule:

- A value of a distinct type on the right hand side of an assignment is assignable to a host variable on the left hand side if and only if the source type of this distinct type is assignable to this host variable.

If the target of the assignment is a column based on a distinct type, the source data type must be castable to the target data type.

**Structured Types:** Assignment to and from host variables is based on the declared type of the host variable; that is, it follows the rule:

- A value of a structured type on the right hand side of an assignment is assignable to a host variable on the left hand side if and only if the declared type of the host variable is the structured type or a supertype of the structured type.

If the target of the assignment is a column of a structured type, the source data type must be the target data type or a subtype of the target data type.

For array types, different rules are applied for assignments to SQL variables and parameters. The validity of an assignment to an SQL variable or parameter is determined according to the following rules:

- If the right hand side of the assignment is an SQL variable or parameter, an invocation of the TRIM\_ARRAY function, an invocation of the ARRAY\_DELETE function, or a CAST expression, then its type must be the same as the type of the SQL variable or parameter on the left hand side of the assignment.
- If the right hand side of the assignment is an array constructor or an invocation of the ARRAY\_AGG function, then it is implicitly cast to the type of the SQL variable or parameter on the left hand side.

For example, assuming that the type of variable *V* is MYARRAY, the statement:

```
SET V = ARRAY[1,2,3];
```

is equivalent to:

```
SET V = CAST(ARRAY[1,2,3] AS MYARRAY);
```

And the statement:

```
SELECT ARRAY_AGG(C1) INTO V FROM T
```

is equivalent to:

```
SELECT CAST(ARRAY_AGG(C1) AS MYARRAY) INTO V FROM T
```

Additional information about specific user-defined types is in the sections that follow:

## Array type assignments

The value for an element of an array must be assignable to the data type of the array elements. The assignment rules for that data type apply to the value assignment. The value specified for an index in the array must be assignable to the data type of the index for the array. The assignment rules for that data type apply to the value assignment. For an ordinary array the index data type is INTEGER and for an associative array the data type is either INTEGER or VARCHAR(*n*), where *n* is any valid length attribute for the VARCHAR data type. If the index value for an assignment to an ordinary array is larger the current cardinality of the array, then the cardinality of the array is increased to the new index value, provided the value does not exceed the maximum value for an INTEGER data type. An assignment of one new element to an associative array increases the cardinality by exactly 1 since the index values can be sparse.

The following are valid assignments involving array type values:

- Array variable to another array variable with the same array type as the source variable.
- An expression of type array to an array variable, where the array element type in the source expression is assignable to the array element type in the target array variable.

## Row type assignments

Assignments to fields within a row variable must conform to the same rules as if the field itself was a variable of the same data type as the field. A row variable can



be assigned only to a row variable with the same user-defined row type. When using FETCH, SELECT, or VALUES INTO to assign values to a row variable, the source value types must be assignable to the target row fields. If the source or the target variable (or both) of an assignment is anchored to the row of a table or view, the number of fields must be the same and the field types of the source value must be assignable to the field types of the target value.

### **Cursor type assignments**

Assignments to cursors depend on the type of cursor. The following values are assignable to a variable or parameter of built-in type CURSOR:

- A cursor value constructor
- A value of built-in type CURSOR
- A value of any user-defined cursor type

The following values are assignable to a variable or parameter of a weakly-typed user defined cursor type:

- A cursor value constructor
- A value of built-in type CURSOR
- A value of a user-defined cursor type with the same type name

The following values are assignable to a variable or parameter of strongly-typed user defined cursor type:

- A cursor value constructor
- A value of a user-defined cursor type with the same type name

### **Boolean type assignments**

The following system-defined values are assignable to a variable, parameter, or return type of built-in type BOOLEAN:

- TRUE
- FALSE
- NULL

The result of the evaluation of a search condition can also be assigned. If the search condition evaluates to unknown, the value of NULL is assigned.

### **Reference type assignments**

A reference type with a target type of  $T$  can be assigned to a reference type column that is also a reference type with target type of  $S$  where  $S$  is a supertype of  $T$ . If an assignment is made to a scoped reference column or variable, no check is performed to ensure that the actual value being assigned exists in the target table or view defined by the scope.

Assignment to host variables is done based on the representation type of the reference type. That is, it follows the rule:

- A value of a reference type on the right hand side of an assignment is assignable to a host variable on the left hand side if and only if the representation type of this reference type is assignable to this host variable.

If the target of the assignment is a column, and the right hand side of the assignment is a host variable, the host variable must be explicitly cast to the reference type of the target column.

## Numeric comparisons

Numbers are compared algebraically; that is, with regard to sign. For example, -2 is less than +1.

If one number is an integer and the other is decimal, the comparison is made with a temporary copy of the integer, which has been converted to decimal.

When decimal numbers with different scales are compared, the comparison is made with a temporary copy of one of the numbers that has been extended with trailing zeros so that its fractional part has the same number of digits as the other number.

If one number is floating-point and the other is integer or decimal, the comparison is made with a temporary copy of the other number, which has been converted to double-precision floating-point.

Two floating-point numbers are equal only if the bit configurations of their normalized forms are identical.

If one number is decimal floating-point and the other number is integer, decimal, single precision floating-point, or double precision floating-point, the comparison is made with a temporary copy of the other number, which has been converted to decimal floating-point.

If one number is DECFLOAT(16) and the other number is DECFLOAT(34), the DECFLOAT(16) value is converted to DECFLOAT(34) before the comparison is made.

The decimal floating-point data type supports both positive and negative zero. Positive and negative zero have different binary representations, but the = (equal) predicate will return true for comparisons of negative and positive zero.

The COMPARE\_DECFLOAT and TOTALORDER scalar functions can be used to perform comparisons at a binary level if, for example, a comparison of 2.0 <> 2.00 is required.

The decimal floating-point data type supports the specification of negative and positive NaN (quiet and signalling), and negative and positive infinity. From an SQL perspective, INFINITY = INFINITY, NAN = NAN, SNAN = SNAN, and -0 = 0.

The comparison and ordering rules for special values are as follows:

- (+/-) INFINITY compares equal only to (+/-) INFINITY of the same sign.
- (+/-) NAN compares equal only to (+/-) NAN of the same sign.
- (+/-) SNAN compares equal only to (+/-) SNAN of the same sign.

The ordering among different special values is as follows:

- -NAN < -SNAN < -INFINITY < 0 < INFINITY < SNAN < NAN



When string and numeric data types are compared, the string is cast to DECFLOAT(34) using the rules for a CAST specification. For more information, see "CAST specification" in the *SQL Reference, Volume 1*. The string must contain a valid string representation of a number.

## String comparisons

Character strings are compared according to the collating sequence specified when the database was created, except those with a FOR BIT DATA attribute, which are always compared according to their bit values.

When comparing character strings of unequal lengths, the comparison is made using a logical copy of the shorter string, which is padded on the right with blanks sufficient to extend its length to that of the longer string. This logical extension is done for all character strings, including those tagged as FOR BIT DATA.

Character strings (except character strings tagged as FOR BIT DATA) are compared according to the collating sequence specified when the database was created. For example, the default collating sequence supplied by the database manager may give lowercase and uppercase versions of the same character the same weight. The database manager performs a two-pass comparison to ensure that only identical strings are considered equal to each other. In the first pass, strings are compared according to the database collating sequence. If the weights of the characters in the strings are equal, a second "tie-breaker" pass is performed to compare the strings on the basis of their actual code point values.

Two strings are equal if they are both empty or if all corresponding bytes are equal. If either operand is null, the result is unknown.

LOB strings are not supported in any comparison operations that use the basic comparison operators (=, <>, <, >, <=, and >=). They are supported in comparisons using the LIKE predicate and the POSSTR function.

Portions of strings can be compared using the SUBSTR and VARCHAR scalar functions. For example, given the columns:

```
MY_SHORT_CLOB CLOB(300)
MY_LONG_VAR VARCHAR(8000)
```

then the following is valid:

```
WHERE VARCHAR(MY_SHORT_CLOB) > VARCHAR(SUBSTR(MY_LONG_VAR,1,300))
```

Examples:

For these examples, 'A', 'Á', 'a', and 'á', have the code point values X'41', X'C1', X'61', and X'E1' respectively.

Consider a collating sequence where the characters 'A', 'Á', 'a', 'á' have weights 136, 139, 135, and 138. Then the characters sort in the order of their weights as follows:

```
'a' < 'A' < 'á' < 'Á'
```

Now consider four DBCS characters D1, D2, D3, and D4 with code points 0xC141, 0xC161, 0xE141, and 0xE161, respectively. If these DBCS characters are in CHAR columns, they sort as a sequence of bytes according to the collation weights of

those bytes. First bytes have weights of 138 and 139, therefore D3 and D4 come before D2 and D1; second bytes have weights of 135 and 136. Hence, the order is as follows:

D4 < D3 < D2 < D1

However, if the values being compared have the FOR BIT DATA attribute, or if these DBCS characters were stored in a GRAPHIC column, the collation weights are ignored, and characters are compared according to their code points as follows:

'A' < 'a' < 'Ã' < 'ã'

The DBCS characters sort as sequence of bytes, in the order of code points as follows:

D1 < D2 < D3 < D4

Now consider a collating sequence where the characters 'A', 'Ã', 'a', 'ã' have (non-unique) weights 74, 75, 74, and 75. Considering collation weights alone (first pass), 'a' is equal to 'A', and 'ã' is equal to 'Ã'. The code points of the characters are used to break the tie (second pass) as follows:

'A' < 'a' < 'Ã' < 'ã'

DBCS characters in CHAR columns sort a sequence of bytes, according to their weights (first pass) and then according to their code points to break the tie (second pass). First bytes have equal weights, so the code points (0xC1 and 0xE1) break the tie. Therefore, characters D1 and D2 sort before characters D3 and D4. Then the second bytes are compared in similar way, and the final result is as follows:

D1 < D2 < D3 < D4

Once again, if the data in CHAR columns have the FOR BIT DATA attribute, or if the DBCS characters are stored in a GRAPHIC column, the collation weights are ignored, and characters are compared according to their code points:

D1 < D2 < D3 < D4

For this particular example, the result happens to be the same as when collation weights were used, but obviously this is not always the case.

## Conversion rules for comparison

When two strings are compared, one of the strings is first converted, if necessary, to the encoding scheme and code page of the other string.

## Ordering of results

Results that require sorting are ordered based on the string comparison rules discussed in "String comparisons" on page 734. The comparison is performed at the database server. On returning results to the client application, code page conversion may be performed. This subsequent code page conversion does not affect the order of the server-determined result set.

## MBCS considerations for string comparisons

Mixed SBCS/MBCS character strings are compared according to the collating sequence specified when the database was created. For databases created with default (SYSTEM) collation sequence, all single-byte ASCII characters are sorted in correct order, but double-byte characters are not necessarily in code point sequence. For databases created with IDENTITY sequence, all double-byte

characters are correctly sorted in their code point order, but single-byte ASCII characters are sorted in their code point order as well. For databases created with COMPATIBILITY sequence, a compromise order is used that sorts properly for most double-byte characters, and is almost correct for ASCII. This was the default collation table in DB2 Version 2.

Mixed character strings are compared byte-by-byte. This may result in unusual results for multi-byte characters that occur in mixed strings, because each byte is considered independently.

Example:

For this example, 'A', 'B', 'a', and 'b' double-byte characters have the code point values X'8260', X'8261', X'8281', and X'8282', respectively.

Consider a collating sequence where the code points X'8260', X'8261', X'8281', and X'8282' have weights 96, 65, 193, and 194. Then:

'B' < 'A' < 'a' < 'b'

and

'AB' < 'AA' < 'Aa' < 'Ab' < 'aB' < 'aA' < 'aa' < 'ab'

Graphic string comparisons are processed in a manner analogous to that for character strings.

Graphic string comparisons are valid between all graphic string data types except DBCLOB.

For graphic strings, the collating sequence of the database is not used. Instead, graphic strings are always compared based on the numeric (binary) values of their corresponding bytes.

Using the previous example, if the literals were graphic strings, then:

'A' < 'B' < 'a' < 'b'

and

'AA' < 'AB' < 'Aa' < 'Ab' < 'aA' < 'aB' < 'aa' < 'ab'

When comparing graphic strings of unequal lengths, the comparison is made using a logical copy of the shorter string which is padded on the right with double-byte blank characters sufficient to extend its length to that of the longer string.

Two graphic values are equal if they are both empty or if all corresponding graphics are equal. If either operand is null, the result is unknown. If two values are not equal, their relation is determined by a simple binary string comparison.

As indicated in this section, comparing strings on a byte by byte basis can produce unusual results; that is, a result that differs from what would be expected in a character by character comparison. The examples shown here assume the same MBCS code page, however, the situation can be further complicated when using different multi-byte code pages with the same national language. For example, consider the case of comparing a string from a Japanese DBCS code page and a Japanese EUC code page.

## Datetime comparisons

A DATE, TIME, or TIMESTAMP value may be compared either with another value of the same data type or with a string representation of that data type. A DATE or a string representation of a date can also be compared with a TIMESTAMP, where the missing time information for the date value is assumed to be all zeros. All comparisons are chronological, which means the farther a point in time is from January 1, 0001, the greater the value of that point in time.

Comparisons involving TIME values and string representations of time values always include seconds. If the string representation omits seconds, zero seconds is implied.

Comparisons involving TIMESTAMP values are chronological without regard to representations that might be considered equivalent.

Example:

```
TIMESTAMP('1990-02-23-00.00.00') > '1990-02-22-24.00.00'
```

## User-defined type comparisons

Values with a user-defined distinct type can only be compared with values of exactly the same user-defined distinct type. The user-defined distinct type must have been defined using the WITH COMPARISONS clause.

Example:

Given the following YOUTH distinct type and CAMP\_DB2\_ROSTER table:

```
CREATE TYPE YOUTH AS INTEGER WITH COMPARISONS

CREATE TABLE CAMP_DB2_ROSTER
(NAME VARCHAR(20),
 ATTENDEE_NUMBER INTEGER NOT NULL,
 AGE YOUTH,
 HIGH_SCHOOL_LEVEL YOUTH)
```

The following comparison is valid:

```
SELECT * FROM CAMP_DB2_ROSTER
WHERE AGE > HIGH_SCHOOL_LEVEL
```

The following comparison is not valid:

```
SELECT * FROM CAMP_DB2_ROSTER
WHERE AGE > ATTENDEE_NUMBER
```

However, AGE can be compared to ATTENDEE\_NUMBER by using a function or CAST specification to cast between the distinct type and the source type. The following comparisons are all valid:

```
SELECT * FROM CAMP_DB2_ROSTER
WHERE INTEGER(AGE) > ATTENDEE_NUMBER
```

```
SELECT * FROM CAMP_DB2_ROSTER
WHERE CAST(AGE AS INTEGER) > ATTENDEE_NUMBER
```

```
SELECT * FROM CAMP_DB2_ROSTER
WHERE AGE > YOUTH(ATTENDEE_NUMBER)
```

```
SELECT * FROM CAMP_DB2_ROSTER
WHERE AGE > CAST(ATTENDEE_NUMBER AS YOUTH)
```

Values with a user-defined structured type cannot be compared with any other value (the NULL predicate and the TYPE predicate can be used).

Comparisons of array type values are not supported. Elements of arrays can be compared based on the comparison rules for the data type of the elements.

### **Row type comparisons**

A row variable cannot be compared to another row variable even if the row type name is the same. Individual fields within a row type can be compared to other values and the comparison rules for the data type of the field apply.

### **Cursor type comparisons**

A cursor variable cannot be compared to another cursor variable even if the cursor type name is the same.

### **Boolean type comparisons**

A Boolean value can be compared with a Boolean literal value. A value of TRUE is greater than a value of FALSE.

### **Reference type comparisons**

Reference type values can be compared only if their target types have a common supertype. The appropriate comparison function will only be found if the schema name of the common supertype is included in the SQL path. The comparison is performed using the representation type of the reference types. The scope of the reference is not considered in the comparison.

### **XML comparisons in a non-Unicode database**

When performed in a non-Unicode database, comparisons between XML data and character or graphic string values require a code page conversion of one of the two sets of data being compared. Character or graphic values used in an SQL or XQuery statement, either as a query predicate or as a host variable with a character or graphic string data type, are converted to the database code page prior to comparison. If any characters included in this data have code points that are not part of the database code page, substitution characters are added in their place, potentially causing unexpected results for the query.

For example, a client with a UTF-8 code page is used to connect to a database server created with the Greek encoding ISO8859-7. The expression  $\Sigma_G \Sigma_M$  is sent as the predicate of an XQuery statement, where  $\Sigma_G$  represents the Greek sigma character in Unicode (U+03A3) and  $\Sigma_M$  represents the mathematical symbol sigma in Unicode (U+2211). This expression is first converted to the database code page, so that both "Σ" characters are converted to the equivalent code point for sigma in the Greek database code page, 0xD3. We may denote this code point as  $\Sigma_A$ . The newly converted expression  $\Sigma_A \Sigma_A$  is then converted again to UTF-8 for comparison with the target XML data. Since the distinction between these two code points was lost as a result of the code page conversion required to pass the predicate expression into the database, the two initially distinct values  $\Sigma_G$  and  $\Sigma_M$  are passed to the XML parser as the expression  $\Sigma_G \Sigma_G$ . This expression then fails to match when compared to the value  $\Sigma_G \Sigma_M$  in an XML document.

One way to avoid the unexpected query results that may be caused by code page conversion issues is to ensure that all characters used in a query expression have matching code points in the database code page. Characters that do not have matching code points can be included through the use of a Unicode character entity reference. A character entity reference will always bypass code page conversion. For example, using the character entity reference `&#2211;` in place of the  $\Sigma_M$  character ensures that the correct Unicode code point is used for the comparison, regardless of the database code page.

## CURRENT CLIENT\_ACCTNG

The `CURRENT CLIENT_ACCTNG` (or `CLIENT ACCTNG`) special register contains the value of the accounting string from the client information specified for this connection. The data type of the register is `VARCHAR(255)`. The default value of this register is an empty string.

The value of the accounting string can be changed by using the Set Client Information (`sqleseti`) API.

Note that the value provided via the `sqleseti` API is in the application code page, and the special register value is stored in the database code page. Depending on the data values used when setting the client information, truncation of the data value stored in the special register may occur during code page conversion.

*Example:* Get the current value of the accounting string for this connection.

```
VALUES (CURRENT CLIENT_ACCTNG)
INTO :ACCT_STRING
```

## CURRENT DATE

The `CURRENT DATE` (or `CURRENT_DATE`) special register specifies a date that is based on a reading of the time-of-day clock when the SQL statement is executed at the application server. If this special register is used more than once within a single SQL statement, or used with `CURRENT TIME` or `CURRENT TIMESTAMP` within a single statement, all values are based on a single clock reading.

When used in an SQL statement inside a routine, `CURRENT DATE` is not inherited from the invoking statement.

In a federated system, `CURRENT DATE` can be used in a query intended for data sources. When the query is processed, the date returned will be obtained from the `CURRENT DATE` register at the federated server, not from the data sources.

*Example:* Run the following command from the DB2 CLP to obtain the current date.

```
db2 values CURRENT DATE
```

*Example:* Using the `PROJECT` table, set the project end date (`PRENDATE`) of the `MA2111` project (`PROJNO`) to the current date.

```
UPDATE PROJECT
SET PRENDATE = CURRENT DATE
WHERE PROJNO = 'MA2111'
```

## CURRENT DECFLOAT ROUNDING MODE

The `CURRENT DECFLOAT ROUNDING MODE` special register specifies the rounding mode that is used for `DECFLOAT` values.

The data type is VARCHAR(128). The following rounding modes are supported:

- ROUND\_CEILING rounds the value towards positive infinity. If all of the discarded digits are zero or if the sign is negative, the result is unchanged (except for the removal of the discarded digits). Otherwise, the result coefficient is incremented by 1.
- ROUND\_DOWN rounds the value towards 0 (truncation). The discarded digits are ignored.
- ROUND\_FLOOR rounds the value towards negative infinity. If all of the discarded digits are zero or if the sign is positive, the result is unchanged (except for the removal of the discarded digits). Otherwise, the sign is negative and the result coefficient is incremented by 1.
- ROUND\_HALF\_EVEN rounds the value to the nearest value. If the values are equidistant, rounds the value so that the final digit is even. If the discarded digits represent more than half of the value of a number in the next left position, the result coefficient is incremented by 1. If they represent less than half, the result coefficient is not adjusted (that is, the discarded digits are ignored). Otherwise, the result coefficient is unaltered if its rightmost digit is even, or incremented by 1 if its rightmost digit is odd (to make an even digit).
- ROUND\_HALF\_UP rounds the value to the nearest value. If the values are equidistant, rounds the value up. If the discarded digits represent half or more than half of the value of a number in the next left position, the result coefficient is incremented by 1. Otherwise, the discarded digits are ignored.

The value of the DECFLOAT rounding mode on a client can be confirmed to match that of the server by invoking the SET CURRENT DECFLOAT ROUNDING MODE statement. However, this statement cannot be used to change the rounding mode of the server. The initial value of CURRENT DECFLOAT ROUNDING MODE is determined by the **decflt\_rounding** database configuration parameter and can only be changed by changing the value of this database configuration parameter.

## CURRENT DEFAULT TRANSFORM GROUP

The CURRENT DEFAULT TRANSFORM GROUP special register specifies a VARCHAR(18) value that identifies the name of the transform group used by dynamic SQL statements for exchanging user-defined structured type values with host programs. This special register does not specify the transform groups used in static SQL statements, or in the exchange of parameters and results with external functions or methods.

Its value can be set by the SET CURRENT DEFAULT TRANSFORM GROUP statement. If no value is set, the initial value of the special register is the empty string (a VARCHAR with a length of zero).

In a dynamic SQL statement (that is, one which interacts with host variables), the name of the transform group used for exchanging values is the same as the value of this special register, unless this register contains the empty string. If the register contains the empty string (no value was set by using the SET CURRENT DEFAULT TRANSFORM GROUP statement), the DB2\_PROGRAM transform group is used for the transform. If the DB2\_PROGRAM transform group is not defined for the structured type subject, an error is raised at run time (SQLSTATE 42741).

*Examples:*



Set the default transform group to MYSTRUCT1. The TO SQL and FROM SQL functions defined in the MYSTRUCT1 transform are used to exchange user-defined structured type variables with the host program.

```
SET CURRENT DEFAULT TRANSFORM GROUP = MYSTRUCT1
```

Retrieve the name of the default transform group assigned to this special register.

```
VALUES (CURRENT DEFAULT TRANSFORM GROUP)
```

## CURRENT DEGREE

The CURRENT DEGREE special register specifies the degree of intra-partition parallelism for the execution of dynamic SQL statements. (For static SQL, the DEGREE bind option provides the same control.) The data type of the register is CHAR(5). Valid values are ANY or the string representation of an integer between 1 and 32 767, inclusive.

If the value of CURRENT DEGREE represented as an integer is 1 when an SQL statement is dynamically prepared, the execution of that statement will not use intra-partition parallelism.

If the value of CURRENT DEGREE represented as an integer is greater than 1 and less than or equal to 32 767 when an SQL statement is dynamically prepared, the execution of that statement can involve intra-partition parallelism with the specified degree.

If the value of CURRENT DEGREE is ANY when an SQL statement is dynamically prepared, the execution of that statement can involve intra-partition parallelism using a degree determined by the database manager.

The actual runtime degree of parallelism will be the lower of:

- The value of the maximum query degree (**max\_querydegree**) configuration parameter
- The application runtime degree
- The SQL statement compilation degree.

If the **intra\_parallel** database manager configuration parameter is set to NO, the value of the CURRENT DEGREE special register will be ignored for the purpose of optimization, and the statement will not use intra-partition parallelism.

The value can be changed by invoking the SET CURRENT DEGREE statement.

The initial value of CURRENT DEGREE is determined by the **dft\_degree** database configuration parameter.

## CURRENT EXPLAIN MODE

The CURRENT EXPLAIN MODE special register holds a VARCHAR(254) value which controls the behavior of the Explain facility with respect to eligible dynamic SQL statements. This facility generates and inserts Explain information into the Explain tables. This information does not include the Explain snapshot. Possible values are YES, EXPLAIN, NO, REOPT, RECOMMEND INDEXES, and EVALUATE INDEXES. (For static SQL, the EXPLAIN bind option provides the same control. In the case of the PREP and BIND commands, the EXPLAIN option values are: YES, NO, and ALL.)



**YES** Enables the Explain facility and causes Explain information for a dynamic SQL statement to be captured when the statement is compiled.

**EXPLAIN**

Enables the facility, but dynamic statements are not executed.

**NO** Disables the Explain facility.

**REOPT**

Enables the Explain facility and causes Explain information for a dynamic (or incremental-bind) SQL statement to be captured only when the statement is reoptimized using real values for the input variables (host variables, special registers, global variables, or parameter markers).

**RECOMMEND INDEXES**

Recommends a set of indexes for each dynamic query. Populates the ADVISE\_INDEX table with the set of indexes.

**EVALUATE INDEXES**

Explains dynamic queries as though the recommended indexes existed. The indexes are picked up from the ADVISE\_INDEX table.

The initial value is NO. The value can be changed by invoking the SET CURRENT EXPLAIN MODE statement.

The CURRENT EXPLAIN MODE and CURRENT EXPLAIN SNAPSHOT special register values interact when the Explain facility is invoked. The CURRENT EXPLAIN MODE special register also interacts with the EXPLAIN bind option. RECOMMEND INDEXES and EVALUATE INDEXES can only be set for the CURRENT EXPLAIN MODE register, and must be set using the SET CURRENT EXPLAIN MODE statement.

*Example:* Set the host variable EXPL\_MODE (VARCHAR(254)) to the value currently in the CURRENT EXPLAIN MODE special register.

```
VALUES CURRENT EXPLAIN MODE
INTO :EXPL_MODE
```

## CURRENT EXPLAIN SNAPSHOT

The CURRENT EXPLAIN SNAPSHOT special register holds a CHAR(8) value that controls the behavior of the Explain snapshot facility. This facility generates compressed information, including access plan information, operator costs, and bind-time statistics.

Only the following statements consider the value of this register: CALL, Compound SQL (Dynamic), DELETE, INSERT, MERGE, REFRESH, SELECT, SELECT INTO, SET INTEGRITY, UPDATE, VALUES, or VALUES INTO. Possible values are YES, EXPLAIN, NO, and REOPT. (For static SQL, the EXPLSNAP bind option provides the same control. In the case of the PREP and BIND commands, the EXPLSNAP option values are: YES, NO, and ALL.)

**YES** Enables the Explain snapshot facility and takes a snapshot of the internal representation of a dynamic SQL statement as the statement is compiled.

**EXPLAIN**

Enables the Explain snapshot facility, but dynamic statements are not executed.

**NO** Disables the Explain snapshot facility.

## REOPT

Enables the Explain facility and causes Explain information for a dynamic (or incremental-bind) SQL statement to be captured only when the statement is reoptimized using real values for the input variables (host variables, special registers, global variables, or parameter markers).

The initial value is NO. The value can be changed by invoking the SET CURRENT EXPLAIN SNAPSHOT statement.

The CURRENT EXPLAIN SNAPSHOT and CURRENT EXPLAIN MODE special register values interact when the Explain facility is invoked. The CURRENT EXPLAIN SNAPSHOT special register also interacts with the EXPLSNAP bind option.

*Example:* Set the host variable EXPL\_SNAP (char(8)) to the value currently in the CURRENT EXPLAIN SNAPSHOT special register.

```
VALUES CURRENT EXPLAIN SNAPSHOT
INTO :EXPL_SNAP
```

## CURRENT FEDERATED ASYNCHRONY

The CURRENT FEDERATED ASYNCHRONY special register specifies the degree of asynchrony for the execution of dynamic SQL statements. (The FEDERATED\_ASYNCHRONY bind option provides the same control for static SQL.) The data type of the register is INTEGER. Valid values are ANY (representing -1) or an integer between 0 and 32 767, inclusive. If, when an SQL statement is dynamically prepared, the value of CURRENT FEDERATED ASYNCHRONY is:

- 0, the execution of that statement will not use asynchrony
- Greater than 0 and less than or equal to 32 767, the execution of that statement can involve asynchrony using the specified degree
- ANY (representing -1), the execution of that statement can involve asynchrony using a degree that is determined by the database manager

The value of the CURRENT FEDERATED ASYNCHRONY special register can be changed by invoking the SET CURRENT FEDERATED ASYNCHRONY statement.

The initial value of the CURRENT FEDERATED ASYNCHRONY special register is determined by the **federated\_async** database manager configuration parameter if the dynamic statement is issued through the command line processor (CLP). The initial value is determined by the FEDERATED\_ASYNCHRONY bind option if the dynamic statement is part of an application that is being bound.

*Example:* Set the host variable FEDASYNC (INTEGER) to the value of the CURRENT FEDERATED ASYNCHRONY special register.

```
VALUES CURRENT FEDERATED ASYNCHRONY INTO :FEDASYNC
```

## CURRENT IMPLICIT XMLPARSE OPTION

The CURRENT IMPLICIT XMLPARSE OPTION special register specifies the whitespace handling options that are to be used when serialized XML data is implicitly parsed by the DB2 server, without validation. An implicit non-validating parse operation occurs when an SQL statement is processing an XML host variable

or an implicitly or explicitly typed XML parameter marker that is not an argument of the XMLVALIDATE function. The data type of the register is VARCHAR(19).

The value of the CURRENT IMPLICIT XMLPARSE OPTION special register can be changed by invoking the SET CURRENT IMPLICIT XMLPARSE OPTION statement. Its initial value is 'STRIP WHITESPACE'.

*Examples:*

Retrieve the value of the CURRENT IMPLICIT XMLPARSE OPTION special register into a host variable named CURXMLPARSEOPT:

```
EXEC SQL VALUES (CURRENT IMPLICIT XMLPARSE OPTION) INTO :CURXMLPARSEOPT;
```

Set the CURRENT IMPLICIT XMLPARSE OPTION special register to 'PRESERVE WHITESPACE'.

```
SET CURRENT IMPLICIT XMLPARSE OPTION = 'PRESERVE WHITESPACE'
```

Whitespace is then preserved when the following SQL statement executes:

```
INSERT INTO T1 (XMLCOL1) VALUES (?)
```

## CURRENT ISOLATION

The CURRENT ISOLATION special register holds a CHAR(2) value that identifies the isolation level (in relation to other concurrent sessions) for any dynamic SQL statements issued within the current session.

The possible values are:

**(blanks)**

Not set; use the isolation attribute of the package.

**UR** Uncommitted Read

**CS** Cursor Stability

**RR** Repeatable Read

**RS** Read Stability

The value of the CURRENT ISOLATION special register can be changed by the SET CURRENT ISOLATION statement.

Until a SET CURRENT ISOLATION statement is issued within a session, or after RESET has been specified for SET CURRENT ISOLATION, the CURRENT ISOLATION special register is set to blanks and is not applied to dynamic SQL statements; the isolation level used is taken from the isolation attribute of the package which issued the dynamic SQL statement. Once a SET CURRENT ISOLATION statement has been issued, the CURRENT ISOLATION special register provides the isolation level for any subsequent dynamic SQL statement compiled within the session, regardless of the settings for the package issuing the statement. This will remain in effect until the session ends or until a SET CURRENT ISOLATION statement is issued with the RESET option.

*Example:* Set the host variable ISOLATION\_MODE (CHAR(2)) to the value currently stored in the CURRENT ISOLATION special register.

```
VALUES CURRENT ISOLATION
INTO :ISOLATION_MODE
```

## CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION

The CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register specifies a VARCHAR(254) value that identifies the types of tables that can be considered when optimizing the processing of dynamic SQL queries. Materialized query tables are never considered by static embedded SQL queries.

The initial value of CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION is SYSTEM. Its value can be changed by the SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION statement.

## CURRENT MDC ROLLOUT MODE

The CURRENT MDC ROLLOUT MODE special register specifies the behavior on multidimensional clustering (MDC) tables of DELETE statements that qualify for rollout processing.

The default value of this register is determined by the DB2\_MDC\_ROLLOUT registry variable. The value can be changed by invoking the SET CURRENT MDC ROLLOUT MODE statement. When the CURRENT MDC ROLLOUT MODE special register is set to a particular value, the execution behavior of subsequent DELETE statements that qualify for rollout is impacted. The DELETE statement does not need to be recompiled for the behavior to change.

## CURRENT OPTIMIZATION PROFILE

The CURRENT OPTIMIZATION PROFILE special register specifies the qualified name of the optimization profile to be used by DML statements that are dynamically prepared for optimization.

The initial value is the null value. The value can be changed by invoking the SET CURRENT OPTIMIZATION PROFILE statement. An optimization profile that is not qualified with a schema name will be implicitly qualified with the value of the CURRENT DEFAULT SCHEMA special register.

*Example 1:* Set the optimization profile to 'JON.SALES'.

```
SET CURRENT OPTIMIZATION PROFILE = JON.SALES
```

*Example 2:* Get the current value of the optimization profile name for this connection.

```
VALUES (CURRENT OPTIMIZATION PROFILE) INTO :PROFILE
```

## CURRENT PACKAGE PATH

The CURRENT PACKAGE PATH special register specifies a VARCHAR(4096) value that identifies the path to be used when resolving references to packages that are needed when executing SQL statements.

The value can be an empty or a blank string, or a list of one or more schema names that are delimited with double quotation marks and separated by commas. Any double quotation marks appearing as part of the string will need to be represented as two double quotation marks, as is common practice with delimited identifiers. The delimiters and commas contribute to the length of the special register.

This special register applies to both static and dynamic statements.

The initial value of CURRENT PACKAGE PATH in a user-defined function, method, or procedure is inherited from the invoking application. In other contexts, the initial value of CURRENT PACKAGE PATH is an empty string. The value is a list of schemas only if the application process has explicitly specified a list of schemas by means of the SET CURRENT PACKAGE PATH statement.

*Examples:*

An application will be using multiple SQLJ packages (in schemas SQLJ1 and SQLJ2) and a JDBC package (in schema DB2JAVA). Set the CURRENT PACKAGE PATH special register to check SQLJ1, SQLJ2, and DB2JAVA, in that order.

```
SET CURRENT PACKAGE PATH = "SQLJ1", "SQLJ2", "DB2JAVA"
```

Set the host variable HVPKLIST to the value currently stored in the CURRENT PACKAGE PATH special register.

```
VALUES CURRENT PACKAGE PATH INTO :HVPKLIST
```

## CURRENT PATH

The CURRENT PATH (or CURRENT\_PATH) special register specifies a VARCHAR(2048) value that identifies the SQL path used when resolving unqualified function names, procedure names, data type names, global variable names, and module object names in dynamically prepared SQL statements. CURRENT FUNCTION PATH is a synonym for CURRENT PATH. The initial value is the default value specified below. For static SQL, the FUNCPATH bind option provides an SQL path that is used for function and data type resolution.

The CURRENT PATH special register contains a list of one or more schema names that are enclosed by double quotation marks and separated by commas. For example, an SQL path specifying that the database manager is to look first in the FERMAT schema, then in the XGRAPHIC schema, and finally in the SYSIBM schema, is returned in the CURRENT PATH special register as:

```
"FERMAT", "XGRAPHIC", "SYSIBM"
```

The default value is "SYSIBM", "SYSFUN", "SYSPROC", "SYSIBMADM", X, where X is the value of the USER special register, delimited by double quotation marks. The value can be changed by invoking the SET CURRENT PATH statement. The schema SYSIBM does not need to be specified. If it is not included in the SQL path, it is implicitly assumed to be the first schema. SYSIBM does not take up any of the 2048 bytes if it is implicitly assumed.

A data type that is not qualified with a schema name will be implicitly qualified with the first schema in the SQL path that contains a data type with the same unqualified name. There are exceptions to this rule, as outlined in the descriptions of the following statements: CREATE TYPE (Distinct), CREATE FUNCTION, COMMENT, and DROP.

*Example:* Using the SYSCAT.ROUTINES catalog view, find all user-defined routines that can be invoked without qualifying the routine name, because the CURRENT PATH special register contains the schema name.

```
SELECT ROUTINENAME, ROUTINESCHEMA FROM SYSCAT.ROUTINES
WHERE POSITION (ROUTINESCHEMA, CURRENT PATH, CODEUNITS16) <> 0
```

## CURRENT QUERY OPTIMIZATION

The CURRENT QUERY OPTIMIZATION special register specifies an INTEGER value that controls the class of query optimization performed by the database manager when binding dynamic SQL statements. The QUERYOPT bind option controls the class of query optimization for static SQL statements. The possible values range from 0 to 9. For example, if the query optimization class is set to 0 (minimal optimization), then the value in the special register is 0. The default value is determined by the `dft_queryopt` database configuration parameter. The value can be changed by invoking the SET CURRENT QUERY OPTIMIZATION statement.

*Example:* Using the SYSCAT.PACKAGES catalog view, find all plans that were bound with the same setting as the current value of the CURRENT QUERY OPTIMIZATION special register.

```
SELECT PKGNAME, PKGSCHEMA FROM SYSCAT.PACKAGES
WHERE QUERYOPT = CURRENT QUERY OPTIMIZATION
```

## CURRENT REFRESH AGE

The CURRENT REFRESH AGE special register specifies a timestamp duration value with a data type of DECIMAL(20,6). It is the maximum duration since a particular timestamped event occurred to a cached data object (for example, a REFRESH TABLE statement processed on a system-maintained REFRESH DEFERRED materialized query table), such that the cached data object can be used to optimize the processing of a query. If CURRENT REFRESH AGE has a value of 99 999 999 999 999, and the query optimization class is 5 or more, the types of tables specified in CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION are considered when optimizing the processing of a dynamic SQL query.

The value of CURRENT REFRESH AGE must be 0 or 99 999 999 999 999. The initial value is 0. The value can be changed by invoking the SET CURRENT REFRESH AGE statement.

## CURRENT TIME

The CURRENT TIME (or CURRENT\_TIME) special register specifies a time that is based on a reading of the time-of-day clock when the SQL statement is executed at the application server. If this special register is used more than once within a single SQL statement, or used with CURRENT DATE or CURRENT TIMESTAMP within a single statement, all values are based on a single clock reading.

When used in an SQL statement inside a routine, CURRENT TIME is not inherited from the invoking statement.

In a federated system, CURRENT TIME can be used in a query intended for data sources. When the query is processed, the time returned will be obtained from the CURRENT TIME register at the federated server, not from the data sources.

*Example:* Run the following command from the DB2 CLP to obtain the current time.

```
db2 values CURRENT TIME
```

*Example:* Using the CL\_SCHED table, select all the classes (CLASS\_CODE) that start (STARTING) later today. Today's classes have a value of 3 in the DAY column.



```
SELECT CLASS_CODE FROM CL_SCHED
WHERE STARTING > CURRENT TIME AND DAY = 3
```

## CURRENT TIMESTAMP

The CURRENT TIMESTAMP (or CURRENT\_TIMESTAMP) special register specifies a timestamp that is based on a reading of the time-of-day clock when the SQL statement is executed at the application server. If this special register is used more than once within a single SQL statement, or used with CURRENT DATE or CURRENT TIME within a single statement, all values are based on a single clock reading. It is possible for separate CURRENT TIMESTAMP special register requests to return the same value; if unique values are required, consider using the GENERATE\_UNIQUE function, a sequence, or an identity column.

If a timestamp with a specific precision is desired, the special register can be referenced as CURRENT\_TIMESTAMP(*integer*), where *integer* can range from 0 to 12. The default precision is 6. The precision of the clock reading varies by platform and the resulting value is padded with zeros where the precision of the retrieved clock reading is less than the precision of the request.

When used in an SQL statement inside a routine, CURRENT\_TIMESTAMP is not inherited from the invoking statement.

In a federated system, CURRENT\_TIMESTAMP can be used in a query intended for data sources. When the query is processed, the timestamp returned will be obtained from the CURRENT\_TIMESTAMP register at the federated server, not from the data sources.

SYSDATE can also be specified as a synonym for CURRENT\_TIMESTAMP(0).

*Example:* Insert a row into the IN\_TRAY table. The value of the RECEIVED column should be a timestamp that indicates when the row was inserted. The values for the other three columns come from the host variables SRC (char(8)), SUB (char(64)), and TXT (VARCHAR(200)).

```
INSERT INTO IN_TRAY
VALUES (CURRENT_TIMESTAMP, :SRC, :SUB, :TXT)
```

## CURRENT TIMEZONE

The CURRENT TIMEZONE (or CURRENT\_TIMEZONE) special register specifies the difference between UTC (Coordinated Universal Time, formerly known as GMT) and local time at the application server. The difference is represented by a time duration (a decimal number in which the first two digits are the number of hours, the next two digits are the number of minutes, and the last two digits are the number of seconds). The number of hours is between -24 and 24 exclusive. Subtracting CURRENT\_TIMEZONE from a local time converts that local time to UTC. The time is calculated from the operating system time at the moment the SQL statement is executed. (The CURRENT\_TIMEZONE value is determined from C runtime functions.)

The CURRENT\_TIMEZONE special register can be used wherever an expression of the DECIMAL(6,0) data type is used; for example, in time and timestamp arithmetic.

When used in an SQL statement inside a routine, CURRENT\_TIMEZONE is not inherited from the invoking statement.

*Example:* Insert a record into the IN\_TRAY table, using a UTC timestamp for the RECEIVED column.

```
INSERT INTO IN_TRAY VALUES (
 CURRENT_TIMESTAMP - CURRENT_TIMEZONE,
 :source,
 :subject,
 :notetext)
```

## CURRENT USER

The CURRENT USER (or CURRENT\_USER) special register specifies the authorization ID that is to be used for statement authorization. For static SQL statements, the value represents the authorization ID that is used when the package is bound. For dynamic SQL statements, the value is the same as the value of the SESSION\_USER special register for packages bound with the DYNAMICRULES(RUN) bind option. The data type of the register is VARCHAR(128).

*Example:* Select table names whose schema matches the value of the CURRENT USER special register.

```
SELECT TABNAME FROM SYSCAT.TABLES
WHERE TABSCHEMA = CURRENT_USER AND TYPE = 'T'
```

If this statement is executed as a static SQL statement, it returns the tables whose schema name matches the binder of the package that includes the statement. If this statement is executed as a dynamic SQL statement, it returns the tables whose schema name matches the current value of the SESSION\_USER special register.

---

## Configuration parameter topics

### agent\_stack\_sz - Agent stack size

This parameter determines the virtual memory that is allocated by DB2 for each agent.

#### Configuration type

Database manager

#### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

#### Parameter type

Configurable

#### Default [range]

##### Linux (32-bit)

256 [16 – 1024]

##### Linux (64-bit) and UNIX

1024 [256 – 32768]

##### Windows

16 [8 – 1000]

#### Unit of measure

Pages (4 KB)



**When allocated**

When an agent is initialized to do work for an application

**When freed**

When an agent completes the work to be done for an application

You can use this parameter to optimize memory utilization of the server for a given set of applications. More complex queries will use more stack space, compared to the space used for simple queries.

This parameter is used to set the initial committed stack size for each agent in a Windows environment. By default, each agent stack can grow up to the default reserve stack size of 256 KB (64 4-KB pages). This limit is sufficient for most database operations. On UNIX and Linux, *agent\_stack\_sz* will be rounded up to the next larger power-of-2 based value. The default setting for UNIX should be sufficient for most workloads

However, when preparing a large SQL or XQuery statement, the agent can run out of stack space and the system will generate a stack overflow exception (0xC00000FD). When this happens, the server will shut down because the error is non-recoverable.

**Note:** In Version 9.5 and later, sqlcode -973 will be returned instead of a stack overflow exception..

The agent stack size can be increased by setting *agent\_stack\_sz* to a value larger than the default reserve stack size of 64 pages. Note that the value for *agent\_stack\_sz*, when larger than the default reserve stack size, is rounded by the Windows operating system to the nearest multiple of 1 MB; setting the agent stack size to 128 4-KB pages actually reserves a 1 MB stack for each agent. Setting the value for *agent\_stack\_sz* less than the default reserve stack size will have no effect on the maximum limit because the stack still grows if necessary up to the default reserve stack size. In this case, the value for *agent\_stack\_sz* is the initial committed memory for the stack when an agent is created.

You can change the default reserve stack size by using the db2hdr utility to change the header information for the db2syscs.exe file. Changing the default reserve stack size will affect all threads while changing *agent\_stack\_sz* only affects the stack size for agents. The advantage of changing the default stack size using the db2hdr utility is that it provides a better granularity, therefore allowing the stack size to be set at the minimum required stack size. However, you will have to stop and restart DB2 for a change to db2syscs.exe to take effect.

**Recommendation:** If you will be working with large or complex XML data in a 32-bit environment, you should update *agent\_stack\_sz* to at least 256 4-KB pages. Very complex XML schemas might require *agent\_stack\_sz* to be set much closer to the limit in order to avoid stack overflow exceptions during schema registration or during XML document validation.

You might be able to reduce the stack size in order to make more address space available to other clients, if your environment matches the following:

- Contains only simple applications (for example light OLTP), in which there are never complex queries
- Requires a relatively large number of concurrent clients (for example, more than 100).

On Windows, the agent stack size and the number of concurrent clients are inversely related: a larger stack size reduces the potential number of concurrent clients that can be running. This occurs because address space is limited on Windows platforms.

## agentpri - Priority of agents

This parameter is deprecated in Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will continue to work exactly as it did in previous versions, and this parameter will continue to be fully supported. If this parameter is used for workload management (WLM), then the WLM service class agent priority will be ignored.

**Note:** The following information applies only to pre-Version 9.5 data servers and clients.

This parameter controls the priority given both to all agents, and to other database manager instance processes and threads, by the operating system scheduler. This priority determines how CPU time is given to the database manager processes, agents, and threads relative to the other processes and threads running on the machine.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default [range]

**AIX** -1 (system) [ 41 - 125 ]

### Other UNIX

-1 (system) [ 41 - 128 ]

### Windows

-1 (system) [ 0 - 6 ]

### Solaris

-1 (system) [ 0 - 59 ]

When the parameter is set to -1 or system, no special action is taken and the database manager is scheduled in the normal way that the operating system schedules all processes and threads. When the parameter is set to a value other than -1 or system, the database manager will create its processes and threads with a static priority set to the value of the parameter. Therefore, this parameter allows you to control the priority with which the database manager processes and threads (in a partitioned database environment, this also includes coordinating and subagents, the parallel system controllers, and the FCM daemons) will execute on your machine.

You can use this parameter to increase database manager throughput. The values for setting this parameter are dependent on the operating system on which the database manager is running. For example, in a Linux or UNIX environment, numerically low values yield high priorities. When the parameter is set to a value

between 41 and 125, the database manager creates its agents with a UNIX static priority set to the value of the parameter. This is important in Linux and UNIX environments because numerically low values yield high priorities for the database manager, but other processes (including applications and users) might experience delays because they cannot obtain enough CPU time. You should balance the setting of this parameter with the other activity expected on the machine.

**Restrictions:**

- If you set this parameter to a non-default value on Linux and UNIX platforms, you cannot use the governor to alter agent priorities.
- On the Solaris operating system, you should not change the default value (-1). Changing the default value sets the priority of DB2 processes to real-time, which can monopolize all available resources on the system.

**Recommendation:** The default value should be used initially. This value provides a good compromise between response time to other users/applications and database manager throughput.

If database performance is a concern, you can use benchmarking techniques to determine the optimum setting for this parameter. You should take care when increasing the priority of the database manager because performance of other user processes can be severely degraded, especially when the CPU utilization is very high. Increasing the priority of the database manager processes and threads can have significant performance benefits.

## **alt\_collate - Alternate collating sequence**

This parameter specifies the collating sequence that is to be used for Unicode tables in a non-Unicode database.

**Configuration type**

Database

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

Null [IDENTITY\_16BIT ]

Until this parameter is set, Unicode tables and routines cannot be created in a non-Unicode database. Once set, this parameter cannot be changed or reset.

This parameter cannot be set for Unicode databases.

## **alternate\_auth\_enc - Alternate encryption algorithm for incoming connections at server configuration parameter**

This configuration parameter specifies the alternate encryption algorithm used to encrypt the user IDs and passwords submitted to a DB2 database server for

authentication. Specifically, this parameter affects the encryption algorithm when the authentication method negotiated between the DB2 client and the DB2 database server is `SERVER_ENCRYPT`.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

`NOT_SPECIFIED` [`AES_CMP`; `AES_ONLY`]

The user ID and password submitted for authentication on the DB2 database server are encrypted when the authentication method negotiated between the DB2 client and the DB2 server is `SERVER_ENCRYPT`. The authentication method negotiated depends on the authentication type setting on the server and the authentication type requested by the client. The choice of the encryption algorithm used to encrypt the user ID and password depends on the setting of the **`alternate_auth_enc`** database manager configuration parameter. It can be either DES or AES depending on this setting.

When the default (`NOT_SPECIFIED`) value is used, the database server accepts the encryption algorithm that the client proposes.

When **`alternate_auth_enc`** is set to `AES_ONLY`, the database server will only accept connections that use AES encryption. If the client does not support AES encryption, then the connection is rejected.

When **`alternate_auth_enc`** is set to `AES_CMP`, the database server will accept user IDs and passwords that are encrypted using either AES or DES, but it will negotiate for AES if the client supports AES encryption.

## **appl\_memory - Application Memory configuration parameter**

This parameter allows DBAs and ISVs to control the maximum amount of application memory that is allocated by DB2 database agents to service application requests. By default, its value is set to `AUTOMATIC`, meaning that all application memory requests will be allowed as long as the total amount of memory allocated by the database partition is within the *instance\_memory* limits.

**Configuration type**

Database

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable online

**Default [range]**

Automatic [128 - 4 294 967 295]

**Unit of measure**

Pages (4 KB)

**When allocated**

During database activation

**When freed**

During database deactivation

**Note:** When *appl\_memory* is set to AUTOMATIC, the initial application memory allocation at database activation time is minimal, and increases (or decreases) as needed. The change is applied in memory and the value of *appl\_memory* does not change on disk as shown by `db2 get db cfg show detail`. On next activation, the value will be recalculated. If *appl\_memory* is set to a specific value, then the requested amount of memory is allocated initially during database activation, and the application memory size does not change. If the initial amount of application memory cannot be allocated from the operating system, or exceeds the *instance\_memory* limit, database activation fails with an SQL1084C error (Shared memory segments cannot be allocated).

## **applheapsz - Application heap size**

In previous releases, the *applheapsz* database configuration parameter referred to the amount of application memory each individual database agent working for that application could consume. With Version 9.5, *applheapsz* refers to the total amount of application memory that can be consumed by the entire application. For DPF, Concentrator, or SMP configurations, this means that the *applheapsz* value used in previous releases may need to be increased under similar workloads, unless the AUTOMATIC setting is used.

With Version 9.5, this database configuration parameter has a default value of AUTOMATIC, meaning that it increases as needed until either the *appl\_memory* limit is reached, or the *instance\_memory* limit is reached.

**Configuration type**

Database

**Parameter type**

Configurable online

**Default [range]**

Automatic [16 - 60 000]

**Unit of measure**

Pages (4 KB)

**When allocated**

When an application associates with, or connects to, a database.

**When freed**

When the application disassociates or disconnects from the database.

**Note:** This parameter defines the maximum size of the application heap. One application heap is allocated per database application when the application first connects with the database. The heap is shared by all database agents working for that application. (In previous releases, each database agent allocated its own application heap.) Memory is allocated from the application heap as needed to process the application, up to the limit specified by this parameter. When set to AUTOMATIC, the application heap is allowed to grow as needed up to either the

*appl\_memory* limit for the database, or the *instance\_memory* limit for the database partition. The entire application heap is freed when the application disconnects with the database.

The online changed value takes effect at an application connection boundary, that is, after it is changed dynamically, currently connected applications still use the old value, but all newly connected applications will use the new value.

## **archretrydelay - Archive retry delay on error**

This parameter specifies the number of seconds to wait after a failed archive attempt before trying to archive the log file again.

### **Configuration type**

Database

### **Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable Online

### **Default [range]**

20 [0 - 65 535 ]

Subsequent retries will only take affect if the value of the *numarchretry* database configuration parameter is at least 1.

## **aslheapsz - Application support layer heap size**

The application support layer heap represents a communication buffer between the local application and its associated agent. This buffer is allocated as shared memory by each database manager agent that is started.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable

### **Default [range]**

15 [1 - 524 288]

### **Unit of measure**

Pages (4 KB)

### **When allocated**

When the database manager agent process is started for the local application

### **When freed**

When the database manager agent process is terminated

If the request to the database manager, or its associated reply, do not fit into the buffer they will be split into two or more send-and-receive pairs. The size of this buffer should be set to handle the majority of requests using a single send-and-receive pair. The size of the request is based on the storage required to hold:

- The input SQLDA
- All of the associated data in the SQLVARs
- The output SQLDA
- Other fields which do not generally exceed 250 bytes.

In addition to this communication buffer, this parameter is also used for two other purposes:

- It is used to determine the I/O block size when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the Data Server Runtime Client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.
- It is used to determine the communication size between agents and db2fmp processes. (A db2fmp process can be a user-defined function or a fenced stored procedure.) The number of bytes is allocated from shared memory for each db2fmp process or thread that is active on the system.

The data sent from the local application is received by the database manager into a set of contiguous memory allocated from the query heap. The *aslheapsz* parameter is used to determine the initial size of the query heap (for both local and remote clients). The maximum size of the query heap is defined by the *query\_heap\_sz* parameter.

**Recommendation:** If your application's requests are generally small and the application is running on a memory constrained system, you might want to reduce the value of this parameter. If your queries are generally very large, requiring more than one send and receive request, and your system is not constrained by memory, you might want to increase the value of this parameter.

Use the following formula to calculate a minimum number of pages for *aslheapsz*:

```
aslheapsz >= (sizeof(input SQLDA)
 + sizeof(each input SQLVAR)
 + sizeof(output SQLDA)
 + 250) / 4096
```

where `sizeof(x)` is the size of `x` in bytes that calculates the number of pages of a given input or output value.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks might yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks might also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the `OPTIMIZE FOR` clause on the `SELECT` statement in your application.



## auto\_del\_rec\_obj - Automated deletion of recovery objects configuration parameter

This parameter specifies whether database log files, backup images, and load copy images should be deleted when their associated recovery history file entry is pruned.

### Configuration type

Database

### Parameter type

Configurable online

### Propagation class

Immediate

### Default [range]

OFF [ON; OFF ]

You can prune the entries in the recovery history file using the PRUNE HISTORY command or the db2Prune API. You can also configure the IBM Data Server database manager to automatically prune the recovery history file after each full database backup. If you set the *auto\_del\_rec\_obj* database configuration parameter to ON, then the database manager will also delete the corresponding physical log files, backup images, and load copy images when it prunes the history file. The database manager can only delete recovery objects such as database logs, backup images, and load copy images when your storage media is disk, or if you are using a storage manager, such as the Tivoli<sup>®</sup> Storage Manager.

## auto\_maint - Automatic maintenance

This parameter is the parent of all the other automatic maintenance database configuration parameters (*auto\_db\_backup*, *auto\_tbl\_maint*, *auto\_runstats*, *auto\_stats\_prof*, *auto\_stmt\_stats*, *auto\_prof\_upd*, and *auto\_reorg*).

### Configuration type

Database

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable Online

### Propagation class

Immediate

### Default [range]

ON [ON; OFF ]

When this parameter is disabled, all of its child parameters are also disabled, but their settings, as recorded in the database configuration file, do not change. When this parent parameter is enabled, recorded values for its child parameters take effect. In this way, automatic maintenance can be enabled or disabled globally.

By default, this parameter is set to ON.



You can enable or disable individual automatic maintenance features independently by setting the following parameters:

#### **auto\_db\_backup**

This automated maintenance parameter enables or disables automatic backup operations for a database. A backup policy (a defined set of rules or guidelines) can be used to specify the automated behavior. The objective of the backup policy is to ensure that the database is being backed up regularly. The backup policy for a database is created automatically when the DB2 Health Monitor first runs. By default, this parameter is set to OFF. To be enabled, this parameter must be set to ON, and its parent parameter must also be enabled.

#### **auto\_tbl\_maint**

This parameter is the parent of all table maintenance parameters (*auto\_runstats*, *auto\_stats\_prof*, *auto\_prof\_upd*, and *auto\_reorg*). When this parameter is disabled, all of its child parameters are also disabled, but their settings, as recorded in the database configuration file, do not change. When this parent parameter is enabled, recorded values for its child parameters take effect. In this way, table maintenance can be enabled or disabled globally.

By default, this parameter is set to ON.

#### **auto\_runstats**

This automated table maintenance parameter enables or disables automatic table runstats operations for a database. A runstats policy (a defined set of rules or guidelines) can be used to specify the automated behavior. Statistics collected by the runstats utility are used by the optimizer to determine the most efficient plan for accessing the physical data. To be enabled, this parameter must be set to On, and its parent parameters must also be enabled.

By default, this parameter is set to ON.

#### **auto\_stats\_prof**

When enabled, this automated table maintenance parameter turns on statistical profile generation, designed to improve applications whose workloads include complex queries, many predicates, joins, and grouping operations over several tables. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to OFF.

#### **auto\_stmt\_stats**

This parameter enables and disables the collection of real-time statistics. It is a child of the *auto\_runstats* configuration parameter. This feature is enabled only if the parent, *auto\_runstats* configuration parameter, is also enabled. For example, to enable *auto\_stmt\_stats*, set *auto\_maint*, *auto\_tbl\_maint*, and *auto\_runstats* to ON. To preserve the child value, the *auto\_runstats* configuration parameter can be ON while the *auto\_maint* configuration parameter is OFF. The corresponding Auto Runstats feature will still be OFF.

Assuming that both Auto Runstats and Auto Reorg are enabled, the settings are as follows:

|                             |                        |
|-----------------------------|------------------------|
| Automatic maintenance       | (AUTO_MAINT) = ON      |
| Automatic database backup   | (AUTO_DB_BACKUP) = OFF |
| Automatic table maintenance | (AUTO_TBL_MAINT) = ON  |
| Automatic runstats          | (AUTO_RUNSTATS) = ON   |

|                                       |                                |
|---------------------------------------|--------------------------------|
| <b>Automatic statement statistics</b> | <b>(AUTO_STMT_STATS) = OFF</b> |
| Automatic statistics profiling        | (AUTO_STATS_PROF) = OFF        |
| Automatic profile updates             | (AUTO_PROF_UPD) = OFF          |
| Automatic reorganization              | (AUTO_REORG) = ON              |

You can disable both Auto Runstats and Auto Reorg features temporarily by setting *auto\_tbl\_maint* to OFF. Both features can be enabled later by setting *auto\_tbl\_maint* back to ON. You do not need to issue db2stop or db2start commands to have the changes take effect.

By default, this parameter is set to OFF.

#### **auto\_prof\_upd**

When enabled, this automated table maintenance parameter (a child of *auto\_stats\_prof*) specifies that the runstats profile is to be updated with recommendations. When this parameter is disabled, recommendations are stored in the *opt\_feedback\_ranking* table, which you can inspect when manually updating the runstats profile. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to OFF.

#### **auto\_reorg**

This automated table maintenance parameter enables or disables automatic table and index reorganization for a database. A reorganization policy (a defined set of rules or guidelines) can be used to specify the automated behavior. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to OFF.

## **avg\_appls - Average number of active applications**

This parameter is used by the query optimizer to help estimate how much buffer pool will be available at run-time for the access plan chosen.

#### **Configuration type**

Database

#### **Parameter type**

Configurable Online

#### **Propagation class**

Statement boundary

#### **Default [range]**

Automatic [1 – maxappls ]

#### **Unit of measure**

Counter

**Recommendation:** When running DB2 in a multi-user environment, particularly with complex queries and a large buffer pool, you might want the query optimizer to know that multiple query users are using your system so that the optimizer should be more conservative in assumptions of buffer pool availability.

When setting this parameter, you should estimate the number of complex query applications that typically use the database. This estimate should exclude all light OLTP applications. If you have trouble estimating this number, you can multiply the following:

- An average number of all applications running against your database. The database system monitor can provide information about the number of

applications at any given time and using a sampling technique, you can calculate an average over a period of time. The information from the database system monitor includes both OLTP and non-OLTP applications.

- Your estimate of the percentage of complex query applications.

As with adjusting other configuration parameters that affect the optimizer, you should adjust this parameter in small increments. This allows you to minimize path selection differences.

You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

## **backup\_pending - Backup pending indicator**

This parameter indicates whether you need to do a full backup of the database before accessing it.

### **Configuration type**

Database

### **Parameter type**

Informational

This parameter is only on if the database configuration is changed so that the database moves from being nonrecoverable to recoverable (that is, initially both the *logretain* and *userexit* parameters were set to NO, then either one or both of these parameters is set to YES, and the update to the database configuration is accepted).

## **blk\_log\_dsk\_ful - Block on log disk full**

This parameter can be set to prevent disk full errors from being generated when DB2 cannot create a new log file in the active log path.

### **Configuration type**

Database

### **Parameter type**

Configurable Online

### **Propagation class**

Immediate

### **Default [range]**

No [Yes; No ]

Instead of generating a disk full error, DB2 will attempt to create the log file every five minutes until it succeeds. After each attempt, DB2 writes a message to the Administration Notification log. The only way that you can confirm that your application is hanging because of a log disk full condition is to monitor the Administration Notification log. Until the log file is successfully created, any user application that attempts to update table data will not be able to commit transactions. Read-only queries might not be directly affected; however, if a query needs to access data that is locked by an update request, or a data page that is fixed in the buffer pool by the updating application, read-only queries will also appear to hang.

Setting *blk\_log\_dsk\_ful* to *yes* causes applications to hang when DB2 encounters a log disk full error, thus allowing you to resolve the error and allowing the

transaction to complete. You can resolve a disk full situation by moving old log files to another file system or by enlarging the file system, so that hanging applications can complete.

If *blk\_log\_dsk\_ful* is set to *no*, then a transaction that receives a log disk full error will fail and will be rolled back. In some situations, the database will come down if a transaction causes a log disk full error.

## catalogcache\_sz - Catalog cache size

This parameter specifies the maximum space in pages that the catalog cache can use from the database heap.

### Configuration type

Database

### Parameter type

Configurable online

### Propagation class

Immediate

### Default [range]

-1 [MAXAPPLS\*5]

### Unit of measure

Pages (4 KB)

### When allocated

When the database is initialized

### When freed

When the database is shut down

This parameter is allocated out of the database shared memory, and is used to cache system catalog information. In a partitioned database system, there is one catalog cache for each database partition.

Caching catalog information at individual database partitions allows the database manager to reduce its internal overhead by eliminating the need to access the system catalogs (or the catalog node in a partitioned database environment) to obtain information that has previously been retrieved. The use of the catalog cache can help improve the overall performance of:

- Binding packages and compiling SQL and XQuery statements
- Operations that involve checking database-level privileges, routine privileges, global variable privileges and role authorizations
- Applications that are connected to non-catalog nodes in a partitioned database environment

By taking the default (-1) in a server or partitioned database environment, the value used to calculate the page allocation is five times the value specified for the *maxappls* configuration parameter. The exception to this occurs if five times *maxappls* is less than 8. In this situation, the default value of -1 will set *catalogcache\_sz* to 8.

**Recommendation:** Start with the default value and tune it by using the database system monitor. When tuning this parameter, you should consider whether the extra memory being reserved for the catalog cache might be more effective if it was allocated for another purpose, such as the buffer pool or package cache.

Tuning this parameter is particularly important if a workload involves many SQL or XQuery compilations for a brief period of time, with few or no compilations thereafter. If the cache is too large, memory might be wasted holding copies of information that will no longer be used.

In an partitioned database environment, consider if the *catalogcache\_sz* at the catalog node needs to be set larger since catalog information that is required at non-catalog nodes will always first be cached at the catalog node.

The *cat\_cache\_lookups* (catalog cache lookups), *cat\_cache\_inserts* (catalog cache inserts), *cat\_cache\_overflows* (catalog cache overflows), and *cat\_cache\_size\_top* (catalog cache high water mark) monitor elements can help you determine whether you should adjust this configuration parameter.

**Note:** The catalog cache exists on all nodes in a partitioned database environment. Since there is a local database configuration file for each node, each node's *catalogcache\_sz* value defines the size of the local catalog cache. In order to provide efficient caching and avoid overflow scenarios, you need to explicitly set the *catalogcache\_sz* value at each node and consider the feasibility of possibly setting the *catalogcache\_sz* on non-catalog nodes to be smaller than that of the catalog node; keep in mind that information that is required to be cached at non-catalog nodes will be retrieved from the catalog node's cache. Hence, a catalog cache at a non-catalog node is like a subset of the information in the catalog cache at the catalog node.

In general, more cache space is required if a unit of work contains several dynamic SQL or XQuery statements or if you are binding packages that contain a large number of static SQL or XQuery statements.

## **chngpgs\_thresh - Changed pages threshold**

This parameter specifies the level (percentage) of changed pages at which the asynchronous page cleaners will be started, if they are not currently active.

### **Configuration type**

Database

### **Parameter type**

Configurable

### **Default [range]**

60 [5 – 99 ]

### **Unit of measure**

Percentage

Asynchronous page cleaners will write changed pages from the buffer pool (or the buffer pools) to disk before the space in the buffer pool is required by a database agent. As a result, database agents should not have to wait for changed pages to be written out so that they might use the space in the buffer pool. This improves overall performance of the database applications.

When the page cleaners are started, they will build a list of the pages to write to disk. Once they have completed writing those pages to disk, they will become inactive again and wait for the next trigger to start.

When the `DB2_USE_ALTERNATE_PAGE_CLEANNING` registry variable is set (that is, the alternate method of page cleaning is used), the `chnpggs_thresh` parameter has no effect, and the database manager automatically determines how many dirty pages to maintain in the buffer pool.

**Recommendation:** For databases with a heavy update transaction workload, you can generally ensure that there are enough clean pages in the buffer pool by setting the parameter value to be equal-to or less-than the default value. A percentage larger than the default can help performance if your database has a small number of very large tables.

## **cluster\_mgr - Cluster manager name**

This parameter enables the database manager to communicate incremental cluster configuration changes to the specified cluster manager.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Multi-partitioned database server with local and remote clients

### **Parameter type**

Informational

### **Default**

No default

### **Valid values**

- TSA

This parameter is set during high availability cluster configuration using the DB2 High Availability Instance Configuration Utility (`db2haicu`).

## **codepage - Code page for the database**

This parameter shows the code page that was used to create the database. The `codepage` parameter is derived based on the `codeset` parameter.

### **Configuration type**

Database

### **Parameter type**

Informational

## **codeset - Codeset for the database**

This parameter shows the codeset that was used to create the database. Codeset is used by the database manager to determine `codepage` parameter values.

### **Configuration type**

Database

### **Parameter type**

Informational

## collate\_info - Collating information

This parameter determines the database's collating sequence. For a language-aware collation, the first 256 bytes contain the string representation of the collation name (for example, "SYSTEM\_819\_US").

This parameter can only be displayed using the db2CfgGet API. It **cannot** be displayed through the command line processor or the Control Center.

### Configuration type

Database

### Parameter type

Informational

This parameter provides 260 bytes of database collating information. The first 256 bytes specify the database collating sequence, where byte "n" contains the sort weight of the code point whose underlying decimal representation is "n" in the code page of the database.

The last 4 bytes contain internal information about the type of the collating sequence. The last four bytes of the parameter is an integer. The integer is sensitive to the endian order of the platform. The possible values are:

- 0 – The sequence contains non-unique weights
- 1 – The sequence contains all unique weights
- 2 – The sequence is the identity sequence, for which strings are compared byte for byte.
- 3 – The sequence is NLSCHAR, used for sorting characters in a TIS620-1 (code page 874) Thai database.
- 4 – The sequence is IDENTITY\_16BIT, which implements the "CESU-8 Compatibility Encoding Scheme for UTF-16: 8-bit" algorithm as specified in the Unicode Technical Report #26 available at the Unicode Technical Consortium Web site at <http://www.unicode.org>
- X'8001' – The sequence is UCA400\_NO, which implements the Unicode Collation<sup>®</sup> Algorithm (UCA) based on the Unicode Standard version 4.00, with normalization implicitly set to ON.
- X'8002' – The sequence is UCA400\_LTH, which implements the Unicode Collation Algorithm (UCA) based on the Unicode Standard version 4.00, and sorts all Thai characters as per the Royal Thai Dictionary order.
- X'8003' – The sequence is UCA400\_LSK, which implements the Unicode Collation Algorithm (UCA) based on the Unicode Standard version 4.00, and sorts all Slovakian characters properly.

**Note:** For a language-aware collation, the first 256 bytes contain the string representation of the collation name.

If you use this internal type information, you need to consider byte reversal when retrieving information for a database on a different platform.

You can specify the collating sequence at database creation time.

## comm\_bandwidth - Communications bandwidth

This parameter helps the query optimizer determine access paths by indicating the bandwidth between database partition servers.



**Configuration type**

Database manager

**Applies to**

Partitioned database server with local and remote clients

**Parameter type**

Configurable Online

**Propagation class**

Statement boundary

**Default [range]**

-1 [1 - 100 000 ]

A value of -1 causes the parameter value to be reset to the default. The default value is calculated based on the speed of the underlying communications adapter. A value of 100 can be expected for systems using Gigabit Ethernet.

**Unit of measure**

Megabytes per second

The value calculated for the communications bandwidth, in megabytes per second, is used by the query optimizer to estimate the cost of performing certain operations between the database partition servers of a partitioned database system. The optimizer does not model the cost of communications between a client and a server, so this parameter should reflect only the nominal bandwidth between the database partition servers, if any.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware.

**Recommendation:** You should only adjust this parameter if you want to model a different environment.

The communications bandwidth is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

## **contact\_host - Location of contact list**

This parameter specifies the location where the contact information used for notification by the Scheduler and the Health Monitor is stored.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

Null [any valid DB2 administration server TCP/IP hostname ]

The location is defined to be a DB2 administration server's TCP/IP hostname. Allowing *contact\_host* to be located on a remote DAS provides support for sharing



a contact list across multiple DB2 administration servers. If *contact\_host* is not specified, the DAS assumes the contact information is local.

This parameter can only be updated from a Version 8 command line processor (CLP).

## country/region - Database territory code

This parameter shows the *territory* code used to create the database.

### Configuration type

Database

### Parameter type

Informational

## cpuspeed - CPU speed

This parameter reflects the CPU speed of the machine(s) the database is installed on.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable online

### Propagation class

Statement boundary

### Default [range]

-1 [  $1 \times 10^{-10}$  — 1 ] A value of -1 will cause the parameter value to be reset based on the running of the measurement program.

### Unit of measure

Milliseconds

This program is executed if benchmark results are not available if the data for the IBM RS/6000® model 530H is not found in the file, or if the data for your machine is not found in the file.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware. By setting it to -1, *cpuspeed* will be re-computed.

**Recommendation:** You should only adjust this parameter if you want to model a different environment.

The CPU speed is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

## cur\_commit - Currently committed configuration parameter

This parameter controls the behavior of cursor stability (CS) scans.

**Configuration type**

Database

**Parameter type**

Configurable

**Default [range]**

ON [ON, AVAILABLE, DISABLED]

For new databases, the default is set to ON. When the default is set to ON your query will return the currently committed value of the data at the time when your query is submitted.

During database upgrade, the **cur\_commit** configuration parameter is set to DISABLED to maintain the same behavior as in previous releases. If you want to use currently committed on cursor stability scans, you need to set the **cur\_commit** configuration parameter to ON after the upgrade.

You can explicitly set the **cur\_commit** configuration parameter to AVAILABLE. Once you set this parameter, you need to explicitly request for currently committed behavior to see the results that are currently committed.

**Note:** Three registry variables DB2\_EVALUNCOMMITTED, DB2\_SKIPDELETED, and DB2\_SKIPINSERTED are affected by currently committed when cursor stability isolation level is used. These registry variables are ignored when USE CURRENTLY COMMITTED or WAIT FOR OUTCOME are specified explicitly on the bind or statement prepare time.

**Recommendation:** Increase the size of the buffer area using log buffer size parameter **logbufsz** if there is considerable read activity on a dedicated log disk, or there is high disk utilization. When increasing the value of **logbufsz** parameter, you should also consider the **dbheap** parameter since the log buffer area uses space controlled by the **dbheap** parameter.

## **das\_codepage - DAS code page**

This parameter indicates the code page used by the DB2 administration server.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

Null [any valid DB2 code page ]

If the parameter is null, then the default code page of the system is used. This parameter should be compatible with the locale of the local DB2 instances. Otherwise, the DB2 administration server cannot communicate with the DB2 instances.

This parameter can only be updated from a Version 8 command line processor (CLP).

## **das\_territory - DAS territory**

This parameter shows the territory used by the DB2 administration server.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

Null [any valid DB2 territory ]

If the parameter is null, then the default territory of the system is used.

This parameter can only be updated from a Version 8 command line processor (CLP).

## **database\_level - Database release level**

This parameter indicates the release level of the database manager which can use the database.

**Configuration type**

Database

**Parameter type**

Informational

In the case of an incomplete or failed database upgrade, this parameter will reflect the release level of the database before the upgrade and might differ from the *release* parameter (the release level of the database configuration file). Otherwise the value of *database\_level* will be identical to value of the *release* parameter.

## **database\_memory - Database shared memory size**

This parameter specifies the amount of memory that is reserved for the database shared memory region. If this amount is less than the amount calculated from the individual memory parameters (for example, locklist, utility heap, bufferpools, and so on), the larger amount will be used.

**Configuration type**

Database

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable online

**Default [range]**

Automatic [Computed, 0 - 4 294 967 295 ]

**Unit of measure**

Pages (4 KB)

**When allocated**

When the database is activated

**When freed**

When the database is deactivated

Setting this parameter to AUTOMATIC enables self-tuning. When enabled, the memory tuner determines the overall memory requirements for the database and increases or decreases the amount of memory allocated for database shared memory depending on the current database requirements. For example, if the current database requirements are high, and there is sufficient free memory on the system, more memory will be consumed by database shared memory. Once the database memory requirements drop, or the amount of free memory on the system drops too low, some database shared memory is released.

The memory tuner will always leave a minimum amount of memory free based on the calculated benefit to providing additional memory to the instance. If there is a great benefit to providing an instance with more memory, then the memory tuner will maintain a lower amount of free memory. If the benefit is lower, then more free memory will be maintained. This allows databases to cooperate in the distribution of system memory.

Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self-tuning to be active.

Automatic tuning of this configuration parameter will only occur when self-tuning memory is enabled for the database (the *self\_tuning\_mem* configuration parameter is set to ON).

To simplify the management of this parameter, the COMPUTED setting instructs the database manager to calculate the amount of memory needed, and to allocate it at database activation time. The database manager will also allocate some additional memory to satisfy peak memory requirements for any heap in the database shared memory region whenever a heap exceeds its configured size. Other operations, such as dynamic configuration updates, also have access to this additional memory. The db2pd command, with the -memsets option, can be used to monitor the amount of unused memory left in the database shared memory region.

**Recommendation:** This value will usually remain at AUTOMATIC. For environments that do not support the AUTOMATIC setting, this should be set to COMPUTED. For example, the additional memory can be used for creating new buffer pools, or for increasing the size of existing buffer pools.

**Note:** In Version 9.5, when you set *database\_memory* configuration parameter to AUTOMATIC, the initial database shared memory allocation is the configured size of all heaps and buffer pools defined for the database, and the memory increases as needed. If *database\_memory* is set to a specific value, then that requested amount of memory is allocated initially, during database activation. If the initial amount of memory cannot be allocated from the operating system, or exceeds the *instance\_memory* limit, database activation fails with an SQL1084C error (Shared memory segments cannot be allocated).

**Controlling DB2 Memory consumption:**

When *instance\_memory* is set to AUTOMATIC, a fixed upper bound on total memory consumption for the instance is set at instance startup (db2start).

Actual memory consumption by the database manager varies depending on the workload. When self-tuning memory manager is enabled to perform *database\_memory* tuning (by default for new databases), during run-time, self-tuning memory manager dynamically updates the size of performance-critical heaps within the database shared memory set according to the free physical memory on the system, while ensuring that there is sufficient free *instance\_memory* available for functional memory requirements. For more information, see the *instance\_memory* configuration parameter.

**Limitation on some Linux<sup>1</sup> kernels:**

Due to operating system limitations on some Linux kernels, self-tuning memory manager currently does not allow setting *database\_memory* to AUTOMATIC. However, this setting is now allowed on these kernels only when *instance\_memory* is set to a specific value, and not AUTOMATIC. If *database\_memory* is set to AUTOMATIC, and *instance\_memory* is later set back to AUTOMATIC, the *database\_memory* configuration parameter is automatically updated to COMPUTED during the next database activation. If some databases are already active, self-tuning memory manager stops tuning the overall *database\_memory* sizes.

<sup>1</sup>On Linux, this parameter supports the AUTOMATIC setting on RHEL5 and on SUSE 10 SP1 and newer. All other validated Linux distributions will return to COMPUTED if the kernel does not support this feature.

## **db2system - Name of the DB2 server system**

This parameter specifies the name that is used by your users and database administrators to identify the DB2 server system.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable Online

**Default [range]**

TCP/IP host name [any valid system name ]

If possible, this name should be unique within your network.

This name is displayed in the system level of the Control Center's object tree to aid administrators in the identification of server systems that can be administered from the Control Center.

When using the 'Search the Network' function of the Configuration Assistant, DB2 discovery returns this name and it is displayed at the system level in the resulting object tree. This name aids users in identifying the system that contains the database they want to access. A value for *db2system* is set at installation time as follows:

- On Windows, the setup program sets it equal to the computer name specified for the Windows system.
- On UNIX systems, it is set equal to the UNIX system's TCP/IP hostname.

## db\_mem\_thresh - Database memory threshold

This parameter represents the maximum percentage of committed, but currently unused, database shared memory that the database manager will allow before starting to release committed pages of memory back to the operating system.

### Configuration type

Database

### Parameter type

Configurable Online

### Propagation class

Immediate

### Default [range]

10 [0–100 ]

### Unit of measure

Percentage

This database configuration parameter relates to how the database manager handles excess unused database shared memory. Typically, as pages of memory are touched by a process, they are committed, meaning that a page of memory has been allocated by the operating system and occupies space either in physical memory or in a page file on disk. Depending on the database workload, there might be peak database shared memory requirements at a certain times of day. Once the operating system has enough committed memory to meet those peak requirements, that memory remains committed, even after peak memory requirements have subsided.

Acceptable values are whole numbers in the range of 0 (immediately release any unused database shared memory) to 100 (never release any unused database shared memory). The default is 10 (release unused memory only when more than 10% of database shared memory is currently unused), which should be suitable for most workloads.

This configuration parameter can be updated dynamically. Care should be taken when updating this parameter, as setting the value too low could cause excessive memory thrashing on the box (memory pages constantly being committed and then released), and setting the value too high might prevent the database manager from returning any database shared memory back to the operating system for other processes to use.

This configuration parameter will be ignored (meaning that unused database shared memory pages will remain committed) if the database shared memory region is pinned through the DB2\_PINNED\_BP registry variable, configured for large pages through the DB2\_LARGE\_PAGE\_MEM registry variable, or if releasing of memory is explicitly disabled through the DB2MEMDISCLAIM registry variable.

Some versions of Linux do not support releasing subranges of a shared memory segment back to the operating system. On such platforms, this parameter will be ignored.

## dbheap - Database heap

This parameter determines the maximum memory used by the database heap.

With Version 9.5, this database configuration parameter has a default value of *AUTOMATIC*, meaning that the database heap can increase as needed until either the *database\_memory* limit is reached, or the *instance\_memory* limit is reached.

**Configuration type**

Database

**Parameter type**

Configurable online

**Propagation class**

Immediate

**Default [range]**

Automatic [32 - 524 288]

**Unit of measure**

Pages (4 KB)

**When allocated**

When the database is activated

**When freed**

When the database is deactivated

There is one database heap per database, and the database manager uses it on behalf of all applications connected to the database. It contains control block information for tables, indexes, table spaces, and buffer pools. It also contains space for the log buffer (*logbufsz*) and temporary memory used by utilities. Therefore, the size of the heap will be dependent on a large number of variables. The control block information is kept in the heap until all applications disconnect from the database.

The minimum amount the database manager needs to get started is allocated at the first connection. The data area is expanded as needed until either the configured upper limit is reached, or, if set to *AUTOMATIC*, until all *database\_memory* or *instance\_memory*, or memory for both, is exhausted.

The following formula can be used as a rough guideline when deciding on a value to assign to the *dbheap* configuration parameter:

10K per tablespace + 4K per table + (1K + 4\*extents used),  
per range clustered table (RCT)

The *dbheap* value that you configure represents only a portion of the database heap that is allocated. The database heap is the main memory area used to satisfy global database memory requirements. It is sized to include basic allocations needed for the startup of a database in addition to the *dbheap* value. Tools which report memory usage such as Memory Tracker, Snapshot Monitor, and db2pd report the statistics of this larger database heap. There is no separate tracking of the allocations that are represented by the *dbheap* configuration parameter. Therefore, it is normal for the statistics on database heap memory usage reported from these tools to exceed the configured value for the *dbheap* parameter.

You can use the database system monitor to track the highest amount of memory that was used for the database heap, using the *db\_heap\_top* (maximum database heap allocated) element.

**Note:**



- Workload Management (WLM) work class sets and work action sets are stored in the database heap. However, a very small part of the memory is consumed for this.
- Trusted contexts, Workload Management, and Audit policy information is cached in memory for fast processing. This memory is allocated from the database heap. Therefore, user-defined trusted contexts, workload management, and audit policy objects would impose more memory requirements on the database heap. In this case, it is suggested that you set your database heap configuration to AUTOMATIC so that the database manager automatically manages the database heap size.

## **decflt\_rounding - Decimal floating point rounding configuration parameter**

This parameter allows you to specify the rounding mode for decimal floating point (DECFLOAT). The rounding mode affects decimal floating-point operations in the server, and in LOAD.

### **Configuration type**

Database

### **Parameter type**

Configurable

See “Consequences of changing decflt\_rounding” on page 774 below.

### **Default [range]**

ROUND\_HALF\_EVEN [ROUND\_CEILING, ROUND\_FLOOR, ROUND\_HALF\_UP, ROUND\_DOWN]

DB2 supports five IEEE-compliant decimal floating point rounding modes. The rounding mode specifies how to round the result of a calculation when the result exceeds the precision. The definitions for all the rounding modes are as follows:

### **ROUND\_CEILING**

Round towards +infinity. If all of the discarded digits are zero or if the sign is negative the result is unchanged. Otherwise, the result coefficient should be incremented by 1 (rounded up).

### **ROUND\_FLOOR**

Round towards -infinity. If all of the discarded digits are zero or if the sign is positive the result is unchanged. Otherwise, the sign is negative and the result coefficient should be incremented by 1.

### **ROUND\_HALF\_UP**

Round to nearest; if equidistant, round up 1. If the discarded digits represent greater than or equal to half (0.5) of the value of a 1 in the next left position then the result coefficient should be incremented by 1 (rounded up). Otherwise, the discarded digits (0.5 or less) are ignored.

### **ROUND\_HALF\_EVEN**

Round to nearest; if equidistant, round so that the final digit is even. If the discarded digits represent greater than half (0.5) the value of a one in the next left position, then the result coefficient should be increment by 1 (rounded up). If they represent less than half, then the result coefficient is not adjusted, that is, the discarded digits are ignored. Otherwise, if they represent exactly half, the result coefficient is unaltered if its rightmost digit is even, or incremented by 1 (rounded up) if its rightmost digit is



odd, to make an even digit. This rounding mode is the default rounding mode as per IEEE decimal floating point specification and is the default rounding mode in DB2 products.

### ROUND\_DOWN

Round towards 0 (truncation). The discarded digits are ignored.

Table 42 shows the result of rounding of 12.341, 12.345, 12.349, 12.355, and -12.345, each to 4 digits, under different rounding modes:

Table 42. Decimal floating point rounding modes

| Rounding mode   | 12.341 | 12.345 | 12.349 | 12.355 | -12.345 |
|-----------------|--------|--------|--------|--------|---------|
| ROUND_DOWN      | 12.34  | 12.34  | 12.34  | 12.35  | -12.34  |
| ROUND_HALF_UP   | 12.34  | 12.35  | 12.35  | 12.36  | -12.35  |
| ROUND_HALF_EVEN | 12.34  | 12.34  | 12.35  | 12.36  | -12.34  |
| ROUND_FLOOR     | 12.34  | 12.34  | 12.34  | 12.35  | -12.35  |
| ROUND_CEILING   | 12.35  | 12.35  | 12.35  | 12.36  | -12.34  |

### Consequences of changing decflt\_rounding

- Previously constructed materialized query tables (MQTs) could contain results that differ from what would be produced with the new rounding mode. To correct this problem, refresh potentially impacted MQTs.
- The results of a trigger may be affected by the new rounding mode. Changing it has no effect on data that has already been written.
- Constraints that allowed data to be inserted into a table, if reevaluated, might reject that same data. Similarly constraints that did not allow data to be inserted into a table, if reevaluated, might accept that same data. Use the SET INTEGRITY statement to check for and correct such problems. The value of a generated column whose calculation is dependent on decflt\_rounding could be different for two identical rows except for the generated column value, if one row was inserted before the change to decflt\_rounding and the other was inserted after.
- The rounding mode is not compiled into sections. Therefore, static SQL does not need to be recompiled after changing *decflt\_rounding*.

**Note:** The value of this configuration parameter is not changed dynamically but will become effective only after all applications disconnect from the database. If the database is activated, it must be deactivated.

### dft\_account\_str - Default charge-back account

This parameter acts as the default suffix of accounting identifiers.

#### Configuration type

Database manager

#### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

#### Parameter type

Configurable Online

**Propagation class**

Immediate

**Default [range]**

Null [any valid string ]

With each application connect request, an accounting identifier consisting of a DB2 Connect-generated prefix and the user supplied suffix is sent from the application requester to a DRDA<sup>®</sup> application server. This accounting information provides a mechanism for system administrators to associate resource usage with each user access.

**Note:** This parameter is only applicable to DB2 Connect.

The suffix is supplied by the application program calling the `sqlsact()` API or the user setting the environment variable `DB2ACCOUNT`. If a suffix is not supplied by either the API or environment variable, DB2 Connect uses the value of this parameter as the default suffix value. This parameter is particularly useful for earlier database clients (anything prior to version 2) that do not have the capability to forward an accounting string to DB2 Connect.

**Recommendation:** Set this accounting string using the following:

- Alphabets (A through Z)
- Numerics (0 through 9)
- Underscore (\_).

## **dft\_degree - Default degree**

This parameter specifies the default value for the `CURRENT DEGREE` special register and the `DEGREE` bind option.

**Configuration type**

Database

**Parameter type**

Configurable Online

**Propagation class**

Connection

**Default [range]**

1 [-1(ANY), 1 - 32 767]

The default value is 1.

A value of 1 means no intra-partition parallelism. A value of -1 (or ANY) means the optimizer determines the degree of intra-partition parallelism based on the number of processors and the type of query.

The degree of intra-partition parallelism for an SQL statement is specified at statement compilation time using the `CURRENT DEGREE` special register or the `DEGREE` bind option. The maximum runtime degree of intra-partition parallelism for an active application is specified using the `SET RUNTIME DEGREE` command. The Maximum Query Degree of Parallelism (*max\_querydegree*) configuration parameter specifies the maximum query degree of intra-partition parallelism for all SQL queries.

The actual runtime degree used is the lowest of:

- *max\_querydegree* configuration parameter
- application runtime degree
- SQL statement compilation degree

## **dft\_extent\_sz - Default extent size of table spaces**

This parameter sets the default extent size of table spaces.

### **Configuration type**

Database

### **Parameter type**

Configurable Online

### **Propagation class**

Immediate

### **Default [range]**

32 [2 – 256 ]

### **Unit of measure**

Pages

When a table space is created, `EXTENTSIZE n` can be optionally specified, where `n` is the extent size. If you do not specify the extent size on the `CREATE TABLESPACE` statement, the database manager uses the value given by this parameter.

**Recommendation:** In many cases, you will want to explicitly specify the extent size when you create the table space. Before choosing a value for this parameter, you should understand how you would explicitly choose an extent size for the `CREATE TABLESPACE` statement.

## **dft\_loadrec\_ses - Default number of load recovery sessions**

This parameter specifies the default number of sessions that will be used during the recovery of a table load.

### **Configuration type**

Database

### **Parameter type**

Configurable Online

### **Propagation class**

Immediate

### **Default [range]**

1 [1 - 30 000]

### **Unit of measure**

Counter

The value should be set to an optimal number of I/O sessions to be used to retrieve a load copy. The retrieval of a load copy is an operation similar to restore. You can override this parameter through entries in the copy location file specified by the environment variable `DB2LOADREC`.

The default number of buffers used for load retrieval is two more than the value of this parameter. You can also override the number of buffers in the copy location file.

This parameter is applicable only if roll forward recovery is enabled.

## **dft\_monswitches - Default database system monitor switches**

This parameter allows you to set a number of switches which are each internally represented by a bit of the parameter.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable Online

### **Propagation class**

Immediate

**Note:** The change takes effect immediately if you explicitly ATTACH to the instance before modifying the dft\_mon\_xxxx switch settings. Otherwise the setting takes effect the next time the instance is restarted.

### **Default**

All switches turned off, except dft\_mon\_timestamp, which is turned on by default

The parameter is unique in that you can update each of these switches independently by setting the following parameters:

### **dft\_mon\_uow**

Default value of the snapshot monitor's unit of work (UOW) switch

### **dft\_mon\_stmt**

Default value of the snapshot monitor's statement switch

### **dft\_mon\_table**

Default value of the snapshot monitor's table switch

### **dft\_mon\_bufpool**

Default value of the snapshot monitor's buffer pool switch

### **dft\_mon\_lock**

Default value of the snapshot monitor's lock switch

### **dft\_mon\_sort**

Default value of the snapshot monitor's sort switch

### **dft\_mon\_timestamp**

Default value of the snapshot monitor's timestamp switch

**Recommendation:** Any switch (except dft\_mon\_timestamp) that is turned ON instructs the database manager to collect monitor data related to that switch. Collecting additional monitor data increases database manager overhead which can impact system performance. Turning the dft\_mon\_timestamp switch OFF becomes important as CPU utilization approaches 100%. When this occurs, the CPU time required for issuing timestamps increases dramatically. Furthermore, if the timestamp switch is turned OFF, the overall cost of other data under monitor switch control is greatly reduced.

All monitoring applications inherit these default switch settings when the application issues its first monitoring request (for example, setting a switch, activating the event monitor, taking a snapshot). You should turn on a switch in the configuration file only if you want to collect data starting from the moment the database manager is started. (Otherwise, each monitoring application can set its own switches and the data it collects becomes relative to the time its switches are set.)

## **dft\_mttb\_types - Default maintained table types for optimization**

This parameter specifies the default value for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register. The value of this register determines what types of refresh deferred materialized query tables will be used during query optimization.

### **Configuration type**

Database

### **Parameter type**

Configurable

### **Default [range]**

SYSTEM [ALL, NONE, FEDERATED\_TOOL, SYSTEM, USER, or a list of values ]

You can specify a list of values separated by commas; for example, 'USER,FEDERATED\_TOOL'. ALL or NONE cannot be listed with other values, and you cannot specify the same value more than once. For use with the db2CfgSet and db2CfgGet APIs, the acceptable parameter values are: 8 (ALL), 4 (NONE), 16 (FEDERATED\_TOOL), 1 (SYSTEM) and 2 (USER). Multiple values can be specified together using bitwise OR; for example, 18 would be the equivalent of USER,FEDERATED\_TOOL. As before, the values 4 and 8 cannot be used with other values.

## **dft\_prefetch\_sz - Default prefetch size**

This parameter sets the default prefetch size of table spaces.

### **Configuration type**

Database

### **Parameter type**

Configurable Online

### **Propagation class**

Immediate

### **Default [range]**

Automatic [0 - 32 767]

### **Unit of measure**

Pages

When a table space is created, PREFETCHSIZE *n* can optionally be specified, where *n* represents the number of pages the database manager will read if prefetching is being performed. If you do not specify the prefetch size on invocation of the CREATE TABLESPACE statement, the database manager uses the current value of the *dft\_prefetch\_sz* parameter.

If a table space is created with `AUTOMATIC DFT_PREFETCH_SZ`, the prefetch size of the table space will become `AUTOMATIC`, which means that DB2 will automatically calculate and update the prefetch size of the table space, using the following equation:

$$\text{prefetch size} = (\# \text{ containers}) * (\# \text{ physical spindles}) * \text{extent size}$$

where the number of physical spindles defaults to 1 and can be specified through the DB2 registry variable `DB2_PARALLEL_IO`. This calculation is performed:

- At database start-up time
- When a table space is first created with `AUTOMATIC` prefetch size
- When the number of containers for a table space changes through execution of an `ALTER TABLESPACE` statement
- When the prefetch size for a table space is updated to be `AUTOMATIC` through execution of an `ALTER TABLESPACE` statement

The `AUTOMATIC` state of the prefetch size can be turned on or off as soon as the prefetch size is updated manually through invocation of the `ALTER TABLESPACE` statement.

**Recommendation:** Using system monitoring tools, you can determine if your CPU is idle while the system is waiting for I/O. Increasing the value of this parameter can help if the table spaces being used do not have a prefetch size defined for them.

This parameter provides the default for the entire database, and it might not be suitable for all table spaces within the database. For example, a value of 32 might be suitable for a table space with an extent size of 32 pages, but not suitable for a table space with an extent size of 25 pages. Ideally, you should explicitly set the prefetch size for each table space.

To help minimize I/O for table spaces defined with the default extent size (`dft_extent_sz`), you should set this parameter as a factor or whole multiple of the value of the `dft_extent_sz` parameter. For example, if the `dft_extent_sz` parameter is 32, you could set `dft_prefetch_sz` to 16 (a fraction of 32) or to 64 (a whole multiple of 32). If the prefetch size is a multiple of the extent size, the database manager might perform I/O in parallel, if the following conditions are true:

- The extents being prefetched are on different physical devices
- Multiple I/O servers are configured (`num_ioservers`).

## **dft\_queryopt - Default query optimization class**

The query optimization class is used to direct the optimizer to use different degrees of optimization when compiling SQL and XQuery queries. This parameter provides additional flexibility by setting the default query optimization class used when neither the `SET CURRENT QUERY OPTIMIZATION` statement nor the `QUERYOPT` option on the `bind` command are used.

### **Configuration type**

Database

### **Parameter type**

Configurable Online

### **Propagation class**

Connection

**Default [range]**  
5 [ 0 — 9 ]

**Unit of measure**  
Query Optimization Class (see below)

The query optimization classes currently defined are:

- 0 - minimal query optimization.
- 1 - roughly comparable to DB2 Version 1.
- 2 - slight optimization.
- 3 - moderate query optimization.
- 5 - significant query optimization with heuristics to limit the effort expended on selecting an access plan. This is the default.
- 7 - significant query optimization.
- 9 - maximal query optimization

## **dft\_refresh\_age - Default refresh age**

This parameter represents the maximum time duration since a REFRESH TABLE statement has been processed on a specific REFRESH DEFERRED materialized query table. After this time limit is exceeded, the materialized query table is not used to satisfy queries until the materialized query table is refreshed.

**Configuration type**  
Database

**Parameter type**  
Configurable

**Default [range]**  
0 [ 0, 99999999999999 (ANY)]

This parameter has the default value used for the REFRESH AGE if the CURRENT REFRESH AGE special register is not specified. This parameter specifies a time stamp duration value with a data type of DECIMAL(20,6). If the CURRENT REFRESH AGE has a value of 99999999999999 (ANY), and the QUERY OPTIMIZATION class has a value of two, or five or more, REFRESH DEFERRED materialized query tables are considered to optimize the processing of a dynamic query.

## **dft\_sqlmathwarn - Continue upon arithmetic exceptions**

This parameter sets the default value that determines the handling of arithmetic errors and retrieval conversion errors as errors or warnings during SQL statement compilation.

**Configuration type**  
Database

**Parameter type**  
Configurable

**Default [range]**  
No [No, Yes]

For static SQL statements, the value of this parameter is associated with the package at bind time. For dynamic SQL DML statements, the value of this parameter is used when the statement is prepared.

**Attention:** If you change the *dft\_sqlmathwarn* value for a database, the behavior of check constraints, triggers, and views that include arithmetic expressions might change. This might, in turn, have an impact on the data integrity of the database. You should only change the setting of *dft\_sqlmathwarn* for a database after carefully evaluating how the new arithmetic exception handling behavior might impact check constraints, triggers, and views. Once changed, subsequent changes require the same careful evaluation.

As an example, consider the following check constraint, which includes a division arithmetic operation:

```
A/B > 0
```

When *dft\_sqlmathwarn* is “No” and an INSERT with B=0 is attempted, the division by zero is processed as an arithmetic error. The insert operation fails because DB2 cannot check the constraint. If *dft\_sqlmathwarn* is changed to “Yes”, the division by zero is processed as an arithmetic warning with a NULL result. The NULL result causes the predicate to evaluate to UNKNOWN and the insert operation succeeds. If *dft\_sqlmathwarn* is changed back to “No”, an attempt to insert the same row will fail, because the division by zero error prevents DB2 from evaluating the constraint. The row inserted with B=0 when *dft\_sqlmathwarn* was “Yes” remains in the table and can be selected. Updates to the row that cause the constraint to be evaluated will fail, while updates to the row that do not require constraint re-evaluation will succeed.

Before changing *dft\_sqlmathwarn* from “No” to “Yes”, you should consider rewriting the constraint to explicitly handle nulls from arithmetic expressions. For example:

```
(A/B > 0) AND (CASE
 WHEN A IS NULL THEN 1
 WHEN B IS NULL THEN 1
 WHEN A/B IS NULL THEN 0
 ELSE 1
 END
 = 1)
```

can be used if both A and B are nullable. And, if A or B is not-nullable, the corresponding IS NULL WHEN-clause can be removed.

Before changing *dft\_sqlmathwarn* from “Yes” to “No”, you should first check for data that might become inconsistent by using, for example, predicates such as the following:

```
WHERE A IS NOT NULL AND B IS NOT NULL AND A/B IS NULL
```

When inconsistent rows are isolated, you should take appropriate action to correct the inconsistency before changing *dft\_sqlmathwarn*. You can also manually re-check constraints with arithmetic expressions after the change. To do this, first place the affected tables in a check pending state (with the OFF clause of the SET CONSTRAINTS statement), then request that the tables be checked (with the IMMEDIATE CHECKED clause of the SET CONSTRAINTS statement). Inconsistent data will be indicated by an arithmetic error, which prevents the constraint from being evaluated.

**Recommendation:** Use the default setting of no, unless you specifically require queries to be processed that include arithmetic exceptions. Then specify the value



of yes. This situation can occur if you are processing SQL statements that, on other database managers, provide results regardless of the arithmetic exceptions that occur.

## diaglevel - Diagnostic error capture level

This parameter specifies the type of diagnostic errors that will be recorded in the db2diag log file.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable Online

### Propagation class

Immediate

### Default [range]

3 [ 0 — 4 ]

Valid values for this parameter are:

- 0 – No diagnostic data captured
- 1 – Severe errors only
- 2 – All errors
- 3 – All errors and warnings
- 4 – All errors, warnings and informational messages

The *diagpath* configuration parameter is used to specify the directory that will contain the error file, alert log file, and any dump files that might be generated, based on the value of the *diaglevel* parameter.

**Recommendation:** You might want to increase the value of this parameter to gather additional problem determination data to help resolve a problem.

## diagpath - Diagnostic data directory path

This parameter allows you to specify the fully qualified path for DB2 diagnostic information.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable online

**Propagation class**

Immediate

**Default [range]**

Null [any valid path name ]

This directory could possibly contain dump files, trap files, an error log, a notification file, an alert log file, and first occurrence data collection (FODC) packages, depending on your platform.

If this parameter is null, the diagnostic information will be written to files in one of the following directories or folders:

- In Windows environments:
  - User data files, for example, files under instance directories, are written to a location that is different from where the code is installed, as follows:
    - In Windows Vista environments, user data files are written to ProgramData\IBM\DB2\.
    - In Windows 2003 and XP environments, user data files are written to Documents and Settings\All Users\Application Data\IBM\DB2\*Copy Name*, where *Copy Name* is the name of your DB2 copy.
- In Linux and UNIX environments: Information is written to *INSTHOME*/sqllib/db2dump, where *INSTHOME* is the home directory of the instance.

In Version 9.5, the default value of **DB2INSTPROF** at the global level is stored at the new location shown above. Other profile registry variables that specify the location of the runtime data files should query the value of **DB2INSTPROF**. The other variables include the following ones:

- **DB2CLIINIPATH**
- **DIAGPATH**
- **SPM\_LOG\_PATH**

**Recommendation:** Use the default setting for the *diagpath* configuration parameter or use a centralized location for the *diagpath* value of multiple instances.

In a partitioned database environment, the *diagpath* parameter should use local storage at the host to get the best performance from logging. This creates a separate logging and diagnostic directory for each physical partition. You can use the PD\_GET\_DIAG\_HIST table function to retrieve the log records from the different partitions, and the PD\_GET\_LOG\_MSGS table function to retrieve the notification log from all partitions.

## dir\_cache - Directory cache support

This parameter determines whether the database, node and DCS directory files will be cached in memory.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

Yes [Yes; No ]

**When allocated**

- When an application issues its first connect, the application directory cache is allocated
- When a database manager instance is started (db2start), the server directory cache is allocated.

**When freed**

- When an the application process terminates, the application directory cache is freed
- When a database manager instance is stopped (db2stop), the server directory cache is freed.

The use of the directory cache reduces connect costs by eliminating directory file I/O and minimizing the directory searches required to retrieve directory information. There are two types of directory caches:

- An application directory cache that is allocated and used for each application process on the machine at which the application is running.
- A server directory cache that is allocated and used for some of the internal database manager processes.

For application directory caches, when an application issues its first connect, each directory file is read and the information is cached in private memory for this application. The cache is used by the application process on subsequent connect requests and is maintained for the life of the application process. If a database is not found in the application directory cache, the directory files are searched for the information, but the cache is not updated. If the application modifies a directory entry, the next connect within that application will cause the cache for this application to be refreshed. The application directory cache for other applications will not be refreshed. When the application process terminates, the cache is freed. (To refresh the directory cache used by a command line processor session, issue a db2 terminate command.)

For server directory caches, when a database manager instance is started (db2start), each directory file is read and the information is cached in the server memory. This cache is maintained until the instance is stopped (db2stop). If a directory entry is not found in this cache, the directory files are searched for the information. This server directory cache is never refreshed during the time the instance is running.

**Recommendation:** Use directory caching if your directory files do not change frequently and performance is critical.

In addition, on remote clients, directory caching can be beneficial if your applications issue several different connection requests. In this case, caching reduces the number of times a single application must read the directory files.

Directory caching can also improve the performance of taking database system monitor snapshots. In addition, you should explicitly reference the database name on the snapshot call, instead of using database aliases.

**Note:** Errors might occur when performing snapshot calls if directory caching is turned on and if databases are cataloged, uncataloged, created, or dropped after the database manager is started.

## **discover - DAS discovery mode**

This parameter determines the type of discovery mode that is started when the DB2 Administration Server starts.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

SEARCH [DISABLE; KNOWN; SEARCH ]

- If discover = SEARCH, the administration server handles SEARCH discovery requests from clients. SEARCH provides a superset of the functionality provided by KNOWN discovery. When discover = SEARCH, the administration server will handle both SEARCH and KNOWN discovery requests from clients.
- If discover = KNOWN, the administration server handles only KNOWN discovery requests from clients.
- If discover = DISABLE, then the administration server will not handle any type of discovery request. The information for this server system is essentially hidden from clients.

The default discovery mode is SEARCH.

This parameter can only be updated from a Version 8 command line processor (CLP).

## **discover - Discovery mode**

This parameter determines what kind of discovery requests, if any, the client can make.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

SEARCH [DISABLE, KNOWN, SEARCH]

From a client perspective, one of the following will occur:

- If *discover* = SEARCH, the client can issue search discovery requests to find DB2 server systems on the network. Search discovery provides a superset of the functionality provided by KNOWN discovery. If *discover* = SEARCH, both search and known discovery requests can be issued by the client.
- If *discover* = KNOWN, only known discovery requests can be issued from the client. By specifying some connection information for the administration server on a particular system, all the instance and database information on the DB2 system is returned to the client.
- If *discover* = DISABLE, discovery is disabled at the client.

The default discovery mode is SEARCH.

## **discover\_db - Discover database**

This parameter is used to prevent information about a database from being returned to a client when a discovery request is received at the server.

### **Configuration type**

Database

### **Parameter type**

Configurable Online

### **Propagation class**

Immediate

### **Default [range]**

Enable [Disable, Enable]

The default for this parameter is that discovery is enabled for this database.

By changing this parameter value to “Disable”, it is possible to hide databases with sensitive data from the discovery process. This can be done in addition to other database security controls on the database.

## **discover\_inst - Discover server instance**

This parameter specifies whether this instance can be detected by DB2 discovery.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable online

### **Propagation class**

Immediate

### **Default [range]**

ENABLE [ENABLE, DISABLE]

The parameter’s default, enable, specifies that the instance can be detected, while disable prevents the instance from being discovered.

## **dyn\_query\_mgmt - Dynamic SQL and XQuery query management**

This parameter determines whether Query Patroller will capture information about submitted queries.

**Important:** This parameter has been deprecated because it is associated with Query Patroller functionality. With the new workload management features introduced in DB2 Version 9.5, Query Patroller and its related components have been deprecated in Version 9.7 and might be removed in a future release.

### **Configuration type**

Database

### **Parameter type**

Configurable Online

### **Default [range]**

0 (DISABLE) [ 1(ENABLE), 0 (DISABLE) ]

This parameter is relevant where DB2 Query Patroller is installed. If this parameter is set to “ENABLE”, Query Patroller captures information about the query, such as the submitter ID and the estimated cost of execution, as calculated by the optimizer. These values are used to determine whether the query should be managed by Query Patroller, based on user- and system-level thresholds.

If this parameter is set to “DISABLE”, Query Patroller does not capture any information about submitted queries, and no query management takes place.

## **enable\_xmlchar - Enable conversion to XML configuration parameter**

This parameter determines whether XMLPARSE operations can be performed on non-BIT DATA CHAR (or CHAR-type) expressions in an SQL statement.

### **Configuration type**

Database

### **Parameter type**

Configurable

### **Default [range]**

Yes [Yes; No]

When pureXML<sup>®</sup> features are used in a non-Unicode database, the XMLPARSE function can cause character substitutions to occur as SQL string data is converted from the client code page into the database code page, and then into Unicode for internal storage. Setting *enable\_xmlchar* to NO blocks the usage of character data types during XML parsing, and any attempts to insert character types into a non-Unicode database will generate an error. The BLOB data type and FOR BIT DATA data types are still allowed when *enable\_xmlchar* is set to NO, as code page conversion does not occur when these data types are used to pass XML data into a database.

By default, *enable\_xmlchar* is set to YES so that parsing of character data types is allowed. In this case, you should ensure that any XML data to be inserted contains only code points that are part of the database code page, in order to avoid substitution characters being introduced during insertion of the XML data.

**Note:** The client needs to disconnect and reconnect to the agent for this change to be reflected.

## **exec\_exp\_task - Execute expired tasks**

This parameter specifies whether or not the Scheduler will execute tasks that have been scheduled in the past, but have not yet been executed.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable

**Default [range]**

No [Yes; No ]

The Scheduler only detects expired tasks when it starts up. For example, if you have a job scheduled to run every Saturday, and the Scheduler is turned off on Friday and then restarted on Monday, the job scheduled for Saturday is now a job that is scheduled in the past. If *exec\_exp\_task* is set to Yes, your Saturday job will run when the Scheduler is restarted.

This parameter can only be updated from a Version 8 command line processor (CLP).

## **failarchpath - Failover log archive path**

This parameter specifies a path to which DB2 will try to archive log files if the log files cannot be archived to either the primary or the secondary (if set) archive destinations because of a media problem affecting those destinations. This specified path must reference a disk.

**Configuration type**

Database

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable Online

**Default [range]**

Null [ ]

If there are log files in the path specified by the current value of *failarchpath*, any updates to *failarchpath* will not take effect immediately. Instead, the update will take effect when all applications disconnect.

## **fed\_noauth - Bypass federated authentication**

This parameter determines whether federated authentication will be bypassed at the instance.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable online

**Propagation class**

Immediate

**Default [range]**

No [Yes; No]

When *fed\_noauth* is set to *yes*, *authentication* is set to *server* or *server\_encrypt*, and *federated* is set to *yes*, then authentication at the instance is bypassed. It is assumed that authentication will happen at the data source. Exercise caution when *fed\_noauth* is set to *yes*. Authentication is done at neither the client nor at DB2. Any user who knows the SYSADM authentication name can assume SYSADM authority for the federated server.

**federated - Federated database system support**

This parameter enables or disables support for applications submitting distributed requests for data managed by data sources (such as the DB2 Family and Oracle).

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

No [Yes; No]

**federated\_async - Maximum asynchronous TQs per query configuration parameter**

This parameter determines the maximum number of asynchrony table queues (ATQs) in the access plan that the federated server supports.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients when federation is enabled.

**Parameter type**

Configurable online



**Default [range]**

0 [0 to 32 767 inclusive, -1, ANY]

When ANY or -1 is specified, the optimizer determines the number of ATQs for the access plan. The optimizer assigns an ATQ to all eligible SHIP or remote pushdown operators in the plan. The value that is specified for DB2\_MAX\_ASYNC\_REQUESTS\_PER\_QUERY server option limits the number of asynchronous requests.

**Recommendation**

The *federated\_async* configuration parameter supplies the default or starting value for the special register and the bind option. You can override the value of this parameter by setting the value of the CURRENT FEDERATED ASYNCHRONY special register, FEDERATED\_ASYNCHRONY bind option, or FEDERATED\_ASYNCHRONY precompile option to a higher or a lower number.

If the special register or the bind option do not override the *federated\_async* configuration parameter, the value of the parameter determines the maximum number of ATQs in the access plan that the federated server allows. If the special register or the bind option overrides this parameter, the value of the special register or the bind option determines the maximum number of ATQs in the plan.

Any changes to the *federated\_async* configuration parameter affect dynamic statements as soon as the current unit of work commits. Subsequent dynamic statements recognize the new value automatically. A restart of the federated database is not needed. Embedded SQL packages are not invalidated nor implicitly rebound when the value of the *federated\_async* configuration parameter changes.

If you want the new value of the *federated\_async* configuration parameter to affect static SQL statements, you need to rebind the package.

## **fenced\_pool - Maximum number of fenced processes**

This parameter represents the number of threads cached in each db2fmp process for threaded db2fmp processes (processes serving threadsafe stored procedures and UDFs). For nonthreaded db2fmp processes, this parameter represents the number of processes cached.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable online

**Default [range]**

-1 (*max\_coordagents*), Automatic [-1; 0–64 000]

**Unit of measure**

Counter

**Restrictions:**

- If this parameter is updated dynamically, and the value is decreased, the database manager does not proactively terminate db2fmp threads or processes,

instead it stops caching them as they are used in order to reduce the number of cached db2fmp's down to the new value.

- If this parameter is updated dynamically, and the value is increased, the database manager caches more db2fmp threads and processes when they are created.
- When this parameter is set to -1, the default, it assumes the value of the *max\_coordagents* configuration parameter. Note that only the value of *max\_coordagents* is assumed and not the automatic setting or behavior.
- When this parameter is set to AUTOMATIC, also the default:
  - The database manager allows the number of db2fmp threads and processes cached to increase based on the high water mark of coordinating agents. Specifically, the automatic behavior of this parameter allows it to grow depending on the maximum number of coordinating agents the database manager has ever registered, at the same time, since it started.
  - The value assigned to this parameter represents a lower bound for the number of db2fmp threads and process to cache.

**Recommendation:** If your environment uses fenced stored procedures or user defined functions, then this parameter can be used to ensure that an appropriate number of db2fmp processes are available to process the maximum number of concurrent stored procedures and UDFs that run on the instance, ensuring that no new fenced mode processes need to be created as part of stored procedure and UDF execution.

If you find that the default value is not appropriate for your environment because an inappropriate amount of system resource is being given to db2fmp processes and is affecting performance of the database manager, the following might be useful in providing a starting point for tuning this parameter:

```
fenced_pool = # of applications allowed to make stored procedure and
UDF calls at one time
```

If *keepfenced* is set to YES, then each db2fmp process that is created in the cache pool will continue to exist and will use system resources even after the fenced routine call has been processed and returned to the agent.

If *keepfenced* is set to NO, then nonthreaded db2fmp processes will terminate when they complete execution, and there is no cache pool. Multithreaded db2fmp processes will continue to exist, but no threads will be pooled in these processes. This means that even when *keepfenced* is set to NO, you can have one threaded C db2fmp process and one threaded Java db2fmp process on your system.

In previous versions, this parameter was known as *maxdari*.

## hadr\_db\_role - HADR database role

This parameter indicates the current role of a database, whether the database is online or offline.

### Configuration type

Database

### Applies to

- Database server with local and remote clients
- Database server with local clients

**Parameter type**  
Informational

Valid values are: STANDARD, PRIMARY, or STANDBY.

**Note:** When a database is active, the HADR role of the database can also be determined using the GET SNAPSHOT FOR DATABASE command.

## **hadr\_local\_host - HADR local host name**

This parameter specifies the local host for high availability disaster recovery (HADR) TCP communication.

**Configuration type**  
Database

**Applies to**

- Database server with local and remote clients
- Database server with local clients

**Parameter type**  
Configurable

**Default**  
Null

Either a host name or an IP address can be used. If a host name is specified and it maps to multiple IP addresses, an error is returned, and HADR will not start up. If the host name maps to multiple IP addresses (even if you specify the same host name on primary and standby), primary and standby can end up mapping this host name to different IP addresses, because some DNS servers return IP address lists in non-deterministic order.

A host name is in the form: myserver.ibm.com. An IP address is in the form: "12.34.56.78".

## **hadr\_local\_svc - HADR local service name**

This parameter specifies the TCP service name or port number for which the local high availability disaster recovery (HADR) process accepts connections.

**Configuration type**  
Database

**Applies to**

- Database server with local and remote clients
- Database server with local clients

**Parameter type**  
Configurable

**Default**  
Null

The value for `hadr_local_svc` on the Primary or Standby database systems cannot be the same as the value of `svcname` or `svcname +1` on their respective nodes.

If you are using SSL, do not set `hadr_local_svc` on the Primary or Standby database system to the same value as you set for `ssl_svcname`.

## hadr\_peer\_window - HADR peer window configuration parameter

When you set **hadr\_peer\_window** to a non-zero time value, then a HADR primary-standby database pair continues to behave as though still in peer state, for the configured amount of time, if the primary database loses connection with the standby database. This helps ensure data consistency.

### Configuration type

Database

### Parameter type

Configurable

### Default [range]

0 [0 - 4 294 967 295]

### Unit of measure

Seconds

### Usage notes:

- The value will need to be the same on both primary and standby databases.
- A recommended minimum value is 120 seconds.
- The **hadr\_peer\_window** value is ignored when the **hadr\_syncmode** value is set to ASYNC. That is, the value is treated as if it were set to zero (0), since it is not meaningful in ASYNC mode.
- To avoid impacting the availability of the primary database when the standby database is intentionally shut down, for example, for maintenance, the peer window is not invoked if the standby database is explicitly deactivated while the HADR pair is in peer state.
- The takeover operation with the **hadr\_peer\_window** parameter may behave incorrectly if the primary database clock and the standby database clock are not synchronized to within 5 seconds of each other. That is, the operation may succeed when it should fail, or fail when it should succeed. You should use a time synchronization service (for example, NTP) to keep the clocks synchronized to the same source.
- On the standby databases, the peer window end time is based on the last heartbeat message received from the primary database rather than disconnection. Therefore, the standby database's remaining time in S-DisconnectedPeer state before transition to S-RemoteCatchupPending ranges from (**hadr\_peer\_window** - **hadr\_timeout**) seconds to (**hadr\_peer\_window**) seconds, depending on when and how the disconnection occurred.

## hadr\_remote\_host - HADR remote host name

This parameter specifies the TCP/IP host name or IP address of the remote high availability disaster recovery (HADR) database server.

### Configuration type

Database

### Applies to

- Database server with local and remote clients
- Database server with local clients

### Parameter type

Configurable

**Default**

Null

Similar to *hadr\_local\_host*, this parameter must map to only one IP address.

**hadr\_remote\_inst - HADR instance name of the remote server**

This parameter specifies the instance name of the remote server. Administration tools, such as the DB2 Control Center, use this parameter to contact the remote server. High availability disaster recovery (HADR) also checks whether a remote database requesting a connection belongs to the declared remote instance.

**Configuration type**

Database

**Applies to**

- Database server with local and remote clients
- Database server with local clients

**Parameter type**

Configurable

**Default**

Null

**hadr\_remote\_svc - HADR remote service name**

This parameter specifies the TCP service name or port number that will be used by the remote high availability disaster recovery (HADR) database server.

**Configuration type**

Database

**Applies to**

- Database server with local and remote clients
- Database server with local clients

**Parameter type**

Configurable

**Default**

Null

**hadr\_syncmode - HADR synchronization mode for log write in peer state**

This parameter specifies the synchronization mode, which determines how primary log writes are synchronized with the standby when the systems are in peer state.

**Configuration type**

Database

**Applies to**

- Database server with local and remote clients
- Database server with local clients

**Parameter type**

Configurable

**Default [range]**

NEARSYNC [ASYNC; SYNC ]

Valid values for this parameter are:

**SYNC** This mode provides the greatest protection against transaction loss, but at a higher cost of transaction response time.

In this mode, log writes are considered successful only when logs have been written to log files on the primary database and when the primary database has received acknowledgement from the standby database that the logs have also been written to log files on the standby database. The log data is guaranteed to be stored at both sites.

#### **NEARSYNC**

This mode provides somewhat less protection against transaction loss, in exchange for a shorter transaction response time than that of SYNC mode.

In this mode, log writes are considered successful only when the log records have been written to the log files on the primary database and when the primary database has received acknowledgement from the standby system that the logs have also been written to main memory on the standby system. Loss of data occurs only if both sites fail simultaneously and if the target site has not transferred to nonvolatile storage all of the log data that it has received.

#### **ASYNC**

This mode has the highest probability of transaction loss in the event of primary failure, in exchange for the shortest transaction response time among the three modes.

In this mode, log writes are considered successful only when the log records have been written to the log files on the primary database and have been delivered to the TCP layer of the primary system's host machine. Because the primary system does not wait for acknowledgement from the standby system, transactions might be considered committed when they are still on their way to the standby.

## **hadr\_timeout - HADR timeout value**

This parameter specifies the time (in seconds) that the high availability disaster recovery (HADR) process waits before considering a communication attempt to have failed.

#### **Configuration type**

Database

#### **Applies to**

- Database server with local and remote clients
- Database server with local clients

#### **Parameter type**

Configurable

#### **Default [range]**

120 [1 - 4 294 967 295 ]

## **health\_mon - Health monitoring**

This parameter allows you to specify whether you want to monitor an instance, its associated databases, and database objects according to various health indicators.

#### **Configuration type**

Database manager

**Parameter type**  
Configurable Online

**Propagation class**  
Immediate

**Default [range]**  
On [On; Off ]

**Related Parameters**

If *health\_mon* is turned on (the default), an agent will collect information about the health of the objects you have selected. If an object is considered to be in an unhealthy position, based on thresholds that you have set, notifications can be sent, and actions can be taken automatically. If *health\_mon* is turned off, the health of objects will not be monitored.

You can use the Health Center or the CLP to select the instance and database objects that you want to monitor. You can also specify where notifications should be sent, and what actions should be taken, based on the data collected by the health monitor.

## indexrec - Index re-creation time

This parameter indicates when the database manager will attempt to rebuild invalid indexes, and whether or not any index build will be redone during DB2 rollforward or HADR log replay on the standby database.

**Configuration type**  
Database and Database Manager

- Applies to**
- Database server with local and remote clients
  - Database server with local clients
  - Partitioned database server with local and remote clients

**Parameter type**  
Configurable Online

**Propagation class**  
Immediate

**Default [range]**

**UNIX Database Manager**  
restart [ restart; restart\_no\_redo; access; access\_no\_redo ]

**Windows Database Manager**  
restart [ restart; restart\_no\_redo; access; access\_no\_redo ]

**Database**  
Use system setting [system; restart; restart\_no\_redo; access; access\_no\_redo ]

There are five possible settings for this parameter:

**SYSTEM**  
*use system setting* specified in the database manager configuration file to decide when invalid indexes will be rebuilt, and whether any index build log records are to be redone during DB2 rollforward or HADR log replay. (Note: This setting is only valid for database configurations.)

## ACCESS

Invalid indexes are rebuilt when the index is first accessed. Any fully logged index builds are redone during DB2 rollforward or HADR log replay. When HADR is started and an HADR takeover occurs, any invalid indexes are rebuilt after takeover when the underlying table is first accessed.

## ACCESS\_NO\_REDO

Invalid indexes will be rebuilt when the underlying table is first accessed. Any fully logged index build will not be redone during DB2 rollforward or HADR log replay and those indexes will be left invalid. When HADR is started and an HADR takeover takes place, any invalid indexes will be rebuilt after takeover when the underlying table is first accessed.

## RESTART

The default value for *indexrec*. Invalid indexes will be rebuilt when a RESTART DATABASE command is either explicitly or implicitly issued. Any fully logged index build will be redone during DB2 rollforward or HADR log replay. When HADR is started and an HADR takeover takes place, any invalid indexes will be rebuilt at the end of takeover.

Note that a RESTART DATABASE command is implicitly issued if the *autorestart* parameter is enabled.

## RESTART\_NO\_REDO

Invalid indexes will be rebuilt when a RESTART DATABASE command is either explicitly or implicitly issued. (A RESTART DATABASE command is implicitly issued if the *autorestart* parameter is enabled.) Any fully logged index build will not be redone during DB2 rollforward or HADR log replay and instead those indexes will be rebuilt when rollforward completes or when HADR takeover takes place.

Indexes can become invalid when fatal disk problems occur. If this happens to the data itself, the data could be lost. However, if this happens to an index, the index can be recovered by re-creating it. If an index is rebuilt while users are connected to the database, two problems could occur:

- An unexpected degradation in response time might occur as the index file is re-created. Users accessing the table and using this particular index would wait while the index was being rebuilt.
- Unexpected locks might be held after index re-creation, especially if the user transaction that caused the index to be re-created never performed a COMMIT or ROLLBACK.

**Recommendation:** The best choice for this option on a high-user server and if restart time is not a concern, would be to have the index rebuilt at DATABASE RESTART time as part of the process of bringing the database back online after a crash.

Setting this parameter to "ACCESS" or to "ACCESS\_NO\_REDO" will result in a degradation of the performance of the database manager while the index is being re-created. Any user accessing that specific index or table would have to wait until the index is recreated.

If this parameter is set to "RESTART", the time taken to restart the database will be longer due to index re-creation, but normal processing would not be impacted once the database has been brought back online.



**Note:** At database recovery time, all SQL procedure executables on the file system that belong to the database being recovered are removed. If *indexrec* is set to RESTART, all SQL procedure executables are extracted from the database catalog and put back on the file system at the next connection to the database. If *indexrec* is not set to RESTART, an SQL executable is extracted to the file system only on first execution of that SQL procedure.

The difference between the RESTART and the RESTART\_NO\_REDO values, or between the ACCESS and the ACCESS\_NO\_REDO values, is only significant when full logging is activated for index build operations, such as CREATE INDEX and REORG INDEX operations, or for an index rebuild. You can activate logging by enabling the *logindexbuild* database configuration parameter or by enabling LOG INDEX BUILD when altering a table. By setting *indexrec* to either RESTART or ACCESS, operations involving a logged index build can be rolled forward without leaving the index object in an invalid state, which would require the index to be rebuilt at a later time.

## **instance\_memory - Instance memory**

This parameter specifies the maximum amount of memory that can be allocated for a database partition.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable online

### **Default [range]**

#### **32-bit platforms**

Automatic [0 - 1 000 000]

#### **64-bit platforms**

Automatic [0 - 68 719 476 736]

### **Unit of measure**

Pages (4 KB)

### **When allocated**

When the instance is started

### **When freed**

When the instance is stopped

The default value of **instance\_memory** is AUTOMATIC, meaning that its actual value is computed at database partition activation time (db2start). The actual value used is between 75 percent and 95 percent of the physical RAM on the system, divided by the number of configured local database partitions in the instance. This value should be suitable for dedicated database server systems.

### **Note:**

- If the specified value of **instance\_memory** is larger than the amount of physical memory on the system, then db2start fails with a SQL1220N (The database manager shared memory set cannot be allocated.)

- If **instance\_memory** is dynamically updated to a value less than the amount of physical RAM, the request is processed and a new upper limit is set. Dynamic decreases to **instance\_memory** are allowed only if the new setting is larger than the current amount of in-use **instance\_memory**, otherwise, the request is deferred to the next db2start.
- If **instance\_memory** is dynamically updated to a value greater than the amount of physical RAM while the instance is active, the request is deferred, and the next db2start fails with a SQL1220N (The database manager shared memory set cannot be allocated.)

When fast communication manager (FCM) shared memory is allocated, each local database partition's share of the overall FCM shared memory size for the system is accounted for in that database partition's **instance\_memory** limit.

If memory is requested for a particular heap, and the database partition memory limit (**instance\_memory**) has already been reached, then DB2 will first attempt to reduce memory usage in all shared and private heaps by the requested amount of memory. If there is still insufficient free **instance\_memory**, then the request fails, and the application that initiated the request receives an appropriate SQLCODE that describes which heap experienced an out-of-memory failure.

The exception to this behavior is for memory requests that are known to be critical to the operation of DB2 (that is, failing the memory request results the database is marked as invalid, or the instance is shut down). Note that the critical requests will first attempt to reduce current memory usage by the database partition. If there is still insufficient free **instance\_memory**, DB2 still requests that memory from the operating system. If the operating system allows the memory request, then the current value of **instance\_memory** will exceed the configured limit, however, all other non-critical memory requests will fail until enough memory has been freed.

**Note: Restriction for DPF instances:** although **instance\_memory** specifies the amount of memory a single DB2 database partition might allocate, it is an instance-level configuration parameter, so all database partitions have the same **instance\_memory** setting. However, if **instance\_memory** is set to AUTOMATIC, the actual upper bound is computed individually on each separate machine based on the amount of RAM and the number of local partitions defined, so it is possible for different partitions to have different memory limits in effect.

#### **Controlling DB2 Memory consumption:**

When **instance\_memory** is set to AUTOMATIC, a fixed upper bound on total memory consumption for the instance is set at instance startup (db2start). Actual memory consumption by DB2 varies depending on the workload. When STMM is enabled to perform **database\_memory** tuning (by default for new databases), during run-time, STMM dynamically updates the size of performance-critical heaps within the database shared memory set according to the free physical memory on the system, while ensuring that there is sufficient free **instance\_memory** available for functional memory requirements.

Depending on workload, DB2's default memory configuration adapts to the memory requirements of the instance without requiring explicit self-tuning of overall instance memory. For instance:

- For heavily-used instances, STMM increases the size of performance-critical heaps as needed. More functional memory is consumed, as there are more database agents servicing applications and

consuming functional memory. If there is enough free **instance\_memory** but very little free physical memory on the system, STMM starts decreasing the size of performance-critical heaps ensure that the system does not start paging. As functional memory requirements drop, free physical memory on the system should increase, and STMM will start increasing the performance-critical heaps again.

- For less heavily-used instances, there is less functional memory consumed by the instance, and if there is insufficient free physical memory left on the system, STMM shrinks performance-critical heaps.

If **instance\_memory** is set to a specific value, and at least one active database has an AUTOMATIC value for **database\_memory**, and STMM is enabled for that database, then STMM increases the **database\_memory** size such that DB2 uses almost the entire amount of memory specified by **instance\_memory**, ensuring only that enough free **instance\_memory** is available for functional memory requests. In this scenario, STMM does not monitor free physical memory on the machine, therefore, **instance\_memory** must be configured properly to ensure that paging will not occur.

Use the new `admin_get_dbp_mem_usage` user-defined function (UDF) to get the total memory consumption by a DB2 instance for a specific database partition, or for all database partitions. This UDF also returns the current upper bound value.

#### Limitation on some Linux kernels:

Due to operating system limitations on some Linux kernels, STMM does not allow setting **database\_memory** to AUTOMATIC unless **instance\_memory** is set to a specific value (not AUTOMATIC). If **database\_memory** is set to AUTOMATIC, and **instance\_memory** is later set back to AUTOMATIC, the **database\_memory** configuration parameter will be updated to COMPUTED during the next database activation. If some databases are already active, STMM stops tuning their overall **database\_memory** sizes. This limitation is removed on Red Hat Enterprise Linux (RHEL) 5 and SUSE Linux Enterprise Server 10 SP1 or higher platforms.

## intra\_parallel - Enable intra-partition parallelism

This parameter specifies whether the database manager can use intra-partition parallelism.

#### Configuration type

Database manager

#### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

#### Parameter type

Configurable

#### Default [range]

NO (0) [SYSTEM (-1), NO (0), YES (1)]

A value of -1 causes the parameter value to be set to "YES" or "NO" based on the hardware on which the database manager is running.

Some of the operations that can take advantage of parallel performance improvements when this parameter is "YES" include database queries and index creation.

**Note:**

- Parallel index creation does not use this configuration parameter.
- If you change this parameter value, packages might be rebound to the database, and some performance degradation might occur.

## **java\_heap\_sz - Maximum Java interpreter heap size**

This parameter determines the maximum size of the heap that is used by the Java interpreter started to service Java DB2 stored procedures and UDFs.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

**HP-UX**

4096 [0 - 524 288]

**All other operating systems**

2048 [0 - 524 288]

**Unit of measure**

Pages (4 KB)

**When allocated**

When a Java stored procedure or UDF starts

**When freed**

When the db2fmp process (fenced) or the db2agent process (trusted) terminates.

There is one heap for each DB2 process (one for each agent or subagent on Linux and UNIX platforms, and one for each instance on other platforms). There is one heap for each fenced UDF and fenced stored procedure process. There is one heap per agent (not including sub-agents) for trusted routines. There is one heap per db2fmp process running a Java stored procedure. For multithreaded db2fmp processes, multiple applications using threadsafe fenced routines are serviced from a single heap. In all situations, only the agents or processes that run Java UDFs or stored procedures ever allocate this memory. On partitioned database systems, the same value is used at each database partition.

XML data is materialized when passed to stored procedures as IN, OUT, or INOUT parameters. When you are using Java stored procedures, the heap size might need to be increased based on the quantity and size of XML arguments, and the number of external stored procedures that are being executed concurrently.

## **jdk\_64\_path - 64-Bit Software Developer's Kit for Java installation path DAS**

This parameter specifies the directory under which the 64-Bit Software Developer's Kit (SDK) for Java, to be used for running DB2 administration server functions, is installed.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

Null [any valid path ]

**Note:** This is different from the **jdk\_path** configuration parameter, which specifies a 32-bit SDK for Java.

Environment variables used by the Java interpreter are computed from the value of this parameter. This parameter is only used on those platforms that support both 32- and 64-bit instances.

Those platforms are:

- 64-bit kernels of AIX, HP-UX, and Solaris operating systems
- 64-bit Windows on X64 and IPF
- 64-bit Linux kernel on AMD64 and Intel EM64T systems (x64), POWER, and zSeries.

On all other platforms, only **jdk\_path** is used.

Because there is no default value for this parameter, you should specify a value when you install the SDK for Java.

This parameter can only be updated from a Version 8 command line processor (CLP).

## **jdk\_path - Software Developer's Kit for Java installation path DAS**

This parameter specifies the directory under which the Software Developer's Kit (SDK) for Java, to be used for running DB2 administration server functions, is installed.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

Default Java install path [any valid path ]

Environment variables used by the Java interpreter are computed from the value of this parameter.

On Windows operating systems, Java files (if needed) are placed under the sqllib directory (in java\jdk) during DB2 installation. The *jdk\_path* configuration parameter is then set to sqllib\java\jdk. Java is never actually installed by DB2 on Windows platforms; the files are merely placed under the sqllib directory, and this is done regardless of whether or not Java is already installed.

This parameter can only be updated from a Version 8 command line processor (CLP).

## **jdk\_path - Software Developer's Kit for Java installation path**

This parameter specifies the directory under which the Software Developer's Kit (SDK) for Java, to be used for running Java stored procedures and user-defined functions, is installed. The CLASSPATH and other environment variables used by the Java interpreter are computed from the value of this parameter.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

Null [Valid path]

If the SDK for Java was installed with your DB2 product, this parameter is set properly. However, if you reset the database manager (dbm cfg) parameter, you need to specify where the SDK for Java is installed.

## **keepfenced - Keep fenced process**

This parameter indicates whether or not a fenced mode process is kept after a fenced mode routine call is complete. Fenced mode processes are created as separate system entities in order to isolate user-written fenced mode code from the database manager agent process. This parameter is only applicable on database servers.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

Yes [Yes; No ]

If *keepfenced* is set to *no*, and the routine being executed is not threadsafe, a new fenced mode process is created and destroyed for each fenced mode invocation. If *keepfenced* is set to *no*, and the routine being executed is threadsafe, the fenced mode process persists, but the thread created for the call is terminated. If *keepfenced* is set to *yes*, a fenced mode process or thread is reused for subsequent fenced mode calls. When the database manager is stopped, all outstanding fenced mode processes and threads will be terminated.

Setting this parameter to *yes* will result in additional system resources being consumed by the database manager for each fenced mode process that is activated, up to the value contained in the *fenced\_pool* parameter. A new process is only created when no existing fenced mode process is available to process a subsequent fenced routine invocation. This parameter is ignored if *fenced\_pool* is set to 0.

**Recommendation:** In an environment in which the number of fenced mode requests is large relative to the number of non-fenced mode requests, and system resources are not constrained, then this parameter can be set to *yes*. This will improve the fenced mode process performance by avoiding the initial fenced mode process creation overhead since an existing fenced mode process will be used to process the call. In particular, for Java routines, this will save the cost of starting the Java Virtual Machine (JVM), a very significant performance improvement.

For example, in an OLTP debit-credit banking transaction application, the code to perform each transaction could be performed in a stored procedure which executes in a fenced mode process. In this application, the main workload is performed out of fenced mode processes. If this parameter is set to *no*, each transaction incurs the overhead of creating a new fenced mode process, resulting in a significant performance reduction. If, however, this parameter is set to *yes*, each transaction would try to use an existing fenced mode process, which would avoid this overhead.

In previous versions of DB2, this parameter was known as *keepdari*.

## log\_retain\_status - Log retain status indicator

If set (when the *logretain* parameter setting is equal to Recovery), this parameter indicates that log files are being retained for use in roll-forward recovery.

**Configuration type**

Database

**Parameter type**

Informational

## logarchmeth1 - Primary log archive method

This parameter specifies the media type of the primary destination for archived logs.

**Configuration type**

Database

**Applies to**

- Database server with local and remote clients
- Client



- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable Online

**Default [range]**

OFF [LOGRETAIN, USEREXIT, DISK, TSM, VENDOR]

**OFF** Specifies that the log archiving method is not to be used. If both **logarchmeth1** and **logarchmeth2** are set to OFF, the database is considered to be using circular logging and will not be rollforward recoverable. This is the default.

**LOGRETAIN**

This value can only be used for **logarchmeth1** and is equivalent to setting the **logretain** configuration parameter to RECOVERY. If you specify this value, the **logretain** configuration parameters will automatically be updated.

**USEREXIT**

This value is only valid for **logarchmeth1** and is equivalent to setting the **userexit** configuration parameter to ON. If specify this value, the **userexit** configuration parameter will be automatically updated.

**DISK** This value must be followed by a colon(:) and then a fully qualified existing path name where the log files will be archived. For example, if you set **logarchmeth1** to DISK:/u/dbuser/archived\_logs the archive log files will be placed in a directory called /u/dbuser/archived\_logs.

**Note:** If you are archiving to tape, you can use the db2tapemgr utility to store and retrieve log files.

**TSM** If specified without any additional configuration parameters, this value indicates that log files should be archived on the local TSM server using the default management class. If followed by a colon(:) and a TSM management class, the log files will be archived using the specified management class.

When archiving logs using TSM, before using the management class specified by the database configuration parameter, TSM first attempts to bind the object to the management class specified in the INCLUDE-EXCLUDE list found in the TSM client options file. If a match is not found, the default TSM management class specified on the TSM server will be used. TSM will then rebind the object to the management class specified by the database configuration parameter.

Thus, the default management class, as well as the management class specified by the database configuration parameter, must contain an archive copy group, or the archive operation will fail.

**VENDOR**

Specifies that a vendor library will be used to archive the log files. This value must be followed by a colon(:) and the name of the library. The APIs provided in the library must use the backup and restore APIs for vendor products.

**Note:**



1. If either **logarchmeth1** or **logarchmeth2** is set to a value other than OFF, the database is configured for rollforward recovery.
2. If you update the **userexit** or **logretain** configuration parameters **logarchmeth1** will automatically be updated and vice versa. However, if you are using either **userexit** or **logretain**, **logarchmeth2** must be set to OFF.

## logarchmeth2 - Secondary log archive method

This parameter specifies the media type of the secondary destination for archived logs.

### Configuration type

Database

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable Online

### Default [range]

OFF [ LOGRETAIN, USEREXIT, DISK, TSM, VENDOR]

**OFF** Specifies that the log archiving method is not to be used. If both *logarchmeth1* and *logarchmeth2* are set to OFF, the database is considered to be using circular logging and will not be rollforward recoverable. This is the default.

### LOGRETAIN

This value can only be used for *logarchmeth1* and is equivalent to setting the *logretain* configuration parameter to RECOVERY. If you specify this value, the *logretain* configuration parameters will automatically be updated.

### USEREXIT

This value is only valid for *logarchmeth1* and is equivalent to setting the *userexit* configuration parameter to ON. If specify this value, the *userexit* configuration parameter will be automatically updated.

**DISK** This value must be followed by a colon(:) and then a fully qualified existing path name where the log files will be archived. For example, if you set *logarchmeth1* to DISK:/u/dbuser/archived\_logs the archive log files will be placed in a directory called /u/dbuser/archived\_logs.

**Note:** If you are archiving to tape, you can use the db2tapemgr utility to store and retrieve log files.

**TSM** If specified without any additional configuration parameters, this value indicates that log files should be archived on the local TSM server using the default management class. If followed by a colon(:) and a TSM management class, the log files will be archived using the specified management class.

## VENDOR

Specifies that a vendor library will be used to archive the log files. This value must be followed by a colon(:) and the name of the library. The APIs provided in the library must use the backup and restore APIs for vendor products.

### Note:

1. If either *logarchmeth1* or *logarchmeth2* is set to a value other than OFF, the database is configured for rollforward recovery.
2. If you update the *userexit* or *logretain* configuration parameters *logarchmeth1* will automatically be updated and vice versa. However, if you are using either *userexit* or *logretain*, *logarchmeth2* must be set to OFF.

If this path is specified, log files will be archived to both this destination and the destination specified by the *logarchmeth1* database configuration parameter.

## logarchopt1 - Primary log archive options

This parameter specifies the options field for the primary destination for archived logs (if required).

### Configuration type

Database

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable Online

### Default [range]

Null [not applicable]

## logarchopt2 - Secondary log archive options

This parameter specifies the options field for the secondary destination for archived logs (if required).

### Configuration type

Database

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable Online

### Default [range]

Null [not applicable]

## logbufsz - Log buffer size

This parameter allows you to specify the amount of the database heap (defined by the *dbheap* parameter) to use as a buffer for log records before writing these records to disk.

### Configuration type

Database

### Parameter type

Configurable

### Default [range]

#### 32-bit platforms

8 [4 - 4 096 ]

#### 64-bit platforms

8 [4 - 131 070 ]

### Unit of measure

Pages (4 KB)

Log records are written to disk when one of the following occurs:

- A transaction commits or a group of transactions commit, as defined by the *mincommit* configuration parameter
- The log buffer is full
- As a result of some other internal database manager event.

This parameter must also be less than or equal to the *dbheap* parameter. Buffering the log records will result in more efficient logging file I/O because the log records will be written to disk less frequently and more log records will be written at each time.

**Recommendation:** Increase the size of this buffer area if there is considerable read activity on a dedicated log disk, or there is high disk utilization. When increasing the value of this parameter, you should also consider the *dbheap* parameter since the log buffer area uses space controlled by the *dbheap* parameter.

You can use the database system monitor to determine how much of the log buffer space is used for a particular transaction (or unit of work). Refer to the *log\_space\_used* (unit of work log space used) monitor element.

## logfilsiz - Size of log files

This parameter defines the size of each primary and secondary log file. The size of these log files limits the number of log records that can be written to them before they become full and a new log file is required.

### Configuration type

Database

### Parameter type

Configurable

### Default [range]

UNIX 1000 [4 - 1 048 572]

#### Windows

1000 [4 - 1 048 572]

## Unit of measure

Pages (4 KB)

The use of primary and secondary log files as well as the action taken when a log file becomes full are dependent on the type of logging that is being performed:

- Circular logging

A primary log file can be reused when the changes recorded in it have been committed. If the log file size is small and applications have processed a large number of changes to the database without committing the changes, a primary log file can quickly become full. If all primary log files become full, the database manager will allocate secondary log files to hold the new log records.

- Log retention logging

When a primary log file is full, the log is archived and a new primary log file is allocated.

**Recommendation:** You must balance the size of the log files with the number of primary log files:

- The value of the *logfilsiz* should be increased if the database has a large number of update, delete, or insert transactions running against it which will cause the log file to become full very quickly.

**Note:** The upper limit of log file size, combined with the upper limit of the number of log files (*logprimary* + *logsecond*), gives an upper limit of 1024 GB of active log space.

A log file that is too small can affect system performance because of the overhead of archiving old log files, allocating new log files, and waiting for a usable log file.

- The value of the *logfilsiz* should be reduced if disk space is scarce, since primary logs are preallocated at this size.

A log file that is too large can reduce your flexibility when managing archived log files and copies of log files, since some media might not be able to hold an entire log file.

If you are using log retention, the current active log file is closed and truncated when the last application disconnects from a database. When the next connection to the database occurs, the next log file is used. Therefore, if you understand the logging requirements of your concurrent applications, you might be able to determine a log file size that will not allocate excessive amounts of wasted space.

## loghead - First active log file

This parameter contains the name of the log file that is currently active.

### Configuration type

Database

### Parameter type

Informational

## logindexbuild - Log index pages created

This parameter specifies whether index creation, recreation, or reorganization operations are to be logged so that indexes can be reconstructed during DB2 rollforward operations or high availability disaster recovery (HADR) log replay procedures.

**Configuration type**

Database

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable Online

**Default [range]**

Off [On; Off ]

## logpath - Location of log files

This parameter contains the current path being used for logging purposes.

**Configuration type**

Database

**Parameter type**

Informational

You cannot change this parameter directly as it is set by the database manager after a change to the *newlogpath* parameter becomes effective.

When a database is created, the recovery log file for it is created in a subdirectory of the directory containing the database. The default is a subdirectory named *SQLLOGDIR* under the directory created for the database.

## logprimary - Number of primary log files

This parameter allows you to specify the number of primary log files to be preallocated. The primary log files establish a fixed amount of storage allocated to the recovery log files.

**Configuration type**

Database

**Parameter type**

Configurable

**Default [range]**

3 [ 2 - 256 ]

**Unit of measure**

Counter

**When allocated**

- The database is created
- A log is moved to a different location (which occurs when the *logpath* parameter is updated)
- When the database is next started following an increase following an increase in the value of this parameter (*logprimary*), provided that the database is not started as an HADR standby database
- A log file has been archived and a new log file is allocated (the *logretain* or *userexit* parameter must be enabled)

- If the *logfilsiz* parameter has been changed, the log files are re-sized during the next database startup, provided that it is not started as an HADR standby database

#### When freed

Not freed unless this parameter decreases. If decreased, unneeded log files are deleted during the next connection to the database.

Under circular logging, the primary logs are used repeatedly in sequence. That is, when a log is full, the next primary log in the sequence is used if it is available. A log is considered available if all units of work with log records in it have been committed or rolled-back. If the next primary log in sequence is not available, then a secondary log is allocated and used. Additional secondary logs are allocated and used until the next primary log in the sequence becomes available or the limit imposed by the *logsecond* parameter is reached. These secondary log files are dynamically deallocated as they are no longer needed by the database manager.

The number of primary and secondary log files must comply with the following:

- If *logsecond* has a value of -1,  $logprimary \leq 256$ .
- If *logsecond* does not have a value of -1,  $(logprimary + logsecond) \leq 256$ .

**Recommendation:** The value chosen for this parameter depends on a number of factors, including the type of logging being used, the size of the log files, and the type of processing environment (for example, length of transactions and frequency of commits).

Increasing this value will increase the disk requirements for the logs because the primary log files are preallocated during the very first connection to the database.

If you find that secondary log files are frequently being allocated, you might be able to improve system performance by increasing the log file size (*logfilsiz*) or by increasing the number of primary log files.

For databases that are not frequently accessed, in order to save disk storage, set the parameter to 2. For databases enabled for roll-forward recovery, set the parameter larger to avoid the overhead of allocating new logs almost immediately.

You can use the database system monitor to help you size the primary log files. Observation of the following monitor values over a period of time will aid in better tuning decisions, as average values might be more representative of your ongoing requirements.

- *sec\_log\_used\_top* (maximum secondary log space used)
- *tot\_log\_used\_top* (maximum total log space used)
- *sec\_logs\_allocated* (secondary logs allocated currently)

## logretain - Log retain enable

This parameter is deprecated in Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the DB2 Version 9.5 database manager.

**Note:** The following information applies only to pre-Version 9.5 data servers and clients.

This parameter determines whether active log files are retained and available for roll-forward recovery.

**Configuration type**

Database

**Parameter type**

Configurable

**Default [range]**

No [ Recovery; No ]

The values are as follows:

- No, to indicate that logs are not retained.
- Recovery, to indicate that the logs are retained, and can be used for forward recovery.

If **logretain** is set to Recovery or **userexit** is set to Yes, the active log files will be retained and become online archive log files for use in roll-forward recovery. This is called log retention logging.

After **logretain** is set to Recovery or **userexit** is set to Yes (or both), you must make a full backup of the database. This state is indicated by the **backup\_pending** flag parameter.

**Note:**

Both **logarchmeth1** or **logretain** will enable rollforward recovery. However, only one method should be enabled for a database at one time.

If using **logarchmeth1**, do not set the **logretain** and **userexit** configuration parameters. If the **logretain** configuration parameter is set to recover, the value for **logarchmeth1** will automatically be set to **logretain**.

It is recommended that **logarchmeth1** (and **logarchmeth2**) be used rather than **logretain** and **userexit** to activate archive logging and rollforward recovery. The **logretain** and **userexit** options have been kept to support users who have not yet migrated to **logarchmeth1**.

## logsecond - Number of secondary log files

This parameter specifies the number of secondary log files that are created and used for recovery log files (only as needed).

**Configuration type**

Database

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

2 [-1; 0 – 254 ]

**Unit of measure**

Counter

**When allocated**As needed when *logprimary* is insufficient (see detail below)

### When freed

Over time as the database manager determines they will no longer be required.

When the primary log files become full, the secondary log files (of size *logfilsiz*) are allocated one at a time as needed, up to a maximum number as controlled by this parameter. An error code will be returned to the application, and the database will be shut down, if more secondary log files are required than are allowed by this parameter.

If you set *logsecond* to -1, the database is configured with infinite active log space. There is no limit on the size or the number of in-flight transactions running on the database. If you set *logsecond* to -1, you still use the *logprimary* and *logfilsiz* configuration parameters to specify how many log files the database manager should keep in the active log path. If the database manager needs to read log data from a log file, but the file is not in the active log path, the database manager retrieves the log file from the archive to the active log path. (The database manager retrieves the files to the overflow log path, if you have configured one.) Once the log file is retrieved, the database manager will cache this file in the active log path so that other reads of log data from the same file will be fast. The database manager will manage the retrieval, caching, and removal of these log files as required.

If your log path is a raw device, you must configure the *overflowlogpath* configuration parameter in order to set *logsecond* to -1.

By setting *logsecond* to -1, you will have no limit on the size of the unit of work or the number of concurrent units of work. However, rollback (both at the savepoint level and at the unit of work level) could be very slow due to the need to retrieve log files from the archive. Crash recovery could also be very slow for the same reason. The database manager writes a message to the administration notification log to warn you that the current set of active units of work has exceeded the primary log files. This is an indication that rollback or crash recovery could be extremely slow.

To set *logsecond* to -1, the *logarchmeth1* configuration parameter must be set to a value other than OFF or LOGRETAIN.

**Recommendation:** Use secondary log files for databases that have periodic needs for large amounts of log space. For example, an application that is run once a month might require log space beyond that provided by the primary log files. Since secondary log files do not require permanent file space they are advantageous in this type of situation.

When infinite logging is enabled (*logsecond* to -1), the database manager does not reserve active log space for transactions that may need to roll back and write log records. During rollback processing, if both the active log path and archive target are full (or if the archive target is inaccessible), then the *blk\_log\_dsk\_ful* (block on log disk full db configuration parameter) should also be ENABLED to avoid database failures.

## max\_connections - Maximum number of client connections

This parameter indicates the maximum number of client connections allowed per database partition.



**Configuration type**

Database manager

**Parameter type**

Configurable online

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Database Server or Connect Server with local and remote clients" (for *max\_connections*, *max\_coordagents*, *num\_initagents*, *num\_poolagents*, and also *federated\_async*, if you are using a Federated environment)

**Default [range]**

-1 and AUTOMATIC (*max\_coordagents*) [-1 and AUTOMATIC; 1–64 000 ]

A setting of -1 means that the value associated with *max\_coordagents* will be used, not the automatic setting or behavior. AUTOMATIC means that the database manager picks the value for this parameter using whatever technique works best. AUTOMATIC is an ON/OFF switch in the configuration file and is independent of the value, hence both -1 and AUTOMATIC can be the default setting.

For details, see: "Restrictions and behavior when configuring *max\_coordagents* and *max\_connections*" on page 839.

**The Concentrator**

The Concentrator is OFF when *max\_connections* is equal to or less than *max\_coordagents*. The Concentrator is ON when *max\_connections* is greater than *max\_coordagents*.

This parameter controls the maximum number of client applications that can be connected to a database partition in the instance. Typically, each application is assigned a coordinator agent. The agent facilitates the operations between the application and the database. When the default value for this parameter is used, the Concentrator feature is not activated. As a result, each agent operates within its own private memory and shares database manager and database global resources, such as the buffer pool, with other agents. When the parameter is set to a value greater than the default, the Concentrator feature is activated.

**max\_coordagents - Maximum number of coordinating agents**

This parameter is used to limit the number of coordinating agents.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable online

**Default [range]**

200, Automatic [-1; 0–64 000]

A setting of -1 translates into a value of 200.

For details, see: “Restrictions and behavior when configuring `max_coordagents` and `max_connections`” on page 839.

## The Concentrator

When the Concentrator is OFF, that is, when `max_connections` is equal to or less than `max_coordagents`, this parameter determines the maximum number of coordinating agents that can exist at one time on a server node.

One coordinating agent is acquired for each local or remote application that connects to a database or attaches to an instance. Requests that require an instance attachment include CREATE DATABASE, DROP DATABASE, and Database System Monitor commands.

When the Concentrator is ON, that is, when `max_connections` is greater than `max_coordagents`, there might be more connections than coordinator agents to service them. An application is in an active state only if there is a coordinator agent servicing it. Otherwise, the application is in an inactive state. Requests from an active application will be serviced by the database coordinator agent (and subagents in SMP or MPP configurations). Requests from an inactive application will be queued until a database coordinator agent is assigned to service the application, when the application becomes active. As a result, this parameter might be used to control the load on the system.

## max\_log - Maximum log per transaction

This parameter specifies if there is a limit to the percentage of log space that a transaction can consume, and what that limit is.

### Configuration type

Database

### Parameter type

Configurable online

### Propagation class

Immediate

### Default [range]

0 [0 — 100 ]

### Unit of measure

Percentage

If the value is not 0, this parameter indicates the percentage of primary log space that can be consumed by one transaction.

If the value is set to 0, there is no limit regarding how much space (as a percentage of total primary log space) one single transaction can consume. This was the behavior of transactions prior to Version 8.

## max\_querydegree - Maximum query degree of parallelism

This parameter specifies the maximum degree of intra-partition parallelism that is used for any SQL statement executing on this instance of the database manager. An SQL statement will not use more than this number of parallel operations within a database partition when the statement is executed.

### Configuration type

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable Online

**Propagation class**

Statement boundary

**Default [range]**

-1 (ANY) [ANY, 1 - 32 767] (ANY means system-determined)

The *intra\_parallel* configuration parameter must be set to “YES” to enable the database partition to use intra-partition parallelism for SQL statements. The *intra\_parallel* parameter is no longer required for parallel index creation.

The default value for this configuration parameter is -1. This value means that the system uses the degree of parallelism determined by the optimizer; otherwise, the user-specified value is used.

**Note:** The degree of parallelism for an SQL statement can be specified at statement compilation time using the CURRENT DEGREE special register or the DEGREE bind option.

The maximum query degree of parallelism for an active application can be modified using the SET RUNTIME DEGREE command. The actual runtime degree used is the lower of:

- *max\_querydegree* configuration parameter
- Application runtime degree
- SQL statement compilation degree

This configuration parameter applies to queries only.

## **maxapps - Maximum number of active applications**

This parameter specifies the maximum number of concurrent applications that can be connected (both local and remote) to a database. Since each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications will potentially use more memory.

**Configuration type**

Database

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

Automatic [1 - 60 000]

**Unit of measure**

Counter

Setting *maxappls* to *automatic* has the effect of allowing any number of connected applications. The database manager will dynamically allocate the resources it needs to support new applications.

If you do not want to set this parameter to *automatic*, the value of this parameter must be equal to or greater than the sum of the connected applications, plus the number of these same applications that might be concurrently in the process of completing a two-phase commit or rollback. Then add to this sum the anticipated number of indoubt transactions that might exist at any one time.

When an application attempts to connect to a database, but *maxappls* has already been reached, an error is returned to the application indicating that the maximum number of applications have been connected to the database.

In a partitioned database environment, this is the maximum number of applications that can be concurrently active against a database partition. This parameter limits the number of active applications against the database partition on a database partition server, regardless of whether the server is the coordinator node for the application or not. The catalog node in a partitioned database environment requires a higher value for *maxappls* than is the case for other types of environments because, in the partitioned database environment, every application requires a connection to the catalog node.

**Recommendation:** Increasing the value of this parameter without lowering the *maxlocks* parameter or increasing the *locklist* parameter could cause you to reach the database limit on locks (*locklist*) rather than the application limit and as a result cause pervasive lock escalation problems.

To a certain extent, the maximum number of applications is also governed by *max\_coordagents*. An application can only connect to the database, if there is an available connection (*maxappls*) as well as an available coordinating agent (*max\_coordagents*).

## maxfilop - Maximum database files open per application

This parameter specifies the maximum number of file handles that can be open for each database.

### Configuration type

Database

### Parameter type

Configurable Online

### Propagation class

Transaction boundary

### Default [range]

#### AIX, Sun, HP, and Linux 64-bit

61 440 [64 - 61 440]

#### Linux 32-bit

30 720 [64 - 30 720]

#### Windows 32-bit

32 768 [64 - 32 768]

#### Windows 64-bit

65 335 [64 - 65 335]

## Unit of measure

Counter

If opening a file causes this value to be exceeded, some files in use by this database are closed. If *maxfilop* is too small, the overhead of opening and closing files will become excessive and might degrade performance.

Both SMS table spaces and DMS table space file containers are treated as files in the database manager's interaction with the operating system, and file handles are required. More files are generally used by SMS table spaces compared to the number of containers used for a DMS file table space. Therefore, if you are using SMS table spaces, you will need a larger value for this parameter compared to what you would require for DMS file table spaces.

You can also use this parameter to ensure that the overall total of file handles used by the database manager does not exceed the operating system limit by limiting the number of handles per database to a specific number; the actual number will vary depending on the number of databases running concurrently.

## min\_dec\_div\_3 - Decimal division scale to 3

This parameter is provided as a quick way to enable a change to computation of the scale for decimal division in SQL.

### Configuration type

Database

### Parameter type

Configurable

### Default [range]

No [Yes, No ]

The *min\_dec\_div\_3* database configuration parameter changes the resulting scale of a decimal arithmetic operation involving division. It can be set to "Yes" or "No". The default value for *min\_dec\_div\_3* is "No". If the value is "No", the scale is calculated as  $31-p+s'$ . If set to "Yes", the scale is calculated as  $\text{MAX}(3, 31-p+s')$ . This causes the result of decimal division to always have a scale of at least 3. Precision is always 31.

Changing this database configuration parameter might cause changes to applications for existing databases. This can occur when the resulting scale for decimal division would be impacted by changing this database configuration parameter. Listed below are some possible scenarios that might impact applications. These scenarios should be considered before changing the *min\_dec\_div\_3* on a database server with existing databases.

- If the resulting scale of one of the view columns is changed, a view that is defined in an environment with one setting could fail with SQLCODE -344 when referenced after the database configuration parameter is changed. The message SQL0344N refers to recursive common table expressions, however, if the object name (first token) is a view, then you will need to drop the view and create it again to avoid this error.
- A static package will not change behavior until the package is rebound, either implicitly or explicitly. For example, after changing the value from NO to YES, the additional scale digits might not be included in the results until rebind occurs. For any changed static packages, an explicit REBIND command can be used to force a rebind.

- A check constraint involving decimal division might restrict some values that were previously accepted. Such rows now violate the constraint but will not be detected until one of the columns involved in the check constraint row is updated or the SET INTEGRITY statement with the IMMEDIATE CHECKED option is processed. To force checking of such a constraint, perform an ALTER TABLE statement in order to drop the check constraint and then perform an ALTER TABLE statement to add the constraint again.

**Note:** *min\_dec\_div\_3* also has the following limitations:

1. The command GET DB CFG FOR DBNAME will not display the *min\_dec\_div\_3* setting. The best way to determine the current setting is to observe the side-effect of a decimal division result. For example, consider the following statement:
 

```
VALUES (DEC(1,31,0)/DEC(1,31,5))
```

If this statement returns sqlcode SQL0419N, the database does not have *min\_dec\_div\_3* support, or it is set to "No". If the statement returns 1.000, *min\_dec\_div\_3* is set to "Yes".
2. *min\_dec\_div\_3* does not appear in the list of configuration keywords when you run the following command: ? UPDATE DB CFG

## mincommit - Number of commits to group

This parameter allows you to delay the writing of log records to disk until a minimum number of commits have been performed, helping reduce the database manager overhead associated with writing log records.

### Configuration type

Database

### Parameter type

Configurable Online

### Propagation class

Immediate

### Default [range]

1 [ 1 – 25 ]

### Unit of measure

Counter

This delay will improve performance when you have multiple applications running against a database and many commits are requested by the applications within a very short time frame.

This grouping of commits will only occur when the value of this parameter is greater than one and when the number of applications connected to the database is greater than or equal to the value of this parameter. When commit grouping is being performed, application commit requests could be held until either one second has elapsed or the number of commit requests equals the value of this parameter.

This parameter should be incremented by small amounts only; for example one (1). You should also use multi-user tests to verify that increasing the value of this parameter provides the expected results.

Changes to the value specified for this parameter take effect immediately; you do not have to wait until all applications disconnect from the database.

**Recommendation:** Increase this parameter from its default value if multiple read/write applications typically request concurrent database commits. This will result in more efficient logging file I/O as it will occur less frequently and write more log records each time it does occur.

You could also sample the number of transactions per second and adjust this parameter to accommodate the peak number of transactions per second (or some large percentage of it). Accommodating peak activity would minimize the overhead of writing log records during transaction intensive periods.

If you increase *mincommit*, you might also need to increase the *logbufsz* parameter to avoid having a full log buffer force a write during these transaction intensive periods. In this case, the *logbufsz* should be equal to:

$\text{mincommit} * (\text{log space used, on average, by a transaction})$

You can use the database system monitor to help you tune this parameter in the following ways:

- Calculating the peak number of transactions per second:  
Taking monitor samples throughout a typical day, you can determine your transaction intensive periods. You can calculate the total transactions by adding the following monitor elements:
  - *commit\_sql\_stmts* (*commit statements attempted*)
  - *rollback\_sql\_stmts* (*rollback statements attempted*)Using this information and the available timestamps, you can calculate the number of transactions per second.
- Calculating the log space used per transaction:  
Using sampling techniques over a period of time and a number of transactions, you can calculate an average of the log space used with the following monitor element:
  - *log\_space\_used* (*unit of work log space used*)

## mirrorlogpath - Mirror log path

This parameter allows you to specify a string of up to 242 bytes for the mirror log path. The string must point to a path name, and it must be a fully qualified path name, not a relative path name.

### Configuration type

Database

### Parameter type

Configurable

### Default [range]

Null [any valid path or device]

**Note:** In a single or multi-partition DB2 ESE environment, the node number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

If *mirrorlogpath* is configured, DB2 will create active log files in both the log path and the mirror log path. All log data will be written to both paths. The mirror log



path has a duplicated set of active log files, such that if there is a disk error or human error that destroys active log files on one of the paths, the database can still function.

If the mirror log path is changed, there might be log files in the old mirror log path. These log files might not have been archived, so you might need to archive these log files manually. Also, if you are running replication on this database, replication might still need the log files from before the log path change. If the database is configured with the User Exit Enable (*userexit*) database configuration parameter set to Yes, and if all the log files have been archived either by DB2 automatically or by yourself manually, then DB2 will be able to retrieve the log files to complete the replication process. Otherwise, you can copy the files from the old mirror log path to the new mirror log path.

If *logpath* or *newlogpath* specifies a raw device as the location where the log files are stored, mirror logging, as indicated by *mirrorlogpath*, is not allowed. If *logpath* or *newlogpath* specifies a file path as the location where the log files are stored, mirror logging is allowed and *mirrorlogpath* must also specify a file path.

**Recommendation:** Just like the log files, the mirror log files should be on a physical disk that does not have high I/O.

It is strongly recommended that this path be on a separate device than the primary log path.

You can use the database system monitor to track the number of I/Os related to database logging.

The following data elements return the amount of I/O activity related to database logging. You can use an operating system monitor tool to collect information about other disk I/O activity, then compare the two types of I/O activity.

- *log\_reads* (number of log pages read)
- *log\_writes* (number of log pages written).

## **mon\_heap\_sz - Database system monitor heap size**

This parameter determines the amount of the memory, in pages, to allocate for database system monitor data. Memory is allocated from the monitor heap when you perform database monitoring activities such as taking a snapshot, turning on a monitor switch, resetting a monitor, or activating an event monitor.

With Version 9.5, this database configuration parameter has a default value of *AUTOMATIC*, meaning that the monitor heap can increase as needed until the *instance\_memory* limit is reached.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable online



**Default [range]**

Automatic [0 - 60 000]

**Unit of measure**

Pages (4 KB)

**When allocated**

When the database manager is started with the db2start command

**When freed**

When the database manager is stopped with the db2stop command

A value of zero prevents the database manager from collecting database system monitor data.

**Recommendation:** The amount of memory required for monitoring activity depends on the number of monitoring applications (applications taking snapshots or event monitors), which switches are set, and the level of database activity.

If the configured memory in this heap runs out and no more unreserved memory is available in the instance shared memory region, one of the following will occur:

- When the first application connects to the database for which this event monitor is defined, an error message is written to the administration notification log.
- If an event monitor being started dynamically using the SET EVENT MONITOR statement fails, an error code is returned to your application.
- If a monitor command or API subroutine fails, an error code is returned to your application.

## **multipage\_alloc - Multipage file allocation enabled**

Multipage file allocation is used to improve insert performance. It applies to SMS table spaces only. If enabled, all SMS table spaces are affected: there is no selection possible for individual SMS table spaces.

**Configuration type**

Database

**Parameter type**

Informational

The default for the parameter is Yes: multipage file allocation is enabled.

Following database creation, this parameter cannot be set to No. Multipage file allocation cannot be disabled once it has been enabled. The db2empfa tool can be used to enable multipage file allocation for a database that currently has it disabled.

## **newlogpath - Change the database log path**

This parameter allows you to specify a string of up to 242 bytes to change the location where the log files are stored.

**Configuration type**

Database

**Parameter type**

Configurable

**Default [range]**

Null [ any valid path or device]

The string can point to either a path name or to a raw device. Note that as of DB2 Version 9, the use of raw devices for database logging is deprecated. As an alternative to using raw logs, you can use either direct input/output (DIO) or concurrent input/output (CIO).

If the string points to a path name, it must be a fully qualified path name, not a relative path name.

In a single or multi-partition DB2 ESE environment, the node number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

If you want to use replication, and your log path is a raw device, the *overflowlogpath* configuration parameter must be configured.

To specify a device, specify a string that the operating system identifies as a device. For example:

- On Windows, \\.\d: or \\.\PhysicalDisk5

**Note:** You must have Windows Version 4.0 with Service Pack 3 or later installed to be able to write logs to a device.

- On Linux and UNIX platforms, /dev/rdblog8

**Note:** You can only specify a device on AIX, Windows 2000, Windows, Solaris, HP-UX, and Linux platforms.

The new setting does not become the value of *logpath* until both of the following occur:

- The database is in a consistent state, as indicated by the *database\_consistent* parameter.
- All applications are disconnected from the database

When the first new connection is made to the database, the database manager will move the logs to the new location specified by *logpath*.

There might be log files in the old log path. These log files might not have been archived. You might need to archive these log files manually. Also, if you are running replication on this database, replication might still need the log files from before the log path change. If the database is configured with the User Exit Enable (*userexit*) database configuration parameter set to Yes, and if all the log files have been archived either by DB2 automatically or by yourself manually, then DB2 will be able to retrieve the log files to complete the replication process. Otherwise, you can copy the files from the old log path to the new log path.

If *logpath* or *newlogpath* specifies a raw device as the location where the log files are stored, mirror logging, as indicated by *mirrorlogpath*, is not allowed. If *logpath* or *newlogpath* specifies a file path as the location where the log files are stored, mirror logging is allowed and *mirrorlogpath* must also specify a file path.

**Recommendation:** Ideally, the log files will be on a physical disk which does **not** have high I/O. For instance, avoid putting the logs on the same disk as the operating system or high volume databases. This will allow for efficient logging activity with a minimum of overhead such as waiting for I/O.

You can use the database system monitor to track the number of I/Os related to database logging.

The monitor elements *log\_reads* (number of log pages read) and *log\_writes* (number of log pages written) return the amount of I/O activity related to database logging. You can use an operating system monitor tool to collect information about other disk I/O activity, then compare the two types of I/O activity.

Do not use a network or local file system that is shared as the log path for both the primary and standby databases in a DB2 High Availability Disaster Recovery (HADR) database pair. The primary and standby databases each have copies of the transaction logs – the primary database ships logs to the standby database. If the log path for both the primary and standby databases points to the same physical location, then the primary and standby database would use the same physical files for their respective copies of the logs. The database manager returns an error if the database manager detects a shared log path.

## **notifylevel - Notify level**

This parameter specifies the type of administration notification messages that are written to the administration notification log.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable Online

### **Propagation class**

Immediate

### **Default [range]**

3 [ 0 — 4 ]

On Linux and UNIX platforms, the administration notification log is a text file called *instance.nfy*. On Windows, all administration notification messages are written to the Event Log. The errors can be written by DB2, the Health Monitor, the Capture and Apply programs, and user applications.

Valid values for this parameter are:

- **0** — No administration notification messages captured. (This setting is not recommended.)
- **1** — Fatal or unrecoverable errors. Only fatal and unrecoverable errors are logged. To recover from some of these conditions, you might need assistance from DB2 service.
- **2** — Immediate action required. Conditions are logged that require immediate attention from the system administrator or the database administrator. If the condition is not resolved, it could lead to a fatal error. Notification of very significant, non-error activities (for example, recovery) might also be logged at this level. This level will capture Health Monitor alarms.

- **3** — Important information, no immediate action required. Conditions are logged that are non-threatening and do not require immediate action but might indicate a non-optimal system. This level will capture Health Monitor alarms, Health Monitor warnings, and Health Monitor attentions.
- **4** — Informational messages.

The administration notification log includes messages having values up to and including the value of *notifylevel*. For example, setting *notifylevel* to 3 will cause the administration notification log to include messages applicable to levels 1, 2, and 3.

For a user application to be able to write to the notification file or Windows Event Log, it must call the db2AdminMsgWrite API.

**Recommendation:** You might want to increase the value of this parameter to gather additional problem determination data to help resolve a problem. Note that you must set *notifylevel* to a value of 2 or higher for the Health Monitor to send any notifications to the contacts defined in its configuration.

## num\_db\_backups - Number of database backups

This parameter specifies the number of database backups to retain for a database.

### Configuration type

Database

### Parameter type

Configurable online

### Propagation class

Transaction boundary

### Default [range]

12 [1 - 32 767]

After the specified number of backups is reached, old backups are marked as expired in the recovery history file. Recovery history file entries for the table space backups and load copy backups that are related to the expired database backup are also marked as expired. When a backup is marked as expired, the physical backups can be removed from where they are stored (for example, disk, tape, TSM). The next database backup will prune the expired entries from the recovery history file.

The *rec\_his\_retentn* configuration parameter should be set to a value compatible with the value of *num\_db\_backups*. For example, if *num\_db\_backup* is set to a large value, the value for *rec\_his\_retentn* should be large enough to support that number of backups.

## num\_freqvalues - Number of frequent values retained

This parameter allows you to specify the number of “most frequent values” that will be collected when the WITH DISTRIBUTION option is specified on the RUNSTATS command.

### Configuration type

Database

### Parameter type

Configurable Online

**Propagation class**

Immediate

**Default [range]**

10 [0 - 32 767 ]

**Unit of measure**

Counter

Increasing the value of this parameter increases the amount of statistics heap (*stat\_heap\_sz*) used when collecting statistics.

The “most frequent value” statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being available to the query optimizer but requires additional catalog space. When 0 is specified, no frequent-value statistics are retained, even if you request that distribution statistics be collected.

You can also specify the number of frequent values retained as part of the RUNSTATS command at the table or the column level. by using the NUM\_FREQVALUES option. If none is specified, the *num\_freqvalues* configuration parameter value is used. Changing the number of frequent values retained through the RUNSTATS command is easier than making the change using the *num\_freqvalues* database configuration parameter.

Updating this parameter can help the optimizer obtain better selectivity estimates for some predicates (=, <, >) over data that is non-uniformly distributed. More accurate selectivity calculations might result in the choice of more efficient access plans.

After changing the value of this parameter, you need to:

- Run the RUNSTATS command again to collect statistics with the changed number of frequent values
- Rebind any packages containing static SQL or XQuery statements.

When using RUNSTATS, you have the ability to limit the number of frequent values collected at both the table level and the column level. This allows you to optimize on space occupied in the catalogs by reducing the distribution statistics for columns where they could not be exploited and yet still using the information for critical columns.

**Recommendation:** In order to update this parameter you should determine the degree of non-uniformity in the most important columns (in the most important tables) that typically have selection predicates. This can be done using an SQL SELECT statement that provides an ordered ranking of the number of occurrences of each value in a column. You should not consider uniformly distributed, unique, long, or LOB columns. A reasonable practical value for this parameter lies in the range of 10 to 100.

Note that the process of collecting frequent value statistics requires significant CPU and memory (*stat\_heap\_sz*) resources.

## **num\_initagents - Initial number of agents in pool**

This parameter determines the initial number of idle agents that are created in the agent pool at DB2START time.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable online

**Default [range]**

0 [0–64 000]

The database manager always starts the *num\_initagents* idle agents as part of the *db2start* command, except if the value of this parameter is greater than *num\_poolagents* during start up and *num\_poolagents* is not set to AUTOMATIC. In this case, the database manager only starts the *num\_poolagents* idle agents since there is no reason to start more idle agents than can be pooled.

**num\_initfenced - Initial number of fenced processes**

This parameter indicates the initial number of nonthreaded, idle db2fmp processes that are created in the db2fmp pool at START DBM time.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable online

**Default [range]**

0 [0–64 000]

Setting this parameter will reduce the initial startup time for running non-threadsafe C and Cobol routines. This parameter is ignored if *keepfenced* is not specified.

It is much more important to set *fenced\_pool* to an appropriate size for your system than to start up a number of db2fmp processes at START DBM time.

In previous versions, this parameter was known as *num\_initdaris*.

**num\_iocleaners - Number of asynchronous page cleaners**

This parameter allows you to specify the number of asynchronous page cleaners for a database.

**Configuration type**

Database

**Parameter type**

Configurable

**Default [range]**

Automatic [0 – 255 ]

**Unit of measure**

Counter

These page cleaners write changed pages from the buffer pool to disk before the space in the buffer pool is required by a database agent. As a result, database agents should not have to wait for changed pages to be written out so that they might use the space in the buffer pool. This improves overall performance of the database applications.

If you set the parameter to zero (0), no page cleaners are started and as a result, the database agents will perform all of the page writes from the buffer pool to disk. This parameter can have a significant performance impact on a database stored across many physical storage devices, since in this case there is a greater chance that one of the devices will be idle. If no page cleaners are configured, your applications might encounter periodic log full conditions.

If this parameter is set to AUTOMATIC, the number of page cleaners started will be based on the number of CPUs configured on the current machine, as well as the number of local logical database partitions in a partitioned database environment. There will always be at least one page cleaner started when this parameter is set to AUTOMATIC.

The number of page cleaners to start when this parameter is set to AUTOMATIC will be calculated using the following formula:

$$\text{number of page cleaners} = \max(\text{ceil}(\# \text{ CPUs} / \# \text{ local logical DPs}) - 1, 1)$$

This formula ensures that the number of page cleaners is distributed almost evenly across your logical database partitions, and that there are no more page cleaners than there are CPUs.

If the applications for a database primarily consist of transactions that update data, an increase in the number of cleaners will speed up performance. Increasing the page cleaners will also decrease recovery time from soft failures, such as power outages, because the contents of the database on disk will be more up-to-date at any given time.

**Recommendation:** Consider the following factors when setting the value for this parameter:

- Application type
  - If it is a query-only database that will not have updates, set this parameter to be zero (0). The exception would be if the query work load results in many TEMP tables being created (you can determine this by using the explain utility).
  - If transactions are run against the database, set this parameter to be between one and the number of physical storage devices used for the database.
- Workload

Environments with high update transaction rates might require more page cleaners to be configured.
- Buffer pool sizes

Environments with large buffer pools might also require more page cleaners to be configured.



You can use the database system monitor to help you tune this configuration parameter using information from the event monitor about write activity from a buffer pool:

- The parameter can be reduced if both of the following conditions are true:
  - *pool\_data\_writes* is approximately equal to *pool\_async\_data\_writes*
  - *pool\_index\_writes* is approximately equal to *pool\_async\_index\_writes*.
- The parameter should be increased if either of the following conditions are true:
  - *pool\_data\_writes* is much greater than *pool\_async\_data\_writes*
  - *pool\_index\_writes* is much greater than *pool\_async\_index\_writes*.

## num\_ioservers - Number of I/O servers

This parameter specifies the number of I/O servers for a database. No more than this number of I/Os for prefetching and utilities can be in progress for a database at any time.

### Configuration type

Database

### Parameter type

Configurable

### Default [range]

Automatic [1 – 255 ]

### Unit of measure

Counter

### When allocated

When an application connects to a database

### When freed

When an application disconnects from a database

I/O servers, also called prefetchers, are used on behalf of the database agents to perform prefetch I/O and asynchronous I/O by utilities such as backup and restore. An I/O server waits while an I/O operation that it initiated is in progress. Non-prefetch I/Os are scheduled directly from the database agents and as a result are not constrained by *num\_ioservers*.

If this parameter is set to AUTOMATIC, the number of prefetchers started will be based on the parallelism settings of the table spaces in the current database partition. (Parallelism settings are controlled by the DB2\_PARALLEL\_IO environment variable.) For each DMS table space, the value of this parallelism setting will be multiplied by the maximum number of containers in the table space stripe set. For each SMS table space, the value of this parallelism setting will be multiplied by the number of containers in the table space. The largest result over all table spaces in the current database partition will be used as the number of prefetchers to start. There will always be at least three prefetchers started when this parameter is set to AUTOMATIC.

When this parameter is set to AUTOMATIC, the number of prefetchers to start will be calculated at database activation time based on the following formula:

```
number of prefetchers = max(max over all table spaces
(parallelism setting * [SMS: # containers;
 DMS: max # containers in stripe set]), 3)
```



**Recommendation:** In order to fully exploit all the I/O devices in the system, a good value to use is generally one or two more than the number of physical devices on which the database resides. It is better to configure additional I/O servers, since there is minimal overhead associated with each I/O server and any unused I/O servers will remain idle.

## **num\_log\_span - Number log span**

This parameter specifies whether there is a limit to how many log files one transaction can span, and what that limit is.

**Configuration type**

Database

**Parameter type**

Configurable online

**Propagation class**

Immediate

**Default [range]**

0 [0 - 65 535 ]

**Unit of measure**

Counter

If the value is not 0, this parameter indicates the number of active log files that one active transaction is allowed to span.

If the value is set to 0, there is no limit to how many log files one single transaction can span. This was the behavior of transactions prior to Version 8.

## **num\_poolagents - Agent pool size**

This parameter sets the maximum size of the idle agent pool.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable online

**Default**

100, Automatic [-1, 0-64 000]

This configuration parameter is set to AUTOMATIC with a value of 100 as the default. A setting of -1 is still supported, and translates into a value of 100. When this parameter is set to AUTOMATIC, the database manager automatically manages the number of idle agents to pool. Typically, this means that once an agent completes its work, it is not terminated, but becomes idle for a period of time. Depending on the workload and type of agent, it might be terminated after a certain amount of time.

When using AUTOMATIC, you can still specify a value for the `num_poolagents` configuration parameter. Additional idle agents will always be pooled when the current number of pooled idle agents is less than or equal to the value that you specified.

**Examples:**

***num\_poolagents* is set to 100 and AUTOMATIC**

As an agent becomes free, it is added to the idle agent pool, where at some point the database manager evaluates whether it should be terminated or not. At the time when the database manager considers terminating the agent, if the total number of idle agents pooled is greater than 100, this agent will be terminated. If there are less than 100 idle agents, the idle agent will remain awaiting work. Using the AUTOMATIC setting allows additional idle agents beyond 100 to be pooled, which might be useful during periods of heavier system activity when the frequency of work can fluctuate on a larger scale. For cases where there are likely to be less than 100 idle agents at any given time, agents are guaranteed to be pooled. Periods of light system activity can benefit from this by incurring a less start up cost for new work.

***num\_poolagents* is configured dynamically**

If the parameter value is increased to a value greater than the number of pooled agents, the effects are immediate. As new agents become idle, they are pooled. If the parameter value is decreased, the database manager does not immediately reduce the number of agents in the pool. Rather, the pool size remains as it is, and agents are terminated as they are used and become idle again—gradually reducing the number of agents in the pool to the new limit.

**Recommendation:** For most environments the default of 0 and AUTOMATIC will be sufficient. If you have a specific workload where you feel too many agents are being created and terminated, you can consider increasing the value of `num_poolagents` while leaving the parameter set to AUTOMATIC.

## **num\_quantiles - Number of quantiles for columns**

This parameter controls the number of quantiles that will be collected when the WITH DISTRIBUTION option is specified on the RUNSTATS command.

**Configuration type**

Database

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

20 [0 - 32 767]

**Unit of measure**

Counter

Increasing the value of this parameter increases the amount of statistics heap (`stat_heap_sz`) used when collecting statistics.

The “quantile” statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being available

to the query optimizer but requires additional catalog space. When 0 or 1 is specified, no quantile statistics are retained, even if you request that distribution statistics be collected.

You can also specify the number of quantiles collected as part of the RUNSTATS command at the table or the column level, by using the NUM\_QUANTILES option. If none is specified, the *num\_quantiles* configuration parameter value is used. Changing the number of quantiles that will be collected through the RUNSTATS command is easier than making the change using the *num\_quantiles* database configuration parameter.

Updating this parameter can help obtain better selectivity estimates for range predicates over data that is non-uniformly distributed. Among other optimizer decisions, this information has a strong influence on whether an index scan or a table scan will be chosen. (It is more efficient to use a table scan to access a range of values that occur frequently and it is more efficient to use an index scan for a range of values that occur infrequently.)

After changing the value of this parameter, you need to:

- Run the RUNSTATS command again to collect statistics with the changed number of frequent values
- Rebind any packages containing static SQL or XQuery statements.

When using RUNSTATS, you have the ability to limit the number of quantiles collected at both the table level and the column level. This allows you to optimize on space occupied in the catalogs by reducing the distribution statistics for columns where they could not be exploited and yet still using the information for critical columns.

**Recommendation:** This default value for this parameter guarantees a maximum estimation error of approximately 2.5% for any single-sided range predicate (>, >=, <, or <=), and a maximum error of 5% for any BETWEEN predicate. A simple way to approximate the number of quantiles is:

- Determine the maximum error that is tolerable in estimating the number of rows of any range query, as a percentage, P.
- The number of quantiles should be approximately 100/P if most of your predicates are BETWEEN predicates, and 50/P if most of your predicates are other types of range predicates (<, <=, >, or >=).

For example, 25 quantiles should result in a maximum estimate error of 4% for BETWEEN predicates and of 2% for ">" predicates. A reasonable practical value for this parameter lies in the range of 10 to 50.

## numarchretry - Number of retries on error

This parameter specifies the number of times that DB2 is to try archiving a log file to the primary or the secondary archive directory before trying to archive log files to the failover directory.

### Configuration type

Database

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients

- Partitioned database server with local and remote clients

**Parameter type**  
Configurable Online

**Default [range]**  
5 [0 - 65 535 ]

This parameter is only used if the *failarchpath* database configuration parameter is set. If *numarchretry* is not set, DB2 will continuously retry archiving to the primary or the secondary log path.

## numdb - Maximum number of concurrently active databases including host and System i databases

This parameter specifies the number of local databases that can be concurrently active (that is, have applications connected to them), or the maximum number of different database aliases that can be cataloged on a DB2 Connect server.

**Configuration type**  
Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**  
Configurable

**Default [range]**  
UNIX 8 [1 — 256 ]

**Windows Database server with local and remote clients**  
8 [1 — 256 ]

**Windows Database server with local clients**  
3 [1 — 256 ]

**Unit of measure**  
Counter

Each database takes up storage, and an active database uses a new shared memory segment.

**Recommendation:** It is generally best to set this value to the actual number of databases that are already defined to the database manager, and to add about 10% to this value to allow for growth.

Changing the *numdb* parameter can impact the total amount of memory allocated. As a result, frequent updates to this parameter are not recommended. When updating this parameter, you should consider the other configuration parameters that can allocate memory for a database or an application connected to that database.

## numsegs - Default number of SMS containers

This parameter is deprecated in Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the DB2 Version 9.5 database manager.

**Note:** The following information applies only to pre-Version 9.5 data servers and clients.

**Configuration type**

Database

**Parameter type**

Informational

**Unit of measure**

Counter

This parameter indicates the number of containers that will be created within the default table spaces. It also shows the information used when you created your database, whether it was specified explicitly or implicitly on the CREATE DATABASE command.

This parameter only applies to SMS table spaces; the CREATE TABLESPACE statement **does not** use it in any way.

## **overflowlogpath - Overflow log path**

This parameter specifies a location for DB2 to find log files needed for a rollforward operation, as well as where to store active log files retrieved from the archive. It also gives a location for finding and storing log files needed for using db2ReadLog API.

**Configuration type**

Database

**Parameter type**

Configurable online

**Propagation class**

Immediate

**Default [range]**

NULL [any valid path ]

This parameter can be used for several functions, depending on your logging requirements.

- This parameter allows you to specify a location for DB2 to find log files that are needed for a rollforward operation. It is similar to the OVERFLOW LOG PATH option on the ROLLFORWARD command. Instead of always specifying OVERFLOW LOG PATH on every ROLLFORWARD command, you can set this configuration parameter once. However, if both are used, the OVERFLOW LOG PATH option will overwrite the *overflowlogpath* configuration parameter, for that particular rollforward operation.
- If *logsecond* is set to -1, *overflowlogpath* allows you to specify a directory for DB2 to store active log files retrieved from the archive. (Active log files have to be retrieved for rollback operations if they are no longer in the active log path). Without *overflowlogpath*, DB2 will retrieve the log files into the active log path. Using *overflowlogpath* allows you to provide additional resource for DB2 to store the retrieved log files. The benefit includes spreading the I/O cost to different disks, and allowing more log files to be stored in the active log path.
- If you need to use the db2ReadLog API (prior to DB2 V8, db2ReadLog was called sqlurlog) for replication, for example, *overflowlogpath* allows you to specify a location for DB2 to search for log files that are needed for this API. If the log file is not found (in either the active log path or the overflow log path) and the

database is configured with *userexit* enabled, DB2 will retrieve the log file. *overflowlogpath* also allows you to specify a directory for DB2 to store the log files retrieved. The benefit comes from reducing the I/O cost on the active log path and allowing more log files to be stored in the active log path.

- If you have configured a raw device for the active log path, *overflowlogpath* must be configured if you want to set *logsecond* to -1, or if you want to use the db2ReadLog API.

To set *overflowlogpath*, specify a string of up to 242 bytes. The string must point to a path name, and it must be a fully qualified path name, not a relative path name. The path name must be a directory, not a raw device.

**Note:** In a single or multi-partition DB2 ESE environment, the node number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

## pagesize - Database default page size

This parameter contains the value that was used as the default page size when the database was created. Possible values are: 4 096, 8 192, 16 384 and 32 768. When a buffer pool or table space is created in that database, the same default page size applies.

**Configuration type**  
Database

**Parameter type**  
Informational

## pckcachesz - Package cache size

This parameter is allocated out of the database shared memory, and is used for caching of sections for static and dynamic SQL and XQuery statements on a database.

**Configuration type**  
Database

**Parameter type**  
Configurable online

**Propagation class**  
Immediate

**Default [range]**

**32-bit operating systems**  
Automatic [-1, 32 - 128 000]

**64-bit operating systems**  
Automatic [-1, 32 - 2 147 483 646]

**Unit of measure**  
Pages (4 KB)

**When allocated**  
When the database is initialized

**When freed**  
When the database is shut down

In a partitioned database system, there is one package cache for each database partition.

Caching packages allows the database manager to reduce its internal overhead by eliminating the need to access the system catalogs when reloading a package; or, in the case of dynamic SQL or XQuery statements, eliminating the need for compilation. Sections are kept in the package cache until one of the following occurs:

- The database is shut down
- The package or dynamic SQL or XQuery statement is invalidated
- The cache runs out of space.

This caching of the section for a static or dynamic SQL or XQuery statement can improve performance, especially when the same statement is used multiple times by applications connected to a database. This is particularly important in a transaction processing environment.

When this parameter is set to `AUTOMATIC`, it is enabled for self tuning. When `self_tuning_mem` is set to `ON`, the memory tuner will dynamically size the memory area controlled by `pckcachesz` as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the `self_tuning_mem` configuration parameter is set to "ON.")

When this parameter is set to `-1`, the value used to calculate the page allocation is eight times the value specified for the `maxappls` configuration parameter. The exception to this occurs if eight times `maxappls` is less than 32. In this situation, the default value of `-1` will set `pckcachesz` to 32.

**Recommendation:** When tuning this parameter, you should consider whether the extra memory being reserved for the package cache might be more effective if it was allocated for another purpose, such as the buffer pool or catalog cache. For this reason, you should use benchmarking techniques when tuning this parameter.

Tuning this parameter is particularly important when several sections are used initially and then only a few are run repeatedly. If the cache is too large, memory is wasted holding copies of the initial sections.

The following monitor elements can help you determine whether you should adjust this configuration parameter:

- `pkg_cache_lookups` (package cache lookups)
- `pkg_cache_inserts` (package cache inserts)
- `pkg_cache_size_top` (package cache high water mark)
- `pkg_cache_num_overflows` (package cache overflows)

**Note:** The package cache is a working cache, so you cannot set this parameter to zero. There must be sufficient memory allocated in this cache to hold all sections of the SQL or XQuery statements currently being executed. If there is more space



allocated than currently needed, then sections are cached. These sections can simply be executed the next time they are needed without having to load or compile them.

The limit specified by the *pckcachesz* parameter is a soft limit. This limit can be exceeded, if required, if memory is still available in the database shared set. You can use the *pkg\_cache\_size\_top* monitor element to determine the largest that the package cache has grown, and the *pkg\_cache\_num\_overflows* monitor element to determine how many times the limit specified by the *pckcachesz* parameter has been exceeded.

## query\_heap\_sz - Query heap size

This parameter is deprecated in Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the DB2 Version 9.5 database manager.

**Note:** The following information applies only to pre-Version 9.5 data servers and clients.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default [range]

1 000 [2 - 524 288 ]

### Unit of measure

Pages (4 KB)

### When allocated

When an application (either local or remote) connects to the database

### When freed

When the application disconnects from the database, or detaches from the instance

This parameter specifies the **maximum** amount of memory that can be allocated for the query heap, ensuring that an application does not consume unnecessarily large amounts of virtual memory within an agent.

A query heap is used to store each query in the agent's private memory. The information for each query consists of the input and output SQLDA, the statement text, the SQLCA, the package name, creator, section number, and consistency token.

The query heap is also used for the memory allocated for blocking cursors. This memory consists of a cursor control block and a fully resolved output SQLDA.

The initial query heap allocated will be the same size as the application support layer heap, as specified by the *aslheapsz* parameter. The query heap size must be greater than or equal to two (2), and must be greater than or equal to the *aslheapsz*



parameter. If this query heap is not large enough to handle a given request, it will be reallocated to the size required by the request (not exceeding *query\_heap\_sz*). If this new query heap is more than 1.5 times larger than *aslheapsz*, the query heap will be reallocated to the size of *aslheapsz* when the query ends.

**Recommendation:** In most cases the default value will be sufficient. As a minimum, you should set *query\_heap\_sz* to a value at least five times larger than *aslheapsz*. This will allow for queries larger than *aslheapsz* and provide additional memory for three or four blocking cursors to be open at a given time.

If you have very large LOBs, you might need to increase the value of this parameter so the query heap will be large enough to accommodate those LOBs.

## **rec\_his\_retentn - Recovery history retention period**

This parameter specifies the number of days that historical information on backups will be retained.

**Configuration type**

Database

**Parameter type**

Configurable

**Default [range]**

366 [-1; 0 - 30 000]

**Unit of measure**

Days

If the recovery history file is not needed to keep track of backups, restores, and loads, this parameter can be set to a small number.

If value of this parameter is -1, the number of entries indicating full database backups (and any table space backups that are associated with the database backup) will correspond with the value specified by the *num\_db\_backups* parameter. Other entries in the recovery history file can only be pruned by explicitly using the available commands or APIs.

No matter how small the retention period, the most recent full database backup plus its restore set will always be kept, unless you use the PRUNE utility with the FORCE option.

## **release - Configuration file release level**

This parameter specifies the release level of the configuration file.

**Configuration type**

Database manager, Database

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Informational

## restore\_pending - Restore pending

This parameter states whether a RESTORE PENDING status exists in the database.

### Configuration type

Database

### Parameter type

Informational

## Restrictions and behavior when configuring max\_coordagents and max\_connections

The Version 9.5 default for the *max\_coordagents* and *max\_connections* parameters will be AUTOMATIC, with *max\_coordagents* set to 200 and *max\_connections* set to -1 (that is, set to the value of *max\_coordagents*). These settings set Concentrator to OFF.

While configuring *max\_coordagents* or *max\_connections* online, there will be some restrictions and behavior to be aware of:

- If the value of *max\_coordagents* is increased, the setting takes effect immediately and new requests will be allowed to create new coordinating agents. If the value is decreased, the number of coordinating agents will not be reduced immediately. Rather, the number of coordinating agents will no longer increase, and existing coordinating agents might terminate after finishing their current set of work, in order to reduce the overall number of coordinating agents. New requests for work that require a coordinating agent will not be serviced until the total number of coordinating agents falls below the new value and a coordinating agent becomes free.
- If the value for *max\_connections* is increased, the setting takes effect immediately and new connections previously blocked because of this parameter will be allowed. If the value is decreased, the database manager will not actively terminate existing connections; instead, new connections will not be allowed until enough of the existing connections are terminated to bring the value down below the new maximum.
- If *max\_connections* is set to -1 (default), then the maximum number of connections allowed is the same as *max\_coordagents*, and when *max\_coordagents* is updated offline or online; the maximum number of connections allowed will be updated as well.

While changing the value of *max\_coordagents* or *max\_connections* online, you cannot change it such that connection Concentrator will be turned either ON, if it's off, or OFF, if it's ON. For example, if at START DBM time *max\_coordagents* is less than *max\_connections* (Concentrator is ON), then all updates done online to these two parameters must maintain the relationship  $max\_coordagents < max\_connections$ . Similarly, if at START DBM time, *max\_coordagents* is greater than or equal to *max\_connections* (Concentrator is OFF), then all updates done online must maintain this relationship.

When you perform this type of update online, the database manager does not fail the operation, instead it defers the update. The warning SQL1362W message is returned, similar to any case when updating the database manager configuration parameters where IMMEDIATE is specified, but is not possible.

When setting *max\_coordagents* or *max\_connections* to AUTOMATIC, the following behavior can be expected:

- Both of these parameters can be configured with a starting value and an AUTOMATIC setting. For example, the following command associates a value of 200 and AUTOMATIC to the *max\_coordagents* parameter:

```
UPDATE DBM CONFIG USING max_coordagents 200 AUTOMATIC
```

These parameters will always have a value associated with them, either the value set as default, or some value that you specified. If only AUTOMATIC is specified when updating either parameter, that is, no value is specified, and the parameter previously had a value associated with it, that value would remain. Only the AUTOMATIC setting would be affected.

**Note:** When Concentrator is ON, the values assigned to these two configuration parameters are important even when the parameters are set to AUTOMATIC.

- If both parameters are set to AUTOMATIC, the database manager allows the number of connections and coordinating agents to increase as needed to suit the workload. However, the following caveats apply:
  1. When Concentrator is OFF, the database manager maintains a one-to-one ratio: for every connection there will be only *one* coordinating agent.
  2. When Concentrator is ON, the database manager tries to maintain the ratio of coordinating agents to connections set by the values in the parameters.

**Note:**

- The approach used to maintain the ratio is designed to be unintrusive and does not guarantee the ratio will be maintained perfectly. New connections are always allowed in this scenario, though they may have to wait for an available coordinating agent. New coordinating agents will be created as needed to maintain the ratio. As connections are terminated, the database manager might also terminate coordinating agents to maintain the ratio
- The database manager will not reduce the ratio that you set. The initial values of *max\_coordagents* and *max\_connections* that you set are considered a lower bound.
- The current and delayed values of both these parameters can be displayed through various means, such as CLP or APIs. The values displayed will always be the values set by the user. For example, if the following command were issued, and then 30 concurrent connections performing work on the instance were started, the displayed values for *max\_connections* and *max\_coordagents* will still be 20, AUTOMATIC:

```
UPDATE DBM CFG USING max_connections 20 AUTOMATIC,
max_coordagents 20 AUTOMATIC
```

To determine the real number of connections and coordinating agents currently running monitor elements, you can also use the Health Monitor.

- If *max\_connections* is set to AUTOMATIC with a value greater than *max\_coordagents* (so that Concentrator is ON), and *max\_coordagents* is not set to AUTOMATIC, then the database manager allows an unlimited number of connections that will use only a limited number of coordinating agents.

**Note:** Connections might have to wait for available coordinating agents.

The use of the AUTOMATIC option for the *max\_coordagents* and *max\_connections* configuration parameters is only valid in the following two scenarios:" . :

1. Both parameters are set to AUTOMATIC

2. Concentrator is enabled with *max\_connections* set to AUTOMATIC, while *max\_coordagents* is not.

All other configurations using AUTOMATIC for these parameters will be blocked and will return SQL6112N, with a reason code that explains the valid settings of AUTOMATIC for these two parameters.

## **resync\_interval - Transaction resync interval**

This parameter specifies the time interval in seconds for which a transaction manager (TM), resource manager (RM) or sync point manager (SPM) should retry the recovery of any outstanding indoubt transactions found in the TM, the RM, or the SPM. This parameter is applicable when you have transactions running in a distributed unit of work (DUOW) environment. This parameter also applies to recovery of federated database systems.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable

### **Default [range]**

180 [1 - 60 000 ]

### **Unit of measure**

Seconds

**Recommendation:** If, in your environment, indoubt transactions will not interfere with other transactions against your database, you might want to increase the value of this parameter. If you are using a DB2 Connect gateway to access DRDA2 application servers, you should consider the effect indoubt transactions might have at the application servers even though there will be no interference with local data access. If there are no indoubt transactions, the performance impact will be minimal.

## **rollfwd\_pending - Roll forward pending indicator**

This parameter informs you whether or not a roll-forward recovery is required, and where it is required.

### **Configuration type**

Database

### **Parameter type**

Informational

This parameter can indicate one of the following states:

- **DATABASE**, meaning that a roll-forward recovery procedure is required for this database
- **TABLESPACE**, meaning that one or more table spaces need to be rolled forward
- **NO**, meaning that the database is usable and no roll-forward recovery is required.

The recovery (using ROLLFORWARD DATABASE) must complete before you can access the database or table space.

## **rqrioblk - Client I/O block size**

This parameter specifies the size of the communication buffer between remote applications and their database agents on the database server. It is also used to determine the I/O block size at the Data Server Runtime Client when a blocking cursor is opened.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### **Parameter type**

Configurable

### **Default [range]**

32 767 [4 096 - 65 535 ]

### **Unit of measure**

Bytes

### **When allocated**

- When a remote client application issues a connection request for a server database
- When a blocking cursor is opened, additional blocks are opened at the client

### **When freed**

- When the remote application disconnects from the server database
- When the blocking cursor is closed

When a Data Server Runtime Client requests a connection to a remote database, this communication buffer is allocated on the client. On the database server, a communication buffer of 32 767 bytes is initially allocated, until a connection is established and the server can determine the value of *rqrioblk* at the client. Once the server knows this value, it will reallocate its communication buffer if the client's buffer is not 32 767 bytes.

The memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the Data Server Runtime Client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

**Recommendation:** For non-blocking cursors, a reason for increasing the value of this parameter would be if the data (for example, large object data) to be transmitted by a single query statement is so large that the default value is insufficient.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks might yield better performance if the

number or size of rows being transferred is large (for example, if the amount of data is greater than 4 096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks might also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application.

## **sched\_enable - Scheduler mode**

This parameter indicates whether or not the Scheduler is started by the administration server.

### **Configuration type**

DB2 Administration Server

### **Applies to**

DB2 Administration Server

### **Parameter type**

Configurable

### **Default [range]**

Off [On; Off ]

The Scheduler allows tools such as the Task Center to schedule and execute tasks at the administration server.

This parameter can only be updated from a Version 8 command line processor (CLP).

## **sched\_userid - Scheduler user ID**

This parameter specifies the user ID used by the Scheduler to connect to the tools catalog database. This parameter is only relevant if the tools catalog database is remote to the DB2 administration server.

### **Configuration type**

DB2 Administration Server

### **Applies to**

DB2 Administration Server

### **Parameter type**

Informational

### **Default [range]**

Null [any valid user ID ]

The userid and password used by the Scheduler to connect to the remote tools catalog database are specified using the db2admin command.

## **self\_tuning\_mem- Self-tuning memory**

This parameter determines whether the memory tuner will dynamically distribute available memory resources as required between memory consumers that are enabled for self-tuning.

### **Configuration type**

Database

**Parameter type**  
Configurable Online

**Propagation class**  
Immediate

**Default [range]**

**Single-database partition environments**  
ON [ON; OFF]

**Multi-database partition environments**  
OFF [ON; OFF]

In a database that is upgraded from an earlier version, *self\_tuning\_mem* will be set to OFF.

Because memory is being traded between memory consumers, there must be at least two memory consumers enabled for self-tuning in order for the memory tuner to be active. When *self\_tuning\_mem* is set to ON, but there are less than two memory consumers enabled for self-tuning, the memory tuner is inactive. (The exception to this is the sort heap memory area, which can be tuned regardless of whether other memory consumers are enabled for self-tuning or not.) When *database\_memory* is set to a numeric value, it is considered enabled for self-tuning.

This parameter is ON by default in single database partition environments. In multi-database partition environments, it is OFF by default.

The memory consumers that can be enabled for self-tuning include:

- Buffer pools (controlled by the size parameter of the ALTER BUFFERPOOL and CREATE BUFFERPOOL statements)
- Package cache (controlled by the *pckcachesz* configuration parameter)
- Lock List ( controlled by the *locklist* and *maxlocks* configuration parameters)
- Sort heap (controlled by the *sheapthres\_shr* and *sortheap* configuration parameters)
- Database shared memory (controlled by the *database\_memory* configuration parameter)

To view the current setting for this parameter, use the GET DATABASE CONFIGURATION command specifying the SHOW DETAIL parameter. The possible settings returned for this parameter are:

|                    |                                   |
|--------------------|-----------------------------------|
| Self Tuning Memory | (SELF_TUNING_MEM) = OFF           |
| Self Tuning Memory | (SELF_TUNING_MEM) = ON (Active)   |
| Self Tuning Memory | (SELF_TUNING_MEM) = ON (Inactive) |
| Self Tuning Memory | (SELF_TUNING_MEM) = ON            |

The following values indicate:

- ON (Active) - the memory tuner is actively tuning the memory on the system
- ON (Inactive) - that although the parameter is set ON, self-tuning is not occurring because there are less than two memory consumers enabled for self-tuning
- ON without (Active) or (Inactive) - from a query without the SHOW DETAIL option, or without a database connection.

In partitioned environments, the *self\_tuning\_mem* configuration parameter will only show ON (Active) for the database partition on which the tuner is running. On all other nodes *self\_tuning\_mem* will show ON (Inactive). As a result, to determine if



the memory tuner is active in a partitioned database, you must check the *self\_tuning\_mem* parameter on all database partitions.

If you have upgraded to DB2 Version 9 from an earlier version of DB2 and you plan to use the self-tuning memory feature, you should configure the following health indicators to disable threshold or state checking:

- Shared Sort Memory Utilization - `db.sort_shrmem_util`
- Percentage of sorts that overflowed - `db.spilled_sorts`
- Long Term Shared Sort Memory Utilization - `db.max_sort_shrmem_util`
- Lock List Utilization - `db.locklist_util`
- Lock Escalation Rate - `db.lock_escal_rate`
- Package Cache Hit Ratio - `db.pkgcache_hitratio`

One of the objectives of the self-tuning memory feature is to avoid having memory allocated to a memory consumer when it is not immediately required. Therefore, utilization of the memory allocated to a memory consumer might approach 100% before more memory is allocated. By disabling these health indicators, you will avoid unnecessary alerts triggered by the high rate of memory utilization by a memory consumer.

Instances created in DB2 Version 9 will have these health indicators disabled by default.

## seqdetect - Sequential detection flag

This parameter controls whether the database manager is allowed to detect sequential page reading during I/O activity.

### Configuration type

Database

### Parameter type

Configurable online

### Propagation class

Immediate

### Default [range]

Yes [Yes; No ]

The database manager can monitor I/O, and if sequential page reading is occurring the database manager can activate I/O prefetching. This type of sequential prefetch is known as *sequential detection*.

If this parameter is set to No, prefetching takes place only if the database manager knows it will be useful, for example table sorts, table scans, or list prefetch.

**Recommendation:** In most cases, you should use the default value for this parameter. Try turning sequential detection off, only if other tuning efforts were unable to correct serious query performance problems.

## sheapthres - Sort heap threshold

This parameter is an instance-wide soft limit on the total amount of memory that can be consumed by private sorts at any given time. When the total private sort memory consumption for an instance reaches this limit, the memory allocated for additional incoming private sort requests is considerably reduced.



**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- OLAP functions

**Parameter type**

Configurable online

**Propagation class**

Immediate

**Default [range]****UNIX 32-bit platforms**

0 [0 - 2 097 152 ]

**Windows 32-bit platforms**

0 [0 - 2 097 152 ]

**64-bit platforms**

0 [0 - 2 147 483 647 ]

**Unit of measure**

Pages (4 KB)

Examples of operations that use the sort heap include: sorts, hash joins, dynamic bitmaps (used for index ANDing and Star Joins), and table in-memory operations.

Explicit definition of the threshold prevents the database manager from using excessive amounts of memory for large numbers of sorts.

There is no reason to increase the value of this parameter when moving from a non-partitioned to a partitioned database environment. Once you have tuned the database and database manager configuration parameters on a single database partition environment, the same values will in most cases work well in a partitioned database environment. The only way to set this parameter to different values on different nodes or database partitions is to create more than one DB2 instance. This will require managing different DB2 databases over different database partition groups. Such an arrangement defeats the purpose of many of the advantages of a partitioned database environment.

When the instance-level *sheapthres* is set to 0, then the tracking of sort memory consumption is done at the database level only and memory allocation for sorts is constrained by the value of the database-level *sheapthres\_shr* configuration parameter.

Automatic tuning of *sheapthres\_shr* is allowed only when the database manager configuration parameter *sheapthres* is set to 0.

This parameter will not be dynamically updatable if any of the following are true:

- The starting value for *sheapthres* is 0 and the target value is a value different from 0.
- The starting value for *sheapthres* is a value different from 0 and the target value is 0.

**Recommendation:** Ideally, you should set this parameter to a reasonable multiple of the largest *sortheap* parameter you have in your database manager instance. This parameter should be **at least** two times the largest *sortheap* defined for any database within the instance.

If you are doing private sorts and your system is not memory constrained, an ideal value for this parameter can be calculated using the following steps:

1. Calculate the typical sort heap usage for each database:  
(typical number of concurrent agents running against the database)  
\* (sortheap, as defined for that database)
2. Calculate the sum of the above results, which provides the total sort heap that could be used under typical circumstances for all databases within the instance.

You should use benchmarking techniques to tune this parameter to find the proper balance between sort performance and memory usage.

You can use the database system monitor to track the sort activity, using the post threshold sorts (*post\_threshold\_sorts*) monitor element.

## **sheapthres\_shr - Sort heap threshold for shared sorts**

This parameter represents a soft limit on the total amount of database shared memory that can be used by sort memory consumers at any one time.

### **Configuration type**

Database

### **Applies to**

OLAP functions

### **Parameter type**

Configurable online

### **Propagation class**

Immediate

### **Default [range]**

#### **32-bit platforms**

Automatic [250 - 524 288]

#### **64-bit platforms**

Automatic [250 - 2 147 483 647]

### **Unit of measure**

Pages (4 KB)

There are other sort memory consumers in addition to sort, like hash join, index ANDing, block index ANDing, merge join, and in-memory tables. When the total amount of shared memory for shared sort memory consumers approaches the *sheapthres\_shr* limit, a memory throttling mechanism is activated and the future shared sort memory consumer requests might be granted less memory than requested, but will always be granted more than the minimum they need for finishing the task. Once the *sheapthres\_shr* limit is exceeded, all requests of shared sort memory from sort memory consumers will be granted the minimum amount of memory required to finish the task. When the total amount of shared memory for active shared sort memory consumers reaches this limit, subsequent sorts could fail (SQL0955C).

When the value of the database manager configuration parameter *sheapthres* is 0, all sort memory consumers for the database will use the database shared memory with *sheapthres\_shr* instead of private sort memory.

When *sheapthres\_shr* is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active. Memory consumers include SHEAPTHRES\_SHR, PCKCACHESZ, BUFFER POOL (each buffer pool counts as one), LOCKLIST, and DATABASE\_MEMORY.

Automatic tuning of *sheapthres\_shr* is allowed only when the database manager configuration parameter *sheapthres* is set to 0.

The value of *sortheap* is tuned together with the *sheapthres\_shr* parameter therefore disabling self tuning of the *sortheap* parameter automatically disables self tuning of the *sheapthres\_shr* parameter. Enabling self tuning of the *sheapthres\_shr* parameter automatically enables self tuning of the *sortheap* parameter.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the *self\_tuning\_mem* configuration parameter is set to "ON.")

When the value of this parameter is updated online, only new requests of shared-sort memory made after the update will use the new value. It is recommended that you reduce the value of *sortheap* before reducing the value of *sheapthres\_shr* and to increase the value of *sheapthres\_shr* before increasing the value of *sortheap*.

When the database manager configuration parameter *sheapthres* is greater than 0, *sheapthres\_shr* is only meaningful in two cases:

- if the *intra\_parallel* database manager configuration parameter is set to *yes*, because when *intra\_parallel* is set to *no*, there will be no shared sorts.
- if the Concentrator is on (that is, when *max\_connections* is greater than *max\_coordagents*), because sorts that use a cursor declared with the WITH HOLD option will be allocated from shared memory.

## **smtp\_server - SMTP server**

When the Scheduler is on, this parameter identifies the SMTP server that the Scheduler will use to send e-mail and pager notifications.

### **Configuration type**

DB2 Administration Server

### **Applies to**

DB2 Administration Server

### **Parameter type**

Configurable Online

### **Propagation class**

Immediate

### **Default [range]**

Null [any valid SMTP server TCP/IP hostname ]

This parameter is used by the Scheduler and the Health Monitor.

This parameter can only be updated from a Version 8 command line processor (CLP).

## **softmax - Recovery range and soft checkpoint interval**

This parameter determines the frequency of soft checkpoints and the recovery range, which help out in the crash recovery process.

### **Configuration Type**

Database

### **Parameter Type**

Configurable

### **Default [range]**

100 [ 1 – 100 \* *logprimary* ]

### **Unit of Measure**

Percentage of the size of one primary log file

This parameter is used to:

- Influence the number of logs that need to be recovered following a crash (such as a power failure). For example, if the default value is used, the database manager will try to keep the number of logs that need to be recovered to 1. If you specify 300 as the value of this parameter, the database manager will try to keep the number of logs that need to be recovered to 3.

To influence the number of logs required for crash recovery, the database manager uses this parameter to trigger the page cleaners to ensure that pages older than the specified recovery window are already written to disk.

- Determine the frequency of soft checkpoints.

At the time of a database failure resulting from an event such as a power failure, there might have been changes to the database which:

- Have not been committed, but updated the data in the buffer pool
- Have been committed, but have not been written from the buffer pool to the disk
- Have been committed and written from the buffer pool to the disk.

When a database is restarted, the log files will be used to perform a crash recovery of the database which ensures that the database is left in a consistent state (that is, all committed transactions are applied to the database and all uncommitted transactions are not applied to the database).

To determine which records from the log file need to be applied to the database, the database manager uses information recorded in a log control file. (The database manager actually maintains two copies of the log control file, `SQLOGCTL.LFH.1` and `SQLOGCTL.LFH.2`, so that if one copy is damaged, the database manager can still use the other copy.) These log control files are periodically written to disk, and, depending on the frequency of this event, the database manager might be applying log records of committed transactions or applying log records that describe changes that have already been written from the buffer pool to disk. These log records have no impact on the database, but applying them introduces some overhead into the database restart process.

The log control files are always written to disk when a log file is full, and during soft checkpoints. You can use this configuration parameter to trigger additional soft checkpoints.

The timing of soft checkpoints is based on the difference between the “current state” and the “recorded state”, given as a percentage of the *logfilesiz*. The “recorded state” is determined by the oldest valid log record indicated in the log control files on disk, while the “current state” is determined by the log control information in memory. (The oldest valid log record is the first log record that the recovery process would read.) The soft checkpoint will be taken if the value calculated by the following formula is greater than or equal to the value of this parameter:

$$( \text{space between recorded and current states} ) / \text{logfilesiz} ) * 100$$

**Recommendation:** You might want to increase or reduce the value of this parameter, depending on whether your acceptable recovery window is greater than or less than one log file. Lowering the value of this parameter will cause the database manager both to trigger the page cleaners more often and to take more frequent soft checkpoints. These actions can reduce both the number of log records that need to be processed and the number of redundant log records that are processed during crash recovery.

Note however, that more page cleaner triggers and more frequent soft checkpoints increase the overhead associated with database logging, which can impact the performance of the database manager. Also, more frequent soft checkpoints might not reduce the time required to restart a database, if you have:

- Very long transactions with few commit points.
- A very large buffer pool and the pages containing the committed transactions are not written back to disk very frequently. (Note that the use of asynchronous page cleaners can help avoid this situation.)

In both of these cases, the log control information kept in memory does not change frequently and there is no advantage in writing the log control information to disk, unless it has changed.

## sortheap - Sort heap size

This parameter defines the maximum number of private memory pages to be used for private sorts, or the maximum number of shared memory pages to be used for shared sorts.

### Configuration type

Database

### Applies to

OLAP functions

### Parameter type

Configurable Online

### Propagation class

Immediate

### Default [range]

#### 32-bit platforms

Automatic [16 - 524 288]

#### 64-bit platforms

Automatic [16 - 4 194 303]

**Unit of measure**

Pages (4 KB)

**When allocated**

As needed to perform sorts

**When freed**

When sorting is complete

If the sort is a private sort, then this parameter affects agent private memory. If the sort is a shared sort, then this parameter affects the database shared memory. Each sort has a separate sort heap that is allocated as needed, by the database manager. This sort heap is the area where data is sorted. If directed by the optimizer, a smaller sort heap than the one specified by this parameter is allocated using information provided by the optimizer.

When this parameter is set to **AUTOMATIC**, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change.

The value of **sortheap** is tuned together with the **sheapthres\_shr** parameter, therefore disabling self tuning of the **sortheap** parameter can not be done without disabling self tuning of the **sheapthres\_shr** parameter. Enabling self tuning of the **sheapthres\_shr** parameter automatically enables self tuning of the **sortheap** parameter. The **sortheap** parameter can, however, be enabled for self tuning without the **sheapthres\_shr** parameter being **AUTOMATIC**.

Automatic tuning of **sortheap** is allowed only when the database manager configuration parameter **sheapthres** is set to 0.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the **self\_tuning\_mem** configuration parameter is set to **ON**.)

**Recommendation:** When working with the sort heap, you should consider the following:

- Appropriate indexes can minimize the use of the sort heap.
- Hash join buffers, block index ANDing, merge join, table in memory and dynamic bitmaps (used for index ANDing and Star Joins) use sort heap memory. Increase the size of this parameter when these techniques are used.
- Increase the size of this parameter when frequent large sorts are required.
- When increasing the value of this parameter, you should examine whether the **sheapthres** and **sheapthres\_shr** parameters in the database manager configuration file also need to be adjusted.
- The sort heap size is used by the optimizer in determining access paths. You should consider rebinding applications (using the **REBIND** command) after changing this parameter.

When the **sortheap** value is updated, the database manager will immediately start using this new value for any current or new sorts.

## **spm\_log\_file\_sz - Sync point manager log file size**

This parameter identifies the sync point manager (SPM) log file size in 4 KB pages.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

256 [4 - 1000]

**Unit of measure**

Pages (4 KB)

The log file is contained in the `spmlog` sub-directory under `sqllib` and is created the first time SPM is started.

**Recommendation:** The sync point manager log file size should be large enough to maintain performance, but small enough to prevent wasted space. The size required depends on the number of transactions using protected conversations, and how often COMMIT or ROLLBACK is issued.

To change the size of the SPM log file:

1. Determine that there are no indoubt transactions by using the LIST DRDA INDOUBT TRANSACTIONS command.
2. If there are none, stop the database manager.
3. Update the database manager configuration with a new SPM log file size.
4. Go to the `$HOME/sqllib` directory and issue `rm -fr spmlog` to delete the current SPM log. (Note: This shows the AIX command. Other systems might require a different remove or delete command.)
5. Start the database manager. A new SPM log of the specified size is created during the startup of the database manager.

## **spm\_log\_path - Sync point manager log file path**

This parameter specifies the directory where the sync point manager (SPM) logs are written.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

`sqllib/spmlog` [any valid path or device]

By default, the logs are written to the `sqllib/spmlog` directory, which, in a high-volume transaction environment, can cause an I/O bottleneck. Use this parameter to have the SPM log files placed on a faster disk than the current `sqllib/spmlog` directory. This allows for better concurrency among the SPM agents.

## spm\_max\_resync - Sync point manager resync agent limit

This parameter identifies the number of agents that can simultaneously perform resync operations.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default [range]

20 [10 — 256 ]

## spm\_name - Sync point manager name

This parameter identifies the name of the sync point manager (SPM) instance to the database manager.

### Configuration type

Database manager

### Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable

### Default

Derived from the TCP/IP hostname

## stat\_heap\_sz - Statistics heap size

This parameter indicates the *maximum* size of the heap used in collecting statistics using the RUNSTATS command.

With Version 9.5, this database configuration parameter has a default value of *AUTOMATIC*, meaning that it increases as needed until either the *appl\_memory* limit is reached, or the *instance\_memory* limit is reached.

### Configuration type

Database

### Parameter type

Configurable online

### Default [range]

Automatic [1 096 - 524 288]

### Unit of measure

Pages (4 KB)

### When allocated

When the RUNSTATS utility is started



**When freed**

When the RUNSTATS utility is completed

**Recommendation:** The default setting of AUTOMATIC is recommended.

## stmtheap - Statement heap size

This parameter specifies the size of the statement heap, which is used as a work space for the SQL or XQuery compiler during compilation of an SQL or XQuery statement.

With Version 9.5, this database configuration parameter has a default value of AUTOMATIC, meaning that it increases as needed until either the *appl\_memory* limit is reached, or the *instance\_memory* limit is reached.

**Configuration type**

Database

**Parameter type**

Configurable Online

**Propagation class**

Statement boundary

**Default [range]**

**For both 32-bit and 64-bit platforms**

Automatic [128 - 524 288]

**Unit of measure**

Pages (4 KB)

**When allocated**

For each statement during precompiling or binding

**When freed**

When precompiling or binding of each statement is complete

This area does not stay permanently allocated, but is allocated and released for every SQL or XQuery statement handled. Note that for dynamic SQL or XQuery statements, this work area will be used during execution of your program; whereas, for static SQL or XQuery statements, it is used during the bind process but not during program execution.

**Recommendation:** In most cases the default AUTOMATIC setting for this parameter is acceptable. When set to AUTOMATIC, there is an internal limit on the total amount of memory allocated during the dynamic programming join enumeration phase of compilation. If this limit is exceeded, the statement is compiled using greedy join enumeration, and is only limited by the amount of remaining *appl\_memory* or *instance\_memory*, or both. If your application is receiving SQL0437W warnings, and the runtime performance for your query is not acceptable, you might want to consider setting a sufficiently large manual *stmtheap* value to ensure that dynamic join enumeration is always used.

**Note:** Dynamic join enumeration occurs only at optimization classes 3 and higher (5 is the default).

## territory - Database territory

This parameter shows the territory used to create the database. *territory* is used by the database manager when processing data that is territory sensitive.

**Configuration type**

Database

**Parameter type**

Informational

**tm\_database - Transaction manager database name**

This parameter identifies the name of the transaction manager (TM) database for each DB2 instance.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default [range]**

1ST\_CONN [any valid database name]

A TM database can be:

- A local DB2 database
- A remote DB2 database that does not reside on a host or AS/400 system
- A DB2 for OS/390 Version 5 database if accessed via TCP/IP and the sync point manager (SPM) is not used.

The TM database is a database that is used as a logger and coordinator, and is used to perform recovery for indoubt transactions.

You can set this parameter to **1ST\_CONN**, which will set the TM database to be the first database to which a user connects.

**Recommendation:** For simplified administration and operation, you might want to create a few databases over a number of instances and use these databases exclusively as TM databases.

**toolscat\_db - Tools catalog database**

This parameter indicates the tools catalog database used by the Scheduler.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable

**Default [range]**

Null [any valid database alias ]

This database must be in the database directory of the instance specified by *toolscat\_inst*.

This parameter can only be updated from a Version 8 command line processor (CLP).

### **toolscat\_inst - Tools catalog database instance**

This parameter indicates the instance name that is used by the Scheduler, along with toolscat\_db and toolscat\_schema, to identify the tools catalog database.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable

**Default [range]**

Null [any valid instance ]

The tools catalog database contains task information created by the Task Center and the Control Center. The tools catalog database must be listed in the database directory of the instance specified by this configuration parameter. The database can be local or remote. If the tools catalog database is local, the instance must be configured for TCP/IP. If the database is remote, the database partition cataloged in the database directory must be a TCP/IP node.

This parameter can only be updated from a Version 8 command line processor (CLP).

### **toolscat\_schema - Tools catalog database schema**

This parameter indicates the schema of the tools catalog database used by the Scheduler.

**Configuration type**

DB2 Administration Server

**Applies to**

DB2 Administration Server

**Parameter type**

Configurable

**Default [range]**

Null [any valid schema ]

The schema is used to uniquely identify a set of tools catalog tables and views within the database.

This parameter can only be updated from a Version 8 command line processor (CLP).

### **tp\_mon\_name - Transaction processor monitor name**

This parameter identifies the name of the transaction processing (TP) monitor product being used.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable

**Default**

No default

**Valid values**

- CICS<sup>®</sup>
- MQ
- ENCINA
- CB
- SF
- TUXEDO
- TOPEND
- blank or some other value (for UNIX and Windows; no other possible values for Solaris or SINIX)
- If applications are run in a WebSphere<sup>®</sup> Enterprise Server Edition CICS environment, this parameter should be set to "CICS"
- If applications are run in a WebSphere Enterprise Server Edition Encina<sup>®</sup> environment, this parameter should be set to "ENCINA"
- If applications are run in a WebSphere Enterprise Server Edition Component Broker environment, this parameter should be set to "CB"
- If applications are run in an IBM MQSeries<sup>®</sup> environment, this parameter should be set to "MQ"
- If applications are run in a BEA Tuxedo environment, this parameter should be set to "TUXEDO"
- If applications are run in an IBM San Francisco environment, this parameter should be set to "SF".

**IBM WebSphere EJB and Microsoft<sup>®</sup> Transaction Server** users do not need to configure any value for this parameter.

If none of the above products are being used, this parameter should not be configured but left blank.

In previous versions of IBM DB2 on Windows, this parameter contained the path and name of the DLL which contained the XA Transaction Manager's functions *ax\_reg* and *ax\_unreg*. This format is still supported. If the value of this parameter does not match any of the above TP Monitor names, it will be assumed that the value is a library name which contains the *ax\_reg* and *ax\_unreg* functions. This is true for UNIX and Windows environments.

**TXSeries<sup>®</sup> CICS and Encina Users:** In previous versions of this product on Windows it was required to configure this parameter as "libEncServer:C" or "libEncServer:E". While this is still supported, it is no longer required. Configuring the parameter as "CICS" or "ENCINA" is sufficient.

**MQSeries Users:** In previous versions of this product on Windows it was required to configure this parameter as "mqmax". While this is still supported, it is no longer required. Configuring the parameter as "MQ" is sufficient.

**Component Broker Users:** In previous versions of this product on Windows it was required to configure this parameter as "somtrx1i". While this is still supported, it is no longer required. Configuring the parameter as "CB" is sufficient.

**San Francisco Users:** In previous versions of this product on Windows it was required to configure this parameter as "ibmsfDB2". While this is still supported, it is no longer required. Configuring the parameter as "SF" is sufficient.

The maximum length of the string that can be specified for this parameter is 19 characters.

It is also possible to configure this information in IBM DB2 Version 9.1's XA OPEN string. If multiple Transaction Processing Monitors are using a single DB2 instance, then it will be required to use this capability.

## **trackmod - Track modified pages enable**

This parameter specifies whether the database manager will track database modifications so that the backup utility can detect which subsets of the database pages must be examined by an incremental backup and potentially included in the backup image.

### **Configuration type**

Database

### **Parameter type**

Configurable

### **Default [range]**

No [Yes, No ]

After setting this parameter to "Yes", you must take a full database backup in order to have a baseline against which incremental backups can be taken. Also, if this parameter is enabled and if a table space is created, then a backup must be taken which contains that table space. This backup could be either a database backup or a table space backup. Following the backup, incremental backups will be permitted to contain this table space.

## **trust\_clntauth - Trusted clients authentication**

This parameter specifies whether a trusted client is authenticated at the server or the client when the client provides a userid and password combination for a connection. This parameter (and *trust\_allclnts*) is only active if the *authentication* parameter is set to CLIENT. If a user ID and password are not provided, the client is assumed to have validated the user, and no further validation is performed at the server.

### **Configuration type**

Database manager

### **Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**  
Configurable

**Default [range]**  
CLIENT [CLIENT, SERVER]

If this parameter is set to CLIENT (the default), the trusted client can connect without providing a user ID and password combination, and the assumption is that the operating system has already authenticated the user. If it is set to SERVER, the user ID and password will be validated at the server.

The numeric value for CLIENT is 0. The numeric value for SERVER is 1.

## **tsm\_mgmtclass - Tivoli Storage Manager management class**

The Tivoli Storage Manager management class determines how the TSM server should manage the backup versions of the objects being backed up.

**Configuration type**  
Database

**Parameter type**  
Configurable

**Default [range]**  
Null [any string]

The default is that there is no DB2-specified management class.

When performing any TSM backup, before using the management class specified by the database configuration parameter, TSM first attempts to bind the backup object to the management class specified in the INCLUDE-EXCLUDE list found in the TSM client options file. If a match is not found, the default TSM management class specified on the TSM server will be used. TSM will then rebind the backup object to the management class specified by the database configuration parameter.

Thus, the default management class, as well as the management class specified by the database configuration parameter, must contain a backup copy group, or the backup operation will fail.

## **tsm\_nodename - Tivoli Storage Manager node name**

This parameter is used to override the default setting for the node name associated with the Tivoli Storage Manager (TSM) product.

**Configuration type**  
Database

**Parameter type**  
Configurable online

**Propagation class**  
Statement boundary

**Default [range]**  
Null [any string]

The node name is needed to allow you to restore a database that was backed up to TSM from another node.

The default is that you can only restore a database from TSM on the same node from which you did the backup. It is possible for the *tsm\_nodename* to be overridden during a backup done through DB2 (for example, with the BACKUP DATABASE command).

## **tsm\_owner - Tivoli Storage Manager owner name**

This parameter is used to override the default setting for the owner associated with the Tivoli Storage Manager (TSM) product.

**Configuration type**

Database

**Parameter type**

Configurable online

**Propagation class**

Statement boundary

**Default [range]**

Null [any string]

The owner name is needed to allow you to restore a database that was backed up to TSM from another node. It is possible for the *tsm\_owner* to be overridden during a backup done through DB2 (for example, with the BACKUP DATABASE command).

**Note:** The owner name is case sensitive.

The default is that you can only restore a database from TSM on the same node from which you did the backup.

## **tsm\_password - Tivoli Storage Manager password**

This parameter is used to override the default setting for the password associated with the Tivoli Storage Manager (TSM) product.

**Configuration type**

Database

**Parameter type**

Configurable online

**Propagation class**

Statement boundary

**Default [range]**

Null [any string]

The password is needed to allow you to restore a database that was backed up to TSM from another node.

**Note:** If the *tsm\_nodename* is overridden during a backup done with DB2 (for example, with the BACKUP DATABASE command), the *tsm\_password* might also have to be set.

The default is that you can only restore a database from TSM on the same node from which you did the backup. It is possible for the *tsm\_nodename* to be overridden during a backup done with DB2.

## **user\_exit\_status - User exit status indicator**

If set to On, this parameter indicates that the database manager is enabled for roll-forward recovery and that the user exit program will be used to archive and retrieve log files when called by the database manager.

### **Configuration type**

Database

### **Parameter type**

Informational

## **userexit - User exit enable**

This parameter is deprecated in Version 9.5, but is still being used by pre-Version 9.5 data servers and clients. Any value specified for this configuration parameter will be ignored by the DB2 Version 9.5 database manager.

**Note:** The following information applies only to pre-Version 9.5 data servers and clients.

If this parameter is enabled, log retention logging is performed regardless of how the *logretain* parameter is set. This parameter also indicates that a user exit program should be used to archive and retrieve the log files.

### **Configuration type**

Database

### **Parameter type**

Configurable

### **Default [range]**

Off [On; Off]

Log files are archived when the log file is full. They are retrieved when the ROLLFORWARD utility needs to use them to restore a database.

After *logretain*, or *userexit*, or both of these parameters are enabled, you must make a full backup of the database. This state is indicated by the *backup\_pending* flag parameter.

If both of these parameters are de-selected, roll-forward recovery becomes unavailable for the database because logs will no longer be retained. In this case, the database manager deletes all log files in the *logpath* directory (including online archive log files), allocates new active log files, and reverts to circular logging.

## **util\_heap\_sz - Utility heap size**

This parameter indicates the maximum amount of memory that can be used simultaneously by the BACKUP, RESTORE, and LOAD (including load recovery) utilities.

### **Configuration type**

Database

### **Parameter type**

Configurable online

### **Propagation class**

Immediate



**Default [range]**  
5000 [16 - 524 288 ]

**Unit of measure**  
Pages (4 KB)

**When allocated**  
As required by the database manager utilities

**When freed**  
When the utility no longer needs the memory

**Recommendation:** Use the default value unless your utilities run out of space, in which case you should increase this value. If memory on your system is constrained, you might want to lower the value of this parameter to limit the memory used by the database utilities. If the parameter is set too low, you might not be able to run utilities concurrently. You should update this parameter dynamically as needed. For a small number of utilities, set this parameter to a small value. For a large number of utilities, or for memory intensive utilities, you should set this parameter to a larger value.

## **util\_impact\_lim - Instance impact policy**

This parameter allows the database administrator (DBA) to limit the performance degradation of a throttled utility on the workload.

**Configuration type**  
Database manager

**Applies to**

- Database server with local clients
- Database server with local and remote clients
- Partitioned database server with local and remote clients

**Parameter type**  
Configurable Online

**Propagation class**  
Immediate

**Default [range]**  
10 [1 - 100 ]

**Unit of measure**  
Percentage of allowable impact on workload

If the performance degradation is limited, the DBA can then run online utilities during critical production periods, and be guaranteed that the performance impact on production work will be within acceptable limits.

For example, a DBA specifying a *util\_impact\_lim* (impact policy) value of 10 can expect that a throttled backup invocation will not impact the workload by more than 10 percent.

If *util\_impact\_lim* is 100, no utility invocations will be throttled. In this case, the utilities can have an arbitrary (and undesirable) impact on the workload. If *util\_impact\_lim* is set to a value that is less than 100, it is possible to invoke utilities in throttled mode. To run in throttled mode, a utility must also be invoked with a non-zero priority.

**Recommendation:** Most users will benefit from setting *util\_impact\_lim* to a low value (for example, between 1 and 10).

A throttled utility will usually take longer to complete than an unthrottled utility. If you find that a utility is running for an excessively long time, increase the value of *util\_impact\_lim*, or disable throttling altogether by setting *util\_impact\_lim* to 100.

## vendoropt - Vendor options

This parameter specifies additional parameters that DB2 might need to use to communicate with storage systems during backup, restore, or load copy operations.

### Configuration type

Database

### Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

### Parameter type

Configurable Online

### Default [range]

Null [ ]

### Restriction

You cannot use the **vendoropt** configuration parameter to specify vendor-specific options for snapshot backup or restore operations. You must use the **OPTIONS** parameter of the backup or restore utilities instead.

## wlm\_collect\_int - Workload management collection interval configuration parameter

This parameter specifies a collect and reset interval, in minutes, for workload management (WLM) statistics.

Every *x wlm\_collect\_int* minutes, (where *x* is the value of the *wlm\_collect\_int* parameter) all workload management statistics are collected and sent to any active statistics event monitor; then the statistics are reset. If an active event monitor exists, depending on how it was created, the statistics are written either to file or to a table. If it does not exist, the statistics are only reset and not collected.

The collect and reset process is initiated from the catalog partition. The *wlm\_collect\_int* parameter must be specified on the catalog partition. It is not used on other partitions.

### Configuration type

Database

### Parameter type

Configurable online

### Default [range]

0 [0 (no collection performed), 5 - 32 767]

The workload management statistics collected by a statistics event monitor can be used to monitor both short term and long term system behavior. A small interval

can be used to obtain both short term and long term system behavior because the results can be merged together to obtain long term behavior. However, having to manually merge the results from different intervals complicates the analysis. If it's not required, a small interval unnecessarily increases the overhead. Therefore, reduce the interval to capture shorter term behavior, and increase the interval to reduce overhead when only analysis of long term behavior is sufficient.

The interval needs to be customized per database, not for each SQL request, or command invocation, or application. There are no other configuration parameters that need to be considered.

**Note:** All WLM statistics table functions return statistics that have been accumulated since the last time the statistics were reset. The statistics will be reset regularly on the interval specified by this configuration parameter.

---

## Appendix B. Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command line help
  - Command help
  - Message help

**Note:** The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at [ibm.com](http://ibm.com).

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks publications online at [ibm.com](http://ibm.com). Access the DB2 Information Management software library site at <http://www.ibm.com/software/data/sw-library/>.

### Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an e-mail to [db2docs@ca.ibm.com](mailto:db2docs@ca.ibm.com). The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this e-mail address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

---

## DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at [www.ibm.com/shop/publications/order](http://www.ibm.com/shop/publications/order). English and translated DB2 Version 9.7 manuals in PDF format can be downloaded from [www.ibm.com/support/docview.wss?rs=71&uid=swg2700947](http://www.ibm.com/support/docview.wss?rs=71&uid=swg2700947).

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

**Note:** The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

*Table 43. DB2 technical information*

| <b>Name</b>                                                                          | <b>Form Number</b> | <b>Available in print</b> | <b>Last updated</b> |
|--------------------------------------------------------------------------------------|--------------------|---------------------------|---------------------|
| <i>Administrative API Reference</i>                                                  | SC27-2435-00       | Yes                       |                     |
| <i>Administrative Routines and Views</i>                                             | SC27-2436-00       | No                        |                     |
| <i>Call Level Interface Guide and Reference, Volume 1</i>                            | SC27-2437-00       | Yes                       |                     |
| <i>Call Level Interface Guide and Reference, Volume 2</i>                            | SC27-2438-00       | Yes                       |                     |
| <i>Command Reference</i>                                                             | SC27-2439-00       | Yes                       |                     |
| <i>Data Movement Utilities Guide and Reference</i>                                   | SC27-2440-00       | Yes                       |                     |
| <i>Data Recovery and High Availability Guide and Reference</i>                       | SC27-2441-00       | Yes                       |                     |
| <i>Database Administration Concepts and Configuration Reference</i>                  | SC27-2442-00       | Yes                       |                     |
| <i>Database Monitoring Guide and Reference</i>                                       | SC27-2458-00       | Yes                       |                     |
| <i>Database Security Guide</i>                                                       | SC27-2443-00       | Yes                       |                     |
| <i>DB2 Text Search Guide</i>                                                         | SC27-2459-00       | Yes                       |                     |
| <i>Developing ADO.NET and OLE DB Applications</i>                                    | SC27-2444-00       | Yes                       |                     |
| <i>Developing Embedded SQL Applications</i>                                          | SC27-2445-00       | Yes                       |                     |
| <i>Developing Java Applications</i>                                                  | SC27-2446-00       | Yes                       |                     |
| <i>Developing Perl, PHP, Python, and Ruby on Rails Applications</i>                  | SC27-2447-00       | No                        |                     |
| <i>Developing User-defined Routines (SQL and External)</i>                           | SC27-2448-00       | Yes                       |                     |
| <i>Getting Started with Database Application Development</i>                         | GI11-9410-00       | Yes                       |                     |
| <i>Getting Started with DB2 Installation and Administration on Linux and Windows</i> | GI11-9411-00       | Yes                       |                     |

Table 43. DB2 technical information (continued)

| <b>Name</b>                                                                             | <b>Form Number</b> | <b>Available in print</b> | <b>Last updated</b> |
|-----------------------------------------------------------------------------------------|--------------------|---------------------------|---------------------|
| <i>Globalization Guide</i>                                                              | SC27-2449-00       | Yes                       |                     |
| <i>Installing DB2 Servers</i>                                                           | GC27-2455-00       | Yes                       |                     |
| <i>Installing IBM Data Server Clients</i>                                               | GC27-2454-00       | No                        |                     |
| <i>Message Reference Volume 1</i>                                                       | SC27-2450-00       | No                        |                     |
| <i>Message Reference Volume 2</i>                                                       | SC27-2451-00       | No                        |                     |
| <i>Net Search Extender Administration and User's Guide</i>                              | SC27-2469-00       | No                        |                     |
| <i>SQL Procedural Languages: Application Enablement and Support</i>                     | SC23-9838-00       | Yes                       |                     |
| <i>Partitioning and Clustering Guide</i>                                                | SC27-2453-00       | Yes                       |                     |
| <i>pureXML Guide</i>                                                                    | SC27-2465-00       | Yes                       |                     |
| <i>Query Patroller Administration and User's Guide</i>                                  | SC27-2467-00       | No                        |                     |
| <i>Spatial Extender and Geodetic Data Management Feature User's Guide and Reference</i> | SC27-2468-00       | No                        |                     |
| <i>SQL Procedural Language Guide</i>                                                    | SC27-2470-00       | Yes                       |                     |
| <i>SQL Reference, Volume 1</i>                                                          | SC27-2456-00       | Yes                       |                     |
| <i>SQL Reference, Volume 2</i>                                                          | SC27-2457-00       | Yes                       |                     |
| <i>Troubleshooting and Tuning Database Performance</i>                                  | SC27-2461-00       | Yes                       |                     |
| <i>Upgrading to DB2 Version 9.7</i>                                                     | SC27-2452-00       | Yes                       |                     |
| <i>Visual Explain Tutorial</i>                                                          | SC27-2462-00       | No                        |                     |
| <i>What's New for DB2 Version 9.7</i>                                                   | SC27-2463-00       | Yes                       |                     |
| <i>Workload Manager Guide and Reference</i>                                             | SC27-2464-00       | Yes                       |                     |
| <i>XQuery Reference</i>                                                                 | SC27-2466-00       | No                        |                     |

Table 44. DB2 Connect-specific technical information

| <b>Name</b>                                                    | <b>Form Number</b> | <b>Available in print</b> | <b>Last updated</b> |
|----------------------------------------------------------------|--------------------|---------------------------|---------------------|
| <i>Installing and Configuring DB2 Connect Personal Edition</i> | SC27-2432-00       | Yes                       |                     |

Table 44. DB2 Connect-specific technical information (continued)

| Name                                                  | Form Number  | Available in print | Last updated |
|-------------------------------------------------------|--------------|--------------------|--------------|
| <i>Installing and Configuring DB2 Connect Servers</i> | SC27-2433-00 | Yes                |              |
| <i>DB2 Connect User's Guide</i>                       | SC27-2434-00 | Yes                |              |

Table 45. Information Integration technical information

| Name                                                                                          | Form Number  | Available in print | Last updated |
|-----------------------------------------------------------------------------------------------|--------------|--------------------|--------------|
| <i>Information Integration: Administration Guide for Federated Systems</i>                    | SC19-1020-02 | Yes                |              |
| <i>Information Integration: ASNCLP Program Reference for Replication and Event Publishing</i> | SC19-1018-04 | Yes                |              |
| <i>Information Integration: Configuration Guide for Federated Data Sources</i>                | SC19-1034-02 | No                 |              |
| <i>Information Integration: SQL Replication Guide and Reference</i>                           | SC19-1030-02 | Yes                |              |
| <i>Information Integration: Introduction to Replication and Event Publishing</i>              | SC19-1028-02 | Yes                |              |

## Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation DVD* are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the *DB2 PDF Documentation DVD* can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the *DB2 PDF Documentation DVD* are available in print.

**Note:** The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7>.

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at <http://www.ibm.com/shop/>

publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.

- To order printed DB2 books from your local IBM representative:
  1. Locate the contact information for your local representative from one of the following Web sites:
    - The IBM directory of world wide contacts at [www.ibm.com/planetwide](http://www.ibm.com/planetwide)
    - The IBM Publications Web site at <http://www.ibm.com/shop/publications/order>. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
  2. When you call, specify that you want to order a DB2 publication.
  3. Provide your representative with the titles and form numbers of the books that you want to order. For titles and form numbers, see "DB2 technical library in hardcopy or PDF format" on page 865.

---

## Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

To start SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, ? 08003 displays help for the 08003 SQL state, and ? 08 displays help for the 08 class code.

---

## Accessing different versions of the DB2 Information Center

For DB2 Version 9.7 topics, the DB2 Information Center URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/>

For DB2 Version 9.5 topics, the DB2 Information Center URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/>

For DB2 Version 9 topics, the DB2 Information Center URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9/>

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: <http://publib.boulder.ibm.com/infocenter/db2luw/v8/>

---

## Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

- To display topics in your preferred language in the Internet Explorer browser:



1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
2. Ensure your preferred language is specified as the first entry in the list of languages.
  - To add a new language to the list, click the **Add...** button.

**Note:** Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.

- To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.
- To display topics in your preferred language in a Firefox or Mozilla browser:
    1. Select the button in the **Languages** section of the **Tools** —> **Options** —> **Advanced** dialog. The Languages panel is displayed in the Preferences window.
    2. Ensure your preferred language is specified as the first entry in the list of languages.
      - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
      - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
    3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you must also change the regional settings of your operating system to the locale and language of your choice.

---

## Updating the DB2 Information Center installed on your computer or intranet server

A locally installed DB2 Information Center must be updated periodically.

### Before you begin

A DB2 Version 9.7 Information Center must already be installed. For details, see the “Installing the DB2 Information Center using the DB2 Setup wizard” topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

### About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates - updates existing Information Center features and languages. An additional benefit of automatic updates is that the Information Center is unavailable for a minimal period of time during the update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates - should be used when you want to add features or languages during the update process. For example, a local Information Center was originally installed with both English and French languages, and now you want

to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process.

### Procedure

This topic details the process for automatic updates. For manual update instructions, see the “Manually updating the DB2 Information Center installed on your computer or intranet server” topic.

To automatically update the DB2 Information Center installed on your computer or intranet server:

1. On Linux operating systems,
  - a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the /opt/ibm/db2ic/V9.7 directory.
  - b. Navigate from the installation directory to the doc/bin directory.
  - c. Run the ic-update script:

```
ic-update
```
2. On Windows operating systems,
  - a. Open a command window.
  - b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the <Program Files>\IBM\DB2 Information Center\Version 9.7 directory, where <Program Files> represents the location of the Program Files directory.
  - c. Navigate from the installation directory to the doc\bin directory.
  - d. Run the ic-update.bat file:

```
ic-update.bat
```

### Results

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center updates were not available, a message is added to the log. The log file is located in doc\eclipse\configuration directory. The log file name is a randomly generated number. For example, 1239053440785.log.

---

## Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

Updating your locally-installed DB2 Information Center manually requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .

2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

**Note:** If your environment requires installing the DB2 Information Center updates on a machine that is not connected to the internet, mirror the update site to a local file system using a machine that is connected to the internet and has the DB2 Information Center installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.

If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.

3. Stop the stand-alone Information Center, and restart the DB2 Information Center on your computer.

**Note:** On Windows 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center.
  - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click **DB2 Information Center** service and select **Stop**.

- On Linux, enter the following command:

```
/etc/init.d/db2icdv97 stop
```

2. Start the Information Center in stand-alone mode.


- On Windows:
  - a. Open a command window.
  - b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the <Program Files>\IBM\DB2 Information Center\Version 9.7 directory, where <Program Files> represents the location of the Program Files directory.
  - c. Navigate from the installation directory to the doc\bin directory.
  - d. Run the help\_start.bat file:

```
help_start.bat
```

- On Linux:
  - a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the /opt/ibm/db2ic/V9.7 directory.
  - b. Navigate from the installation directory to the doc/bin directory.
  - c. Run the help\_start script:

```
help_start
```

The systems default Web browser opens to display the stand-alone Information Center.

3. Click the **Update** button (). (JavaScript must be enabled in your browser.) On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check the selections you want to install, then click **Install Updates**.

5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
  - On Windows, navigate to the installation directory's doc\bin directory, and run the help\_end.bat file:
 

```
help_end.bat
```

**Note:** The help\_end batch file contains the commands required to safely stop the processes that were started with the help\_start batch file. Do not use Ctrl-C or any other method to stop help\_start.bat.
  - On Linux, navigate to the installation directory's doc/bin directory, and run the help\_end script:
 

```
help_end
```

**Note:** The help\_end script contains the commands required to safely stop the processes that were started with the help\_start script. Do not use any other method to stop the help\_start script.
7. Restart the DB2 Information Center.
  - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click **DB2 Information Center** service and select **Start**.
  - On Linux, enter the following command:
 

```
/etc/init.d/db2icdv97 start
```

The updated DB2 Information Center displays the new and updated topics.

---

## DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

### Before you begin

You can view the XHTML version of the tutorial from the Information Center at <http://publib.boulder.ibm.com/infocenter/db2help/>.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

### DB2 tutorials

To view the tutorial, click the title.

**“pureXML” in *pureXML Guide***

Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

**“Visual Explain” in *Visual Explain Tutorial***

Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

---

## DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 database products.

### **DB2 documentation**

Troubleshooting information can be found in the *DB2 Troubleshooting Guide* or the Database fundamentals section of the *DB2 Information Center*. There you will find information about how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 database products.

### **DB2 Technical Support Web site**

Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the DB2 Technical Support Web site at [http://www.ibm.com/software/data/db2/support/db2\\_9/](http://www.ibm.com/software/data/db2/support/db2_9/)

---

## **Terms and Conditions**

Permissions for the use of these publications is granted subject to the following terms and conditions.

**Personal use:** You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

**Commercial use:** You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

---

## Appendix C. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

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 grant 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 character set (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/region 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 that has been exchanged, should contact:

IBM Canada Limited  
Office of the Lab Director  
8200 Warden Avenue  
Markham, Ontario  
L6G 1C7  
CANADA

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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain 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.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *\_enter the year or years\_*. All rights reserved.

## Trademarks

IBM, the IBM logo, and [ibm.com](http://ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos 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.
- 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.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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





---

# Index

## A

- administration notification log 713
- agent pool size configuration parameter 830
- agent process
  - applheapsz configuration parameter 754
  - aslheapsz configuration parameter 755
  - priority of agents configuration parameter 751
- agent\_stack\_sz database manager configuration parameter 749
- agentpri database manager configuration parameter 751
- ALIAS clause
  - COMMENT statement 83
  - DROP statement 245
  - MODULE clause
    - COMMENT statement 83
- aliases
  - adding comments to catalog 83
  - alias name 565
  - deleting using DROP statement 245
  - description 565
  - TABLE\_NAME function 486
  - TABLE\_SCHEMA function 487
- ALL clause
  - SELECT statement 419
- ALL option 382
- ALL PRIVILEGES clause
  - GRANT statement (Table, View or Nickname) 307
  - REVOKE table, view or nickname privileges 349
- alt\_collate configuration parameter 752
- ALTER AUDIT POLICY statement 3
- ALTER clause
  - GRANT statement (Table, View or Nickname) 307
  - REVOKE statement, removing privilege 349
- ALTER DATABASE PARTITION GROUP statement 6
- ALTER FUNCTION statement 9
- ALTER METHOD statement 11
- ALTER NODEGROUP statement
  - see ALTER DATABASE PARTITION GROUP 6
- ALTER PROCEDURE (External) statement 12
- ALTER SECURITY LABEL COMPONENT statement 14
- ALTER SECURITY POLICY statement 17
- ALTER TABLE statement
  - authorization required 20
  - examples 20
  - syntax diagram 20
- ALTER TABLESPACE statement
  - description 66
- ALTER VIEW statement
  - authorization 78
  - description 78
  - syntax diagram 78
- alternate\_auth\_enc 753
- alternate\_auth\_enc configuration parameter 753
- ambiguous reference errors 565
- APIs
  - for setting contexts between threads
    - sqlAttachToCtx() 517
    - sqlBeginCtx() 517
    - sqlDetachFromCtx() 517
    - sqlEndCtx() 517
    - sqlGetCurrentCtx() 517
    - sqlInterruptCtx() 517
    - sqlSetTypeCtx() 517
  - plug-in 620, 627
    - security plug-in 619, 621, 623, 626, 627, 633, 634, 635, 636, 638, 640, 642, 643, 644, 646, 647, 649
- appl\_memory database configuration parameter 753
- application development
  - routines 541
- application performance
  - comparison of sequences and identity columns 511
- application processes
  - definition 715
- application programs
  - controlling sequences 510
- application support layer heap size configuration parameter 755
- applications
  - maximum number of coordinating agents at node 814
- archretrydelay configuration parameter 755
- AS clause
  - CREATE VIEW statement 225
  - in SELECT clause 419
  - ORDER BY clause 419
- ASC clause
  - CREATE INDEX statement 101
  - SELECT statement 419
- aslheapsz configuration parameter 755
- assignments
  - basic SQL operations 723
- asterisk (\*)
  - in select column names 419
  - in subselect column names 419
- asterisks
  - select column names 419
  - subselect column names 419
- ASUTIME
  - in CREATE PROCEDURE statement 124
- asynchronous
  - events 517
- attributes
  - attribute name 565
- AUDIT statement 80
- audit\_buf\_sz configuration parameter 677
- authentication
  - GSS-API 593
  - ID/password 593
  - Kerberos 593
  - plug-ins
    - API for checking if authentication ID exists 649
    - API for cleaning client authentication resources 634
    - API for cleaning up resources 643
    - API for getting authentication IDs 647
    - API for initializing a client authentication plug-in 633
    - API for initializing server authentication 644
    - API for validating passwords 640
    - clean up server authentication 646
    - deploying 602, 604
    - for initializing a client authentication plug-in 633
    - library locations 597
    - user ID/ password 627
  - security plug-in 593
  - trust all clients configuration parameter 685
  - trusted clients authentication configuration parameter 858
  - two-part user IDs 598
- authentication configuration parameter 678
- authentication DAS configuration parameter 679
- authorities
  - defining group names
    - system administration authority group name configuration parameter 682
    - system control authority group name configuration parameter 683
    - system maintenance authority group name configuration parameter 683
- authorization ID 565
- authorization names
  - definition 565
  - description 565
  - restrictions governing 565
- authorizations
  - granting control on database operations 278
  - granting control on index 285
  - granting create on schema 295
  - public control on index 285

- authorizations (*continued*)
  - public create on schema 295
  - revoking 324
- auto restart enable configuration
  - parameter 703
- auto\_del\_rec\_obj database configuration
  - parameter 757
- auto\_maint configuration parameter 757
- automatic restart 713
- avg\_appls configuration parameter 759

## B

- backup
  - track modified pages 858
- backup\_pending configuration
  - parameter 760
- BIGINT data type
  - CREATE TABLE statement 143
- binary large objects (BLOBs) 143
- Bind API
  - creating packages 509
- bind behavior
  - DYNAMICRULES 513
- BIND command
  - creating packages 509
- bind options
  - overview 509
- BINDADD parameter
  - grant privilege 278
- binding
  - changing configuration
    - parameters 659, 660
    - GRANT statement 286
    - overview 509
    - revoking all privileges 331
- blk\_log\_dsk\_ful configuration parameter
  - details 760
- BLOBs (binary large objects)
  - CREATE TABLE statement 143
- books
  - printed
    - ordering 868
- buffer insert 313
- buffer pools
  - deleting using DROP statement 245
  - names 565
- BUFFERPOOL clause
  - ALTER TABLESPACE statement 66
  - CREATE TABLESPACE
    - statement 212
  - DROP statement 245
- bypass federated authentication
  - configuration parameter 789

## C

- C/C++ applications
  - multiple thread database access 517
- C/C++ language
  - connecting to databases 501
  - disconnecting from databases 502
- CASCADE delete rule 143
- catalog cache size configuration
  - parameter 761

- catalog\_noauth configuration
  - parameter 680
- catalogcache\_sz database configuration
  - parameter 761
- catalogs
  - COMMENT statement 83
- CCSID (coded character set identifier)
  - CREATE TABLE statement 143
- change the database log path
  - configuration parameter 822
- CHAR VARYING data type 143
- character conversion
  - rules for assignments 723
  - rules for comparison 723
- CHARACTER data type 143
- character strings
  - assignment 723
  - comparisons 723
  - equality
    - collating sequence examples 723
    - definition 723
- CHARACTER VARYING data type 143
- CHECK clause
  - CREATE VIEW statement 225
- check constraints
  - ALTER TABLE statement 20
  - CREATE TABLE statement 143
  - INSERT statement 313
- chngpgs\_thresh configuration
  - parameter 762
- client I/O block size configuration
  - parameter 842
- client support
  - client I/O block size configuration
    - parameter 842
  - TCP/IP service name configuration
    - parameter 681
- clnt\_krb\_plugin configuration
  - parameter 653
- clnt\_pw\_plugin configuration
  - parameter 653
- CLOBs (character large objects)
  - data type
    - creating columns 143
- CLOSE in CREATE INDEX
  - statement 101
- closing connection
  - importance of 521, 526
- CLUSTER clause
  - CREATE INDEX statement 101
- cluster managers
  - cluster manager name configuration
    - parameter 763
- cluster\_mgr configuration
  - parameter 763
- COBOL language
  - Connecting to databases 501
  - Disconnecting from databases 502
- code pages
  - database configuration
    - parameter 763
- codepage database configuration
  - parameter 763
- codeset database configuration
  - parameter 763
- collate\_info database configuration
  - parameter 764

- collating sequences
  - string comparison rules 723
- COLLID
  - in CREATE PROCEDURE
    - statement 124
- COLUMN clause
  - COMMENT statement 83
- column options
  - CREATE TABLE statement 143
- columns
  - adding comments to catalog 83
  - adding with ALTER TABLE
    - statement 20
  - ambiguous name reference
    - errors 565
  - column name
    - definition 565
    - qualification in COMMENT ON
      - statement 565
    - uses 565
  - constraint name
    - FOREIGN KEY rules 143
  - creating index keys 101
  - grant add privileges 307
  - GROUP BY
    - use in limiting in SELECT
      - clause 419
  - grouping column names in GROUP
    - BY 419
  - HAVING
    - use in limiting in SELECT
      - clause 419
  - HAVING clause
    - search names rules 419
  - inserting values 313
  - names
    - in ORDER BY clause 419
    - INSERT statement 313
    - qualified conditions 565
    - unqualified conditions 565
  - naming conventions 565
  - nested table expression 565
  - null values
    - in ALTER TABLE statement,
      - prevention 20
    - in result columns 419
  - qualified column name rules 565
  - result data 419
  - scalar fullselect 565
  - searching using WHERE clause 419
  - SELECT clause syntax diagram 419
  - string assignment rules 723
  - subquery 565
  - undefined name reference errors 565
  - updating 354
- combining grouping sets 419
- comm\_bandwidth database manager
  - configuration parameter
    - description 765
- COMMENT statement 83
- comments
  - in catalog table 83
  - SQL static statements 491
- commit
  - number of commits to group
    - (mincommit) 819
  - release of locks 715

commit (*continued*)  
     transaction, JDBC 525  
 COMMIT ON RETURN  
     in CREATE PROCEDURE  
         statement 124  
 COMMIT statement  
     description 365  
 Common Criteria  
     supported interfaces xi  
 Common Criteria certification ix  
 common table expressions  
     definition 390  
     recursive 390  
     select statement 390  
 communications  
     connection elapse time 697  
 comparing 723  
 comparison  
     SQL operation 723  
 compatibility  
     data types 723  
     rules 723  
     rules for operation types 723  
 compiling  
     overview 508  
 component-name  
     description 565  
 composite column values 419  
 concurrency control  
     LOCK TABLE statement 386  
     maximum number of active  
         applications 816  
 condition name  
     SQL procedures 565  
 configuration  
     changing database parameters 660  
 configuration file release level  
     configuration parameter 838  
 configuration files  
     description 659  
     location 659  
 configuration parameters 692, 693, 694,  
     695, 753, 767  
     agent\_stack\_sz 749  
     agentpri 751  
     alt\_collate 752  
     appl\_memory 753  
     applheapsz 754  
     archretrydelay 755  
     aslheapsz 755  
     audit\_buf\_sz 677  
     authentication 678  
     authentication (DAS) 679  
     auto\_del\_rec\_obj 757  
     auto\_maint 757  
     autorestart 703  
     avg\_appls 759  
     backup\_pending 760  
     blk\_log\_dsk\_ful 760  
     catalog\_noauth 680  
     catalogcache\_sz 761  
     chnpggs\_thresh 762  
     clnt\_krb\_plugin 653  
     clnt\_pw\_plugin 653  
     cluster\_mgr 763  
     codepage 763  
     codeset 763  
 configuration parameters (*continued*)  
     collate\_info 764  
     comm\_bandwidth 765  
     conn\_elapse 697  
     contact\_host 765  
     cpuspeed 766  
     das\_codepage 767  
     das\_territory 768  
     dasadm\_group 680  
     database  
         changing 659  
     database\_consistent 705  
     database\_level 768  
     database\_memory 768  
     db\_mem\_thresh 771  
     db2system 770  
     dbheap 772  
     decflt\_rounding 773  
     description 659  
     dft\_account\_str 774  
     dft\_degree 775  
     dft\_extent\_sz 776  
     dft\_loadrec\_ses 776  
     dft\_monswitches 777  
     dft\_mttb\_types 778  
     dft\_prefetch\_sz 778  
     dft\_queryopt 779  
     dft\_refresh\_age 780  
     dft\_sqlmathwarn 780  
     dftdbpath 681  
     diaglevel 782  
     diagpath 782  
     dir\_cache 783  
     discover 785  
     discover (DAS) 785  
     discover\_db 786  
     discover\_inst 786  
     dlchktime 686  
     dyn\_query\_mgmt 787  
     enable\_xmlchar 787  
     exec\_exp\_task 788  
     failarchpath 788  
     fcm\_num\_buffers 697  
     fcm\_num\_channels 698  
     fed\_noauth 789  
     federated 789  
     federated\_async 789  
     fenced\_pool 790  
     group\_plugin 654  
     hadr\_db\_role 791  
     hadr\_local\_host 792  
     hadr\_local\_svc 792  
     hadr\_peer\_window 793  
     hadr\_remote\_host 793  
     hadr\_remote\_inst 794  
     hadr\_remote\_svc 794  
     hadr\_synemode 794  
     hadr\_timeout 795  
     health\_mon 795  
     indexrec 796  
     instance\_memory 798  
     intra\_parallel 800  
     java\_heap\_sz 801  
     jdk\_64\_path 802  
     jdk\_path 803  
     jdk\_path (DAS) 802  
     keepfenced 803  
 configuration parameters (*continued*)  
     local\_gssplugin 654  
     locklist 686  
     locktimeout 689  
     log\_retain\_status 804  
     logarchmeth1 804  
     logarchmeth2 806  
     logarchopt1 807  
     logarchopt2 807  
     logbufsz 808  
     logfilsiz 808  
     loghead 809  
     logindexbuild 810  
     logpath 810  
     logprimary 810  
     logretain 811  
     logsecond 812  
     max\_connections 814  
         restrictions 839  
     max\_connretries 699  
     max\_coordagents 814  
         restrictions 839  
     max\_querydegree 815  
     max\_time\_diff 699  
     maxappls 816  
     maxfilop 817  
     maxlocks 690  
     maxlog 815  
     min\_dec\_div\_3 818  
     mincommit 819  
     mirrorlogpath 820  
     mon\_heap\_sz 821  
     multipage\_alloc 822  
     newlogpath 822  
     nodetype 707  
     notifylevel 824  
     num\_db\_backups 825  
     num\_freqvalues 825  
     num\_initagents 827  
     num\_initfenced 827  
     num\_iocleaners 827  
     num\_ioservers 829  
     num\_poolagents 830  
     num\_quantiles 831  
     numarchretry 832  
     numdb 833  
     numlogspan 830  
     numsegs 834  
     overflowlogpath 834  
     pagesize 835  
     pckcachesz 835  
     query\_heap\_sz 837  
     rec\_his\_retentn 838  
     release 838  
     restore\_pending 839  
     restrict\_access 709  
     resync\_interval 841  
     rollfwd\_pending 841  
     rqrioblk 842  
     sched\_enable 843  
     sched\_userid 843  
     self\_tuning\_mem 843  
     seqdetect 845  
     sheapthres 846  
     sheapthres\_shr 847  
     smtp\_server 848  
     softmax 849

configuration parameters (*continued*)

- sortheap 850
- spm\_log\_file\_sz 851
- spm\_log\_path 852
- spm\_max\_resync 853
- spm\_name 853
- srv\_plugin\_mode 656
- srvcon\_auth 654
- srvcon\_gssplugin\_list 655
- srvcon\_pw\_plugin 656
- start\_stop\_time 700
- stat\_heap\_sz 853
- stmheap 854
- summary
  - database 663
  - database manager 663
  - section heading descriptions 663
- svcname 681
- sysadm\_group 682
- sysctrl\_group 683
- sysmaint\_group 683
- sysmon\_group 684
- territory 855
- tm\_database 855
- toolscat\_db 855
- toolscat\_inst 856
- toolscat\_schema 856
- tp\_mon\_name 856
- trackmod 858
- trust\_allclnts 685
- trust\_clntauth 858
- tsm\_mgmtclass 859
- tsm\_nodename 859
- tsm\_owner 860
- tsm\_password 860
- user\_exit\_status 861
- userexit 861
- util\_heap\_sz 861
- util\_impact\_lim 862
- vendoropt 863
- wlm\_collect\_int configuration
  - parameter 863

conn\_elapse configuration

- parameter 697

CONNECT parameter

- GRANT...ON DATABASE statement 278

CONNECT statement

- application server information 366
- disconnecting from current server 366
- implicit connection 366
- new password information 366
- Type 2 373
- with no operand, returning information 366

CONNECT TO statement

- successful connection 366, 373
- unsuccessful connection 366, 373

connecting to a data source

- DataSource interface 523
- SQLJ 520

connection context

- class 520
- closing 521
- default 520
- object 520

connection declaration clause

- SQLJ 521

connection elapse time configuration

- parameter 697

connections

- elapse time 697
- consistency
  - points of 715

CONSTRAINT clause 83

constraints

- adding comments to catalog 83
- adding with ALTER TABLE 20
- dropping
  - with ALTER TABLE 20
- names, definition 565

contact\_host configuration

- parameter 765

container clause

- CREATE TABLESPACE statement 212

containers

- CREATE TABLESPACE statement 212

context clause

- SQLJ 520

contexts

- setting in multithreaded DB2 applications
  - described 517
  - SQLJ routines 546

CONTROL clause

- GRANT statement (Table, View or Nickname) 307
- revoking 349

CONTROL parameter

- revoking privileges for packages 331

conversions

- datetime to string variable 723
- numeric, scale and precision, summary 723
- rules
  - assignments 723
  - comparisons 723

Coordinated Universal Time 699

COPY

- CREATE INDEX statement 101

correlated reference

- in nested table expression 565
- in scalar fullselect 565
- in subquery 565
- in subselect 419

correlation name

- definition 565
- FROM clause
  - subselect rules 419
- in SELECT clause
  - syntax diagram 419
- qualified reference 565
- rules 565

cpuspeed configuration parameter

- described 766

crash recovery 713

CREATE AUDIT POLICY statement 95

CREATE DATABASE PARTITION GROUP statement 98

CREATE FUNCTION statement

- description 100

CREATE INDEX statement

- column-names in index keys 101
- description 101
- XML column 101

CREATE METHOD statement

- description 119

CREATE NODEGROUP statement

- CREATE DATABASE PARTITION GROUP statement 98

CREATE PROCEDURE (SQL) statement 124

CREATE ROLE statement 134

CREATE SCHEMA statement 135

CREATE SECURITY LABEL

- COMPONENT statement 139

CREATE SECURITY LABEL statement 137

CREATE SECURITY POLICY statement 141

CREATE TABLE statement

- syntax diagram 143

CREATE TABLESPACE statement

- description 212

CREATE VIEW statement

- description 225

CREATETAB parameter

- GRANT...ON DATABASE statement 278

creating

- databases, granting authority 278

cross-tabulation rows 419

CUBE grouping

- examples 419
- query description 419

cur\_commit 767

cur\_commit configuration

- parameter 767

CURRENT CLIENT\_ACCTNG special register 739

CURRENT DATE special register 739

CURRENT DECFLOAT ROUNDING MODE special register 740

CURRENT DEFAULT TRANSFORM GROUP special register 740

CURRENT DEGREE special register

- description 741

CURRENT EXPLAIN MODE special register

- description 741

CURRENT EXPLAIN SNAPSHOT special register

- description 742

CURRENT FEDERATED ASYNCHRONY special register 743

CURRENT FUNCTION PATH special register

- description 746

CURRENT IMPLICIT XMLPARSE OPTION special register 743

CURRENT ISOLATION special register 744

CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register 745

CURRENT MDC ROLLOUT MODE special register 745

- CURRENT OPTIMIZATION PROFILE
  - special register 745
- CURRENT PACKAGE PATH special register 745
- CURRENT PATH special register
  - description 746
- CURRENT QUERY OPTIMIZATION special register
  - description 747
- CURRENT REFRESH AGE special register
  - description 747
- CURRENT TIME special register 747
- CURRENT TIMESTAMP special register 748
- CURRENT TIMEZONE special register 748
- CURRENT USER special register 749
  - cursor name
    - definition 565
  - cursor variable name
    - definition 565
  - cursors
    - deleting, search condition details 239
    - terminating for unit of work 388
  - WITH HOLD
    - lock clause of COMMIT statement, effect 365

## D

- DAS discovery mode configuration
  - parameter 785
- das\_codepage configuration
  - parameter 767
- das\_territory configuration
  - parameter 768
- dasadm\_group configuration
  - parameter 680
- data
  - integrity
    - locks 386
- data source
  - connecting to using JDBC 522
  - connecting using DriverManager 538
  - connecting using JDBC
    - DataSource 523
- data sources
  - identifying 565
- data structures
  - packed decimal 555
- data types
  - result columns 419
- database authorities
  - granting
    - GRANT (database authorities) statement 278
- database configuration file
  - changing 677
- database configuration parameters
  - autorestart 713
- database heap configuration
  - parameter 772
- database manager
  - machine node type configuration
    - parameter 707
  - start timeout 700
- database manager (*continued*)
  - stop timeout 700
- database manager configuration
  - parameters
    - summary 663
- database partition groups
  - adding comments to catalog 83
  - adding partitions 6
  - creating 98
  - distribution map creation 98
  - dropping partitions 6
- database partitions
  - database configuration updates 677
- database system monitor
  - default database system monitor
    - switches configuration
      - parameter 777
- database territory code configuration
  - parameter 766
- database\_consistent configuration
  - parameter 705
- database\_level configuration
  - parameter 768
- database\_memory database configuration
  - parameter
    - description 768
- database-managed space (DMS)
  - table spaces
    - CREATE TABLESPACE statement 212
- databases
  - accessing
    - granting authority 278
    - multiple threads 517
  - appl\_memory configuration
    - parameter 753
  - autorestart configuration
    - parameter 703
  - backup\_pending configuration
    - parameter 760
  - codepage configuration
    - parameter 763
  - codeset configuration parameter 763
  - collating information 764
  - configuration parameter
    - summary 663
  - contexts 517
  - CREATE TABLESPACE statement 212
  - distributed 515
  - maximum number of concurrently
    - active databases 833
  - release level configuration
    - parameter 838
  - territory code configuration
    - parameter 766
  - territory configuration parameter 855
- db\_mem\_thresh configuration
  - parameter 771
- DB2 administration server (DAS)
  - configuration parameters
    - authentication 679
    - contact\_host 765
    - das\_codepage 767
    - das\_territory 768
    - dasadm\_group 680
    - db2system 770
- DB2 administration server (DAS) (*continued*)
  - configuration parameters (*continued*)
    - exec\_exp\_task 788
    - jdk\_64\_path 802
    - jdk\_path 802
    - sched\_enable 843
    - sched\_userid 843
    - smtp\_server 848
    - toolscat\_db 855
    - toolscat\_inst 856
    - toolscat\_schema 856
- DB2 Information Center
  - languages 869
  - updating 870, 871
  - versions 869
  - viewing in different languages 869
- DB2 JDBC Type 2 Driver
  - DriverManager interface 527
  - security 526
- DB2INSTPROF registry variable
  - location 659
- db2nodes.cfg file
  - ALTER DATABASE PARTITION GROUP statement 6
  - CONNECT (Type 1) statement 366
  - CREATE DATABASE PARTITION GROUP statement 98
  - DBPARTITIONNUM function 476
- DB2SECURITYLABEL data type
  - CREATE TABLE statement 143
- db2system configuration parameter 770
- DBADM (database administration)
  - authority
    - granting 278
- DBCLOB data type
  - CREATE TABLE statement 143
- dbheap database configuration parameter
  - overview 772
- DBPARTITIONNUM function
  - description 476
- deadlocks
  - checking for 686
  - dlchktme configuration
    - parameter 686
- debugging
  - security plug-ins 600
- decflt\_rounding database configuration
  - parameter 773
- decimal conversion 723
- DECIMAL data type
  - assignments
    - cursor type 723
  - conversion
    - floating-point 723
- decimal division scale to 3 configuration
  - parameter 818
- DECRYPT\_BIN function 477
- DECRYPT\_CHAR function 477
- decrypting information 477
- default connection context 520
- default database path configuration
  - parameter 681
- default number of SMS containers
  - configuration parameter 834
- deletable views
  - description 225



- DELETE statement
  - description 239
- dependent objects
  - DROP statement 245
- descriptor-name
  - syntax diagram 565
- dft\_account\_str configuration
  - parameter 774
- dft\_degree configuration parameter
  - description 775
- dft\_extent\_sz configuration
  - parameter 776
- dft\_loadrec\_ses configuration
  - parameter 776
- dft\_mon\_bufpool configuration
  - parameter 777
- dft\_mon\_lock configuration
  - parameter 777
- dft\_mon\_sort configuration
  - parameter 777
- dft\_mon\_stmt configuration
  - parameter 777
- dft\_mon\_table configuration
  - parameter 777
- dft\_mon\_timestamp configuration
  - parameter 777
- dft\_mon\_uow configuration
  - parameter 777
- dft\_monswitches configuration
  - parameter 777
- dft\_mttb\_types configuration
  - parameter 778
- dft\_prefetch\_sz configuration
  - parameter 778
- dft\_queryopt configuration
  - parameter 779
- dft\_refresh\_age configuration
  - parameter 780
- dft\_sqlmathwarn configuration
  - parameter 780
- dftdbpath configuration parameter 681
- diaglevel configuration parameter
  - description 782
- diagpath configuration parameter 782
- dir\_cache configuration parameter 783
- directory cache support configuration
  - parameter
  - description 783
- DISCONNECT statement 379
- discover server instance configuration
  - parameter 786
- discover\_db configuration
  - parameter 786
- discover\_inst configuration
  - parameter 786
- discovery feature
  - discovery mode configuration
    - parameter 785
- discovery mode configuration
  - parameter 785
- DISTINCT keyword
  - subselect statement 419
- distinct types
  - comparisons
    - overview 723
  - DROP statement 245
  - names 565
- distributed relational databases
  - units of work 515
- dlchktme configuration parameter 686
- documentation
  - overview 865
  - PDF 865
  - printed 865
  - terms and conditions of use 874
- double-byte character set (DBCS)
  - characters truncated during
    - assignment 723
- DriverManager interface
  - DB2 JDBC Type 2 Driver 527
- DROP statement
  - description 245
  - transforms 245
- dyn\_query\_mgmt configuration
  - parameter
    - description 787
- dynamic SQL
  - cursors
    - DECLARE CURSOR
      - statement 491
    - DYNAMICRULES effects 513
    - EXECUTE statement
      - invoking SQL statements 491
    - FETCH statement
      - invoking SQL statements 491
    - invoking statements 491
    - OPEN statement 491
    - PREPARE statement
      - invoking SQL statements 491
    - SQLDA
      - description 555
  - DYNAMICRULES precompile/bind
    - option
      - effects on dynamic SQL 513
- E**
- embedded SQL applications
  - authorization 512
  - overview 491
- enable\_xmlchar database configuration
  - parameter
    - description 787
- ENCRYPT scalar function 479
- encrypted security-sensitive data
  - IBM Data Server Driver for JDBC and
    - SQLJ 534
- encrypted user ID or encrypted password
  - security
    - IBM Data Server Driver for JDBC and
      - SQLJ 534
- encryption
  - ENCRYPT function 479
  - GETHINT function 481
- error messages
  - return codes 491
  - security plug-ins 618
  - SQLCA definitions 549
  - UPDATE statement 354
- event monitors
  - DROP statement 245
  - names 565
- EXCEPT operator of fullselect 382
- exception tables
  - SET INTEGRITY statement 400
- EXCLUSIVE MODE connection 366
- exec\_exp\_task configuration
  - parameter 788
- executable SQL statements 491
- EXECUTE IMMEDIATE statement
  - embedded 491
- EXECUTE statement
  - embedded 491
- exposed correlation names
  - FROM clause 565
- expressions
  - GROUP BY clause 419
  - ORDER BY clause 419
  - SELECT clause 419
  - subselect 419
- F**
- failarchpath configuration
  - parameter 788
- failed database partition server 717
- failure transaction 713
- FCM (Fast Communications Manager)
  - channels 698
  - monitor elements
    - fcm\_num\_channels 698
- fcm\_num\_buffers configuration
  - parameter 697
- fcm\_num\_channel configuration
  - parameter 698
- fed\_noauth configuration parameter 789
- federated configuration parameter 789
- federated databases
  - system support configuration
    - parameter 789
- federated\_async database manager
  - configuration parameter 789
- fenced\_pool database manager
  - configuration parameter 790
- FIELDPROC clause
  - in ALTER TABLE statement 20
- file reference variables
  - BLOBs 565
  - CLOBs 565
  - DBCLOBs 565
- first active log file configuration
  - parameter 809
- flagger utility for precompiling 506
- FLOAT data type 143
- floating point
  - to decimal conversion 723
- FOR BIT DATA clause
  - CREATE TABLE statement 143
- FOR clause
  - CREATE TABLE statement 143
- FOR FETCH ONLY clause
  - SELECT statement 390
- FOR READ ONLY clause
  - SELECT statement 390
- FOREIGN KEY clause 143
- foreign keys
  - adding with ALTER TABLE 20
  - constraint name conventions 143
  - dropping with ALTER TABLE 20

FORTRAN language  
 Connecting to databases 501  
 FREEPAGE in CREATE INDEX  
 statement 101  
 FROM clause  
 correlation-name example 565  
 DELETE statement 239  
 exposed names explained 565  
 non-exposed names explained 565  
 subselect syntax 419  
 use of correlation names 565  
 fullselect  
 CREATE VIEW statement 225  
 detailed syntax 382  
 examples 382  
 initialization 390  
 iterative 390  
 multiple operations, order of  
 execution 382  
 ORDER BY clause 419  
 subquery role, search condition 565  
 table reference 419  
 FUNCTION clause in COMMENT ON  
 statement 83  
 function designator syntax element 497  
 function mappings  
 mapping name 565  
 function name 565  
 functions  
 adding comments to catalog 83  
 arguments 475  
 client plug-in  
 check if authentication ID  
 exists 649  
 clean up client authentication 634  
 clean up resources 643  
 clean up server  
 authentication 646  
 free memory held by token 643  
 generate initial credentials 638  
 get authentication IDs 647  
 get default login context 636  
 initialize client authentication 633  
 initialize server  
 authentication 644  
 process service principal  
 name 642  
 remap user ID and password 635  
 validate password 640  
 description 475  
 group plug-in  
 check if group exists 626  
 clean up 623  
 free error message memory 627  
 free group list memory 627  
 get list of groups 623  
 initialization 621  
 in expressions 475  
 scalar  
 DBPARTITIONNUM 476  
 DECRYPTBIN 477  
 DECRYPTCHAR 477  
 ENCRYPT 479  
 GETHINT 481  
 HASHEDVALUE 482  
 NODENUMBER (see  
 DBPARTITIONNUM) 476

functions (*continued*)  
 scalar (*continued*)  
 PARTITION (see  
 HASHEDVALUE) 482  
 SECLABEL 483  
 SECLABEL\_BY\_NAME 483  
 SECLABEL\_TO\_CHAR 484  
 TABLE\_NAME 486  
 TABLE\_SCHEMA 487  
**G**  
 GBPCACHE  
 in CREATE INDEX statement 101  
 generated columns  
 CREATE TABLE statement 143  
 GETHINT function  
 description 481  
 values and arguments 481  
 grand total row 419  
 GRANT (Exemption) statement 283  
 GRANT (Package Privileges)  
 statement 286  
 GRANT (Role) statement 289  
 GRANT (routine privileges) statement  
 description 291  
 GRANT (Schema Privileges) statement  
 description 295  
 GRANT (Security Label) statement 297  
 GRANT (Sequence Privileges) statement  
 description 300  
 GRANT (Server Privileges) statement  
 description 302  
 GRANT (SETSESSIONUSER Privilege)  
 statement 303  
 GRANT (Table Space Privileges)  
 statement  
 description 305  
 GRANT statement  
 CONTROL ON INDEX  
 description 285  
 CREATE ON SCHEMA 295  
 database authority  
 description 278  
 Nickname Privileges 307  
 Package Privileges  
 description 286  
 Table Privileges 307  
 Table, View or Nickname Privileges  
 description 307  
 View Privileges 307  
 GRAPHIC data type  
 CREATE TABLE statement 143  
 GROUP BY clause  
 subselect results 419  
 subselect rules and syntax 419  
 group\_plugin configuration  
 parameter 654  
 grouping-expression 419  
 groups  
 defining names 565  
 GSS-APIs  
 authentication plug-ins 649  
 Restrictions 649

**H**  
 hadr\_db\_role configuration  
 parameter 791  
 hadr\_local\_host configuration  
 parameter 792  
 hadr\_local\_svc configuration  
 parameter 792  
 hadr\_peer\_window database  
 configuration parameter  
 description 793  
 hadr\_remote\_host configuration  
 parameter 793  
 hadr\_remote\_inst configuration  
 parameter 794  
 hadr\_remote\_svc configuration  
 parameter 794  
 hadr\_syncmode configuration  
 parameter 794  
 hadr\_timeout configuration  
 parameter 795  
 HASHEDVALUE function 482  
 hashing on partition keys 143  
 HAVING clause 419  
 health monitor  
 health\_mon configuration  
 parameter 795  
 health\_mon configuration parameter 795  
 help  
 configuring language 869  
 SQL statements 869  
 host identifiers  
 overview 565  
 host variables  
 BLOB 565  
 CLOB 565  
 DBCLOB 565  
 embedded SQL statements 491  
 indicator variables 565  
 inserting in rows 313  
 overview 565  
 syntax diagrams 565  
**I**  
 IBM Data Server Driver for JDBC and  
 SQLJ  
 connecting to a data source  
 DriverManager interface 538  
 encrypted user ID or encrypted  
 password security 534  
 Kerberos security 531  
 security 536  
 user ID and password security 529  
 user ID-only security 530  
 identifiers  
 delimited 565  
 host 565  
 ordinary 565  
 SQL 565  
 Identifiers  
 cursor-name 565  
 identity columns  
 comparison with sequences 511  
 CREATE TABLE statement 143  
 implicit connections  
 CONNECT statement 366



- implicit schemas
  - GRANT (Database Authorities) statement 278
  - REVOKE (Database Authorities) statement 324
- IN
  - CREATE TABLE statement 143
- IN EXCLUSIVE MODE clause
  - LOCK TABLE statement 386
- IN SHARE MODE clause
  - LOCK TABLE statement 386
- INCLUDE clause
  - CREATE INDEX statement 101
- INDEX clause
  - COMMENT statement 83
  - CREATE INDEX statement 101
  - GRANT statement (Table, View or Nickname) 307
  - REVOKE statement, removing privileges 349
- INDEX keyword
  - DROP statement 245
- index over XML data
  - CREATE INDEX statement 101
  - Syntax and option descriptions 101
- indexes
  - catalog specification comments, adding 83
  - correspondence to inserted row values 313
  - deleting
    - using the DROP statement 245
  - grant control 285, 307
  - name
    - definition 565
    - primary key constraint 143
    - unique constraint 143
  - primary key, use in matching 20
  - privileges
    - revoking 330
    - renaming 321
    - unique key, use in matching 20
- indexrec configuration parameter 796
- indicator variables
  - description 565
  - host variable, uses in declaring 565
- initial number of agents in pool
  - configuration parameter 827
- initial number of fenced processes
  - configuration parameter 827
- initializing
  - fullselect 390
- inoperative
  - views 225
- INSERT
  - inserting values 313
  - restrictions leading to failure 313
- INSERT clause
  - GRANT statement (Table, View or Nickname) 307
  - REVOKE statement, removing privileges 349
- INSERT statement
  - description 313
- insertable views 225

- instances
  - instance\_memory configuration parameter 798
- INTEGER data type 143
- integers
  - decimal conversion summary 723
  - in ORDER BY clause 419
- integrity constraints
  - adding comments to catalog 83
- intermediate result tables 419
- INTERSECT operator
  - duplicate rows, use of ALL 382
  - of fullselect, role in comparison 382
- INTO clause
  - FETCH statement, use in host variable 565
  - INSERT statement, naming table or view 313
  - restrictions on using 313
  - SELECT INTO statement, use in host variable 565
  - values from applications programs 565
- intra\_parallel database manager
  - configuration parameter 800
- invoke behavior
  - DYNAMICRULES 513
- IS clause
  - COMMENT statement 83
- isolation level
  - SQLJ 519
- isolation levels
  - in DELETE statement 239, 313, 354, 390
- iterative fullselect 390

**J**

- Java
  - applications 519
- java\_heap\_sz database manager
  - configuration parameter 801
- JDBC connection
  - using 525
- JDBC transaction
  - committing 525
  - rolling back 525
- jdk\_64\_path configuration
  - parameter 802
- jdk\_path configuration parameter
  - application development 547
  - description 803
- jdk\_path DAS configuration
  - parameter 802
- joins
  - CREATE TABLE statement 143
  - subselect component of fullselect 419
- tables
  - subselect clause 419
- types
  - full outer 419
  - inner 419
  - left outer 419
  - right outer 419

**K**

- keepfenced configuration parameter
  - description 803
  - updating 547
- Kerberos authentication protocol
  - samples 651
- Kerberos security
  - IBM Data Server Driver for JDBC and SQLJ 531

**L**

- label-based access control (LBAC)
  - ALTER SECURITY LABEL COMPONENT statement 14
  - ALTER SECURITY POLICY statement 17
  - CREATE SECURITY LABEL COMPONENT statement 139
  - CREATE SECURITY LABEL statement 137
  - CREATE SECURITY POLICY statement 141
  - GRANT (Exemption) statement 283
  - GRANT (Security Label) statement 297
  - REVOKE (Exemption) statement 328
  - REVOKE (Security Label) statement 341
- labels
  - object names in SQL procedures 565
- latches
  - status with multiple threads 517
- lateral correlation 419
- LBAC (label-based access control)
  - ALTER SECURITY LABEL COMPONENT statement 14
  - ALTER SECURITY POLICY statement 17
  - CREATE SECURITY LABEL COMPONENT statement 139
  - CREATE SECURITY LABEL statement 137
  - CREATE SECURITY POLICY statement 141
  - GRANT (Exemption) statement 283
  - GRANT (Security Label) statement 297
  - REVOKE (Exemption) statement 328
  - REVOKE (Security Label) statement 341
- rule exemptions
  - GRANT (Exemption) statement 283
  - REVOKE (Exemption) statement 328
- security label components
  - ALTER SECURITY LABEL COMPONENT statement 14
  - CREATE SECURITY LABEL COMPONENT statement 139
- security labels
  - ALTER SECURITY LABEL COMPONENT statement 14
  - CREATE SECURITY LABEL COMPONENT statement 139

LBAC (label-based access control)  
*(continued)*  
 security labels *(continued)*  
 CREATE SECURITY LABEL statement 137  
 GRANT (Security Label) statement 297  
 REVOKE (Security Label) statement 341  
 security policies  
 ALTER SECURITY POLICY statement 17  
 CREATE SECURITY POLICY statement 141

libraries  
 security plug-in  
 loading in DB2 609  
 restrictions 613  
 shared  
 rebuilding routine 547

linking  
 description 508

LOAD parameter  
 GRANT...ON DATABASE statement 278

loading  
 granting database authority 278

local\_gssplugin configuration parameter 654

locators  
 variable description 565

LOCK TABLE statement  
 description 386

locklist configuration parameter  
 description 686

locks  
 COMMIT statement 365  
 definition 715  
 during UPDATE 354  
 INSERT statement, default rules for 313  
 LOCK TABLE statement 386  
 maximum percent of lock list before escalation 690  
 maximum storage for lock list 686  
 restricting access 386  
 terminating for unit of work 388  
 time interval for checking deadlock configuration parameter 686

locktimeout configuration parameter 689

log\_retain\_status configuration parameter 804

logarchmeth1 configuration parameter 804

logarchmeth2 configuration parameter 806

logarchopt1 configuration parameter 807

logarchopt2 configuration parameter 807

LOGBUFSZ configuration parameter 808

logfilsiz configuration parameter 808

loghead configuration parameter 809

logindexbuild configuration parameter 810

logpath configuration parameter 810

logprimary configuration parameter 810

logretain database configuration parameter 811

logs  
 block on log disk full configuration parameter 760  
 creating table without initial logging 143  
 first active log file configuration parameter 809  
 location of log files configuration parameter 810  
 log buffer size configuration parameter 808  
 log retain enable configuration parameter 811  
 log retain status indicator configuration parameter 804  
 mirror log path configuration parameter 820  
 newlogpath configuration parameter 822  
 number of primary log files configuration parameter 810  
 number of secondary log files configuration parameter 812  
 overflow log path configuration parameter 834  
 recovery range and soft checkpoint interval configuration parameter 849  
 size of log files configuration parameter 808  
 user exit enable configuration parameter 861

logsecond configuration parameter 812

## M

MANAGED BY clause  
 CREATE TABLESPACE statement 212

materialized query tables (MQTs)  
 definition 143

max\_connections database manager configuration parameter  
 restrictions 839

max\_connretries configuration parameter 699

max\_coordagents database manager configuration parameter 814  
 restrictions 839

max\_logicagents configuration parameter 814

max\_querydegree configuration parameter 815

max\_time\_diff configuration parameter 699

maxappls configuration parameter 816

MAXDARI configuration parameter  
 renamed to fenced\_pool configuration parameter 790

maxfilop database configuration parameter 817

maximum database files open per application configuration parameter 817

maximum Java interpreter heap size configuration parameter 801

maximum log per transaction configuration parameter 815

maximum number of active applications configuration parameter 816

maximum number of concurrently active databases configuration parameter 833

maximum number of coordinating agents configuration parameter 814

maximum number of fenced processes configuration parameter 790

maximum percent of lock list before escalation configuration parameter 690

maximum query degree of parallelism configuration parameter 815

maximum storage for lock list configuration parameter 686

maximum time difference among nodes configuration parameter 699

maxlocks configuration parameter 690

maxlog configuration parameter 815

memory  
 applheapsz configuration parameter 754  
 application memory configuration parameter 753  
 aslheapsz configuration parameter 755  
 dbheap configuration parameter 772  
 instance memory configuration parameter 798  
 package cache size configuration parameter 835  
 sort heap size configuration parameter 850  
 sort heap threshold configuration parameter 846  
 statement heap size configuration parameter 854

METHOD clause  
 DROP statement 245

method designator syntax element 497

method name 565

methods  
 common uses 543  
 features 543  
 limitations 543  
 supported languages 543

min\_dec\_div\_3 configuration parameter 818

mincommit configuration parameter 819

mirror log path configuration parameter 820

mirrorlogpath configuration parameter 820

MODE keyword  
 LOCK TABLE statement 386

MODULE keyword  
 DROP statement 245

mon\_heap\_sz database manager configuration parameter 821

MQTs (materialized query tables)  
 definition 143

multi-threaded applications  
 SQLJ routines 546

multipage\_alloc configuration  
parameter 822

## N

names

deleting rows 239  
subselect columns 419

naming conventions  
identifiers 565

qualified column rules 565

nested table expressions 419

newlogpath configuration  
parameter 822

NICKNAME clause in DROP  
statements 245

nicknames

definition 565

FROM clause

exposed names 565  
nonexposed names 565  
subselect 419

privileges

grant 307  
grant control 307  
revoking 349

qualifying a column name 565  
SELECT clause 419

NO ACTION delete rule 143

node connection retries configuration  
parameter 699

NODENUMBER function

DBPARTITIONNUM 476

nodes

connection elapse time 697  
coordinating agents 814  
maximum time difference among 699

nodetype configuration parameter 707

nonexecutable SQL statements

invoking 491  
precompiler requirements 491

nonexposed correlation-name in FROM  
clause 565

NOT FENCED routines 544

NOT NULL clause

in the CREATE TABLE statement 143

notices 875

notify level configuration parameter  
overview 824

NULL value

SQL

assignment 723  
grouping-expressions, allowable  
uses 419  
occurrences in duplicate rows 419  
result columns 419  
specified by indicator  
variable 565

num\_db\_backups configuration  
parameter

overview 825

num\_freqvalues configuration  
parameter 825

num\_initfenced database manager  
configuration parameter 827

num\_iocleaners configuration  
parameter 827

num\_ioservers configuration  
parameter 829

num\_poolagents database manager  
configuration parameter 830

num\_quantiles configuration  
parameter 831

numarchretry configuration  
parameter 832

number log span configuration  
parameter 830

number of commits to group  
configuration parameter 819

number of database backups  
configuration parameter 825

numbers

precision 555  
scale 555

NUMDB

configuration parameter 833

numeric

assignments in SQL operations 723  
comparisons 723

numinitagents configuration

parameter 827

numlogspan configuration  
parameter 830

numsegs database configuration  
parameter

overview 834

## O

object identifier (OID)

columns

overview 143  
CREATE TABLE statement 143  
CREATE VIEW statement 225

object table 565

OF clause

CREATE VIEW statement 225

OID

see object identifier (OID) 143

ON clause

CREATE INDEX statement 101

ON TABLE clause

GRANT statement 307  
REVOKE statement 349

ON UPDATE clause 143

on-db-partitions-clause

CREATE TABLESPACE  
statement 212

ONLY clause

DELETE statement 239  
UPDATE statement 354

operations

assignments 723  
comparisons 723

OPTION clause

CREATE VIEW statement 225

ORDER BY clause

select statement 419

ordering DB2 books 868

outer joins

joined table 419

overflowlogpath configuration  
parameter 834

## P

packages

ALTER TABLE STATEMENT 20

authority to create

granting 278

authorization IDs

binding 565  
dynamic statements 565

catalog comments 83

COMMIT statement effect on  
cursors 365

creating

BIND command and existing bind  
file 509

deleting 245

names

overview 565

privileges

granting 286  
revoking using REVOKE (package  
privileges) statement 331  
revoking using REVOKE (table,  
view, or nickname privileges)  
statement 349

pages

sizes

database default 835

pagesize configuration parameter 835

parallelism

configuration parameters

dft\_degree 775  
intra\_parallel 800  
max\_querydegree 815

parameter markers

dynamic SQL

host variables 565

parameters

naming conventions 565

PARTITION function

substitution for HASHEDVALUE  
name 482

partitioned database environment

communications 697

partitioned database environments

transactions  
failure recovery 717

partitioning keys

adding 20  
defining when creating tables 143  
dropping 20

partitioning maps

creating for database partition  
groups 98

pckcachesz configuration parameter 835

performance

managing sequences 510  
partitioning key  
recommendation 143  
routines  
benefits 541

PIECESIZE

CREATE INDEX statement 101

plug-ins

group retrieval 620  
GSS-API authentication 649  
ID authentication 627  
password authentication 627

- plug-ins (*continued*)
  - security 593
    - APIs 610, 619
    - deploying 602, 603, 604
    - error messages 618
    - library restrictions 613
    - naming conventions 597
    - restrictions (GSS-API authentication) 650
    - restrictions (summary) 606
    - return codes 615
    - samples 651
    - versions 651
- point of consistency
  - database 713, 715
- pool size for agents
  - controlling 830
- positional updating of columns by row 354
- precision
  - numbers
    - SQLLEN variable 555
- precompiling
  - accessing host application servers through DB2 Connect 506
  - accessing multiple servers 506
  - embedded SQL applications 506
  - flagger utility 506
  - non-executable SQL statements 491
- PREPARE statement
  - embedded 491
- primary keys
  - adding
    - ALTER TABLE statement 20
    - CREATE TABLE statement 143
  - dropping 20
  - privileges required 307
- privileges
  - database
    - revoking 339
  - INDEX
    - revoking 330
  - packages
    - revoking 331, 349
  - revoking
    - REVOKE statement 349
- problem determination
  - information available 874
  - security plug-ins 600
  - tutorials 874
- procedure designator syntax
  - element 497
- procedures
  - authorization for creating
    - CREATE PROCEDURE (SQL) statement 124
  - creating 124
  - names
    - overview 565
- PROGRAM
  - DROP statement 245
- protocols
  - TCP/IP service name configuration
    - parameter 681
- PUBLIC AT ALL LOCATIONS 307

## Q

- qualified column names 565
- Qualified names 565
- qualifiers
  - object name 565
- queries
  - authorization IDs 418
  - definition 418
  - examples
    - SELECT statement 390
  - recursive 390
  - statement heap size configuration
    - parameter 854
- query\_heap\_sz database manager
  - configuration parameter 837

## R

- read-only views 225
- REAL SQL data type
  - CREATE TABLE statement 143
- rec\_his\_reten configuration
  - parameter 838
- records
  - locks to row data 313
- recovery
  - auto restart enable configuration
    - parameter 703
  - backup pending indicator
    - configuration parameter 760
  - crash 713
  - default number of load recovery sessions configuration
    - parameter 776
  - index re-creation time configuration
    - parameter 796
  - log retain status indicator
    - configuration parameter 804
  - number of database backups
    - configuration parameter 825
  - restore pending configuration
    - parameter 839
  - roll forward pending indicator
    - configuration parameter 841
  - two-phase commit protocol 717
  - user exit status indicator configuration
    - parameter 861
- recovery history file
  - retention period configuration
    - parameter 838
- recovery range and soft checkpoint
  - interval configuration parameter 849
- recursion queries 390
- recursive common table expression 390
- reference types
  - comparisons 723
- REFERENCES clause
  - GRANT statement (Table, View or Nickname) 307
  - REVOKE statement, removing privileges 349
- referential constraints
  - adding comments to catalog 83
- release configuration parameter 838
- releasing resources
  - closing connection 521, 526

- remote
  - function name 565
  - type name 565
- remote access
  - CONNECT statement
    - EXCLUSIVE MODE, dedicated connection 366
    - ON SINGLE DBPARTITIONNUM, dedicated connection 366
    - server information only, no operand 366
    - SHARE MODE, read-only for non-connector 366
    - successful connections 366
    - unsuccessful connections 366
- remote authorization name 565
- remote unit of work
  - characteristics 516
  - example 516
  - overview 516
- remote-object-name 565
- remote-schema-name 565
- remote-table-name 565
- RENAME statement 321
- RENAME TABLESPACE statement 323
- Resolution of identifiers 565
- RESTART DATABASE command 713
- restore\_pending configuration
  - parameter 839
- RESTRICT delete rule 143
- restrict\_access configuration
  - parameter 709
- RESTRICTIVE option
  - CREATE DATABASE
    - database configuration
      - parameter 709
- result columns
  - subselect 419
- result tables
  - query 418
- resync\_interval configuration
  - parameter 841
- return codes
  - embedded statements 491
  - executable SQL statements 491
- REVOKE (Exemption) statement 328
- REVOKE (Package Privileges) statement 331
- REVOKE (Role) statement 334
- REVOKE (Security Label) statement 341
- REVOKE (Sequence Privileges) statement
  - description 342
- REVOKE (SETSESSIONUSER Privilege) statement 346
- REVOKE statement
  - database authorities 324
  - index privileges 330
  - nickname privileges 349
  - package privileges 331
  - routine privileges 336
  - schema privileges 339
  - server privileges 344
  - table privileges 349
  - table space privileges 347
  - view privileges 349
- REXX language
  - Connecting to databases 501



- REXX language (*continued*)
  - Disconnecting from databases 502
- roll back
  - transaction, JDBC 525
- rollback
  - definition 715
- ROLLBACK statement
  - description 388
  - syntax 388
- ROLLBACK TO SAVEPOINT statement
  - description 388
- rollforward utility
  - roll forward pending indicator 841
- rollfwd\_pending configuration
  - parameter 841
- ROLLUP grouping of GROUP BY
  - clause 419
- routines
  - altering 547
  - benefits 541
  - classes 547
  - creating
    - security 503
  - description 541
  - invoking
    - security 503
  - libraries 547
  - NOT FENCED
    - security 503, 504, 544
  - rebuilding shared libraries 547
  - scalar UDFs
    - overview 542
    - security 503, 504, 544
  - THREADSAFE
    - security 544
- row fullselect
  - UPDATE statement 354
- rows
  - deleting 239
  - grant privilege 307
  - GROUP BY clause 419
  - HAVING clause 419
  - index keys with UNIQUE clause 101
  - indexes 101
  - inserting 313
  - locks to row data, INSERT
    - statement 313
  - restrictions leading to failure 313
  - SELECT clause
    - syntax diagram 419
  - updating column values, UPDATE
    - statement 354
- rqioblk configuration parameter 842
- run behavior
  - DYNAMICRULES 513
- run-time authorization ID 565
- run-time services
  - multiple threads
    - effect on latches 517

## S

- sampling
  - subselect tablesample-clauses 419
- savepoints
  - names 565
  - ROLLBACK TO SAVEPOINT 388

- scalar functions
  - description 542
- scale
  - data
    - comparisons in SQL 723
    - determined by SQLEEN
      - variable 555
    - number conversion in SQL 723
  - numbers
    - determined by SQLEEN
      - variable 555
- sched\_enable configuration
  - parameter 843
- sched\_userid configuration
  - parameter 843
- SCHEMA clause
  - COMMENT statement 83
  - DROP statement 245
- schemas
  - adding comments to catalog 83
  - CREATE SCHEMA statement 135
  - implicit
    - granting authority 278
    - revoking authority 324
  - names 565
- scope
  - adding with ALTER TABLE
    - statement 20
  - adding with ALTER VIEW
    - statement 78
  - CREATE VIEW statement 225
  - defining with added column 20
  - defining with CREATE TABLE
    - statement 143
- SCOPE clause
  - ALTER TABLE statement 20
  - ALTER VIEW statement 78
  - CREATE TABLE statement 143
  - CREATE VIEW statement 225
- search conditions
  - HAVING clause
    - arguments and rules 419
  - WHERE clause 419
  - with DELETE
    - row selection 239
  - with UPDATE
    - arguments and rules 354
- SECADM
  - database authority
    - GRANT (Database Authorities)
      - statement 278
    - revoking 324
  - parameter
    - GRANT (Database Authorities)
      - statement 278
    - REVOKE (Database Authorities)
      - statement 324
- SECLABEL
  - scalar function 483
- SECLABEL\_BY\_NAME scalar function
  - description 483
- SECLABEL\_TO\_CHAR scalar function
  - description 484
- SECURED WITH clause
  - ALTER TABLE statement 20
  - CREATE TABLE statement 143

- security
  - applications 519
  - CONNECT statement 366
  - DB2 JDBC Type 2 Driver 526
  - IBM Data Server Driver for JDBC and SQLJ 536
  - plug-ins 593
    - 32 bit considerations 600
    - 64 bit considerations 600
    - API for validating passwords 640
    - API versions 651
    - APIs 619, 621, 623, 626, 627, 633, 634, 635, 636, 638, 642, 643, 644, 646, 647, 649
    - APIs for group retrieval 620
    - APIs for GSS-API 649
    - APIs for user ID/password 627
    - calling sequence of, order in which called 610
    - configuration parameters 653, 654, 656, 678
    - debugging, problem
      - determination 600
    - deploying 602, 603, 604
    - deployment 593, 606
    - developing 593
    - enabling 593
    - error messages 618
    - GSS-API 603
    - GSS-API on restrictions 650
    - initialization 609
    - libraries; location of security
      - plug-in 597
    - limitations on deployment of
      - plug-ins 606
    - loading 593, 609
    - naming 597
    - overview 593
    - restrictions on libraries 613
    - return codes 615
    - SQLCODES and SQLSTATES 600
    - two-part user ID support 598
  - routines 503, 504
  - samples 651
  - security administrator authority (SECADM)
    - GRANT (Database Authorities)
      - statement 278
    - revoking 324
  - SECURITY LABEL clause
    - COMMENT statement 83
    - DROP statement 245
  - SECURITY LABEL COMPONENT clause
    - COMMENT statement 83
    - DROP statement 245
  - security labels (LBAC)
    - ALTER SECURITY LABEL
      - COMPONENT statement 14
    - CREATE SECURITY LABEL
      - COMPONENT statement 139
      - statement 137
    - GRANT (Security Label)
      - statement 297
  - policies
    - ALTER SECURITY POLICY
      - statement 17

- security labels (LBAC) *(continued)*
  - policies *(continued)*
    - CREATE SECURITY POLICY statement 141
    - REVOKE (Security Label) statement 341
- security plug-ins
  - developing 609
- SECURITY POLICY clause
  - COMMENT statement 83
  - CREATE TABLE statement 143
  - DROP statement 245
- security-label-name 565
- security-policy-name 565
- security, encrypted security-sensitive data
  - IBM Data Server Driver for JDBC and SQLJ 534
- security, encrypted user ID or encrypted password
  - IBM Data Server Driver for JDBC and SQLJ 534
- security, Kerberos
  - IBM Data Server Driver for JDBC and SQLJ 531
- security, user ID and password
  - IBM Data Server Driver for JDBC and SQLJ 529
- security, user ID-only
  - IBM Data Server Driver for JDBC and SQLJ 530
- SELECT clause
  - GRANT statement (Table, View or Nickname) 307
  - list notation
    - column reference 419
  - REVOKE statement, removing privileges 349
  - with DISTINCT keyword 419
- select list
  - application rules and syntax 419
  - description 419
  - notation rules and conventions 419
- SELECT statement
  - definition 390
  - description 390
  - examples 390
  - fullselect detailed syntax 382
  - subselects 419
  - VALUES clause 382
- select-statement SQL statement construct
  - definition 491
  - dynamic invocation 491
  - static invocation 491
- self tuning memory
  - enabling 843
- self\_tuning\_mem
  - configuration parameter 843
- seqdetect configuration parameter 845
- SEQUENCE clause
  - COMMENT statement 83
- sequence expressions
  - description 509
- sequences
  - comparison with identity columns 511
  - DROP statement 245
  - managing behavior 510
- sequences *(continued)*
  - using 509
- sequential values
  - generating 509
- serialization
  - SQL statement execution 517
- servers
  - granting privileges 302
  - names 565
- SET clause
  - UPDATE statement 354
- SET CONSTRAINTS statement 400
- SET CURRENT SQLID statement 417
- set integrity pending state 400
- SET INTEGRITY statement 400
- SET NULL delete rule 143
- set operators
  - EXCEPT, comparing differences 382
  - INTERSECT, role of AND in comparisons 382
  - UNION, correspondence to OR 382
- SET PASSTHRU statement
  - independence from COMMIT statement 365
  - independence from ROLLBACK statement 388
- SET ROLE statement 354
- SET SCHEMA statement 417
- SET SERVER OPTION statement
  - independence from COMMIT statement 365
  - independence from ROLLBACK statement 388
- SET TRANSACTION clause
  - SQLJ 519
- SETSESSIONUSER privilege
  - GRANT (SETSESSIONUSER Privilege) statement 303
  - REVOKE (SETSESSIONUSER Privilege) statement 346
- SHARE MODE connection 366
- SHARE option
  - LOCK TABLE statement 386
- shared libraries
  - rebuilding routine 547
- sheapthres configuration parameter 846
- sheapthres\_shr configuration parameter 847
- shift-in characters
  - not truncated by assignments 723
- single precision float data type 143
- SMALLINT data type
  - static SQL 143
- SMS (system managed space)
  - table spaces
    - CREATE TABLESPACE statement 212
- smtp\_server configuration parameter 848
- softmax configuration parameter 849
- sortheap database configuration parameter
  - description 850
- sorting
  - ordering of results 723
  - sort heap size configuration parameter 850
- sorting *(continued)*
  - sort heap threshold configuration parameter 846
  - sort heap threshold for shared sorts 847
  - string comparisons 723
- sources
  - embedded SQL applications 506
- special registers
  - CLIENT ACCTNG 739
  - CURRENT CLIENT\_ACCTNG 739
  - CURRENT DATE 739
  - CURRENT DECFLOAT ROUNDING MODE 740
  - CURRENT DEFAULT TRANSFORM GROUP 740
  - CURRENT DEGREE 741
  - CURRENT EXPLAIN MODE 741
  - CURRENT EXPLAIN SNAPSHOT 742
  - CURRENT FEDERATED ASYNCHRONY 743
  - CURRENT FUNCTION PATH 746
  - CURRENT IMPLICIT XMLPARSE OPTION 743
  - CURRENT ISOLATION 744
  - CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 745
  - CURRENT MDC ROLLOUT MODE 745
  - CURRENT OPTIMIZATION PROFILE 745
  - CURRENT PACKAGE PATH 745
  - CURRENT PATH 746
  - CURRENT QUERY OPTIMIZATION 747
  - CURRENT REFRESH AGE 747
  - CURRENT TIME 747
  - CURRENT TIMESTAMP 748
  - CURRENT TIMEZONE 748
  - CURRENT USER 749
- SPECIFIC FUNCTION clause
  - COMMENT statement 83
- specific name
  - definition 565
- SPECIFIC PROCEDURE clause
  - COMMENT statement 83
- spm\_log\_file\_sz configuration parameter 851
- spm\_log\_path configuration parameter 852
- spm\_max\_resync configuration parameter 853
- spm\_name configuration parameter 853
- SQL (Structured Query Language)
  - authorization
    - embedded SQL 512
- SQL objects
  - deleting 245
- SQL operations
  - basic 723
- SQL path 565
- SQL return codes 491
- SQL statements
  - ALTER AUDIT POLICY 3
  - ALTER DATABASE PARTITION GROUP 6

SQL statements (*continued*)

- ALTER FUNCTION 9
- ALTER METHOD 11
- ALTER NODEGROUP (see ALTER DATABASE PARTITION GROUP) 6
- ALTER PROCEDURE (External) 12
- ALTER SECURITY LABEL COMPONENT 14
- ALTER SECURITY POLICY 17
- ALTER TABLE 20
- ALTER TABLESPACE 66
- ALTER VIEW 78
- AUDIT 80
- COMMENT 83
- COMMIT 365
- CONNECT (Type 1) 366
- CONNECT (Type 2) 373
- control 495
- CREATE AUDIT POLICY 95
- CREATE DATABASE PARTITION GROUP 98
- CREATE FUNCTION, overview 100
- CREATE INDEX 101
- CREATE METHOD 119
- CREATE NODEGROUP (see CREATE DATABASE PARTITION GROUP) 98
- CREATE PROCEDURE (SQL) 124
- CREATE ROLE 134
- CREATE SCHEMA 135
- CREATE SECURITY LABEL 137
- CREATE SECURITY LABEL COMPONENT 139
- CREATE SECURITY POLICY 141
- CREATE TABLE 143
- CREATE TABLESPACE 212
- CREATE VIEW 225
- DELETE 239
- DISCONNECT 379
- displaying help 869
- DROP 245
- DROP TRANSFORM 245
- embedded 491
- GRANT (Database Authorities) 278
- GRANT (Exemption) 283
- GRANT (Index Privileges) 285
- GRANT (Nickname Privileges) 307
- GRANT (Package Privileges) 286
- GRANT (Role) 289
- GRANT (routine privileges) 291
- GRANT (Schema Privileges) 295
- GRANT (Security Label) 297
- GRANT (Sequence Privileges) 300
- GRANT (Server Privileges) 302
- GRANT (SETSESSIONUSER Privilege) 303
- GRANT (Table Privileges) 307
- GRANT (Table Space Privileges) 305
- GRANT (View Privileges) 307
- INSERT 313
- interactive entry 491
- invoking 491
- LOCK TABLE 386
- RENAME 321
- RENAME TABLESPACE 323
- REVOKE (Database Authorities) 324

SQL statements (*continued*)

- REVOKE (Exemption) 328
- REVOKE (Index Privileges) 330
- REVOKE (Nickname Privileges) 349
- REVOKE (Package Privileges) 331
- REVOKE (Role) 334
- REVOKE (routine privileges) 336
- REVOKE (Schema Privileges) 339
- REVOKE (Security Label) 341
- REVOKE (Sequence Privileges) 342
- REVOKE (Server Privileges) 344
- REVOKE (SETSESSIONUSER Privilege) 346
- REVOKE (Table Privileges) 349
- REVOKE (Table Space Privileges) 347
- REVOKE (View Privileges) 349
- ROLLBACK 388
- ROLLBACK TO SAVEPOINT 388
- SELECT 390
- serializing execution 517
- SET CONSTRAINTS 400
- SET INTEGRITY 400
- SET ROLE 354
- SET SCHEMA 417
- statement heap size configuration parameter 854
- TRANSFER OWNERSHIP 458
- UPDATE 354

SQL subquery

- WHERE clause 419

SQL syntax

- GROUP BY clause subselect 419
- multiple operations, order of execution 382
- SELECT clause description 419
- WHERE clause search conditions 419

SQL variable name 565

SQL/XML

- CREATE INDEX statement 101

SQL92 standard

- rules for dynamic SQL 417

SQLCA (SQL communication area)

- description 549
- entry changed by UPDATE 354
- error reporting 549
- partitioned database systems 549
- viewing interactively 549

SQLCA structure

- overview 491

SQLCODE

- description 491

SQLD field in SQLDA 555

SQLDA (SQL descriptor area)

- contents 555

SQLDABC field in SQLDA 555

SQLDAID field in SQLDA 555

SQLDATA field in SQLDA 555

SQLDATALEN field in SQLDA 555

SQLDATATYPE\_NAME field in SQLDA 555

SQLDBCON configuration file 659, 660

SQLDBCONF configuration file 659, 660

sqlAttachToCtx API 517

sqlBeginCtx API 517

sqlDetachFromCtx API 517

sqlEndCtx API 517

sqlGetCurrentCtx API 517

sqlInterruptCtx API 517

sqlSetTypeCtx API 517

SQLIND field in SQLDA 555

SQLJ

- connecting to a data source 520
- connection declaration clause 521
- context clause 520
- isolation level 519
- routines
  - connection contexts 546
  - SET TRANSACTION clause 519

SQLLEN field in SQLDA 555

SQLLONGLEN field in SQLDA 555

SQLN field in SQLDA 555

SQLNAME field in SQLDA 555

SQLSTATE

- description 491

SQLTYPE field in SQLDA 555

SQLVAR field in SQLDA 555

srv\_plugin\_mode configuration parameter 656

srvcon\_auth configuration parameter 654

srvcon\_gssplugin\_list configuration parameter 655

srvcon\_pw\_plugin configuration parameter 656

ssl\_cipherspecs 694

ssl\_cipherspecs configuration parameter 694

ssl\_clnt\_keydb 695

ssl\_clnt\_keydb configuration parameter 695

ssl\_clnt\_stash 695

ssl\_clnt\_stash configuration parameter 695

ssl\_svcname 693

ssl\_svcname configuration parameter 693

ssl\_svr\_keydb 692

ssl\_svr\_keydb configuration parameter 692

ssl\_svr\_label 693

ssl\_svr\_label configuration parameter 693

ssl\_svr\_stash 692

ssl\_svr\_stash configuration parameter 692

ssl\_versions 694

ssl\_versions configuration parameter 694

standards

- setting rules for dynamic SQL 417

start and stop timeout configuration parameter 700

start\_stop\_time configuration parameter 700

stat\_heap\_sz database configuration parameter 853

statement heap size configuration parameter 854

statements

- names 565

static SQL

- DECLARE CURSOR statement 491

- static SQL (*continued*)
  - FETCH statement 491
  - invoking 491
  - OPEN statement 491
  - select 491
  - statements 491
- STAY RESIDENT
  - CREATE PROCEDURE statement 124
- STMM (Self-Tuning Memory Manager)
  - enabling 843
- stmheap database configuration
  - parameter 854
- storage
  - backing out, unit of work, ROLLBACK 388
- storage structures
  - ALTER TABLESPACE statement 66
  - CREATE TABLESPACE statement 212
- stored procedures
  - fenced 505
  - how used 505
- strings
  - assignment conversion rules 723
- Structured Query Language (SQL)
  - assignments 723
  - basic operands, assignments and comparisons 723
  - comparison operation, overview 723
  - control statements 495
- structured types
  - DROP statement 245
  - host variables 565
- sub-total rows 419
- subqueries
  - HAVING clause 419
  - using fullselect as search condition 565
  - WHERE clause 419
- subselect
  - description 419
  - example sequence of operations 419
  - examples 419
  - FROM clause
    - subselect 419
- summary tables
  - definition 143
- super-aggregate rows 419
- super-groups 419
- supertypes
  - identifier names 565
- svcname configuration parameter 681
- symmetric super-aggregate rows 419
- SYNONYM clause
  - in place of ALIAS clause 245
- synonyms
  - DROP ALIAS statement 245
  - qualifying a column name 565
- syntax
  - common elements 497
  - elements 497
  - function designator 497
  - method designator 497
  - procedure designator 497
- sysadm\_group configuration
  - parameter 682

- sysctrl\_group configuration
  - parameter 683
- sysmaint\_group configuration
  - parameter 683
- sysmon\_group configuration
  - parameter 684
- system-containers
  - CREATE TABLESPACE statement 212

**T**

- TABLE clause
  - COMMENT statement 83
  - DROP statement 245
  - table reference 419
- TABLE HIERARCHY clause
  - DROP statement 245
- table name
  - CREATE TABLE statement 143
- table reference
  - alias 419
  - nested table expressions 419
  - nickname 419
  - table name 419
  - view name 419
- table spaces
  - adding
    - comments to catalog 83
  - creating
    - CREATE TABLESPACE statement 212
  - deleting using DROP statement 245
  - dropping
    - DROP statement 245
  - grant privileges 305
  - identification
    - CREATE TABLE statement 143
  - index
    - CREATE TABLE statement 143
  - name 565
  - page size 212
  - renaming 323
  - revoking privileges 347
- TABLE\_NAME function
  - alias 486
  - description 486
  - values and arguments 486
- TABLE\_SCHEMA function
  - alias 487
  - description 487
  - values and arguments 487
- tables
  - adding
    - columns, ALTER TABLE 20
    - comments to catalog 83
  - alias 245
  - altering 20
  - authorization for creating 143
  - correlation name 565
  - creating
    - granting authority 278
    - SQL statement instructions 143
  - deleting
    - using DROP statement 245
  - designator to avoid ambiguity 565
  - exception 400

- tables (*continued*)
  - exposed names in FROM clause 565
  - expressions
    - common table expressions 390
  - FROM clause
    - subselect naming conventions 419
  - generated columns 20
  - grant privileges 307
  - indexes 101
  - inserting rows 313
  - joining
    - partitioning key considerations 143
  - names
    - description 565
    - in ALTER TABLE statement 20
    - in FROM clause 419
    - in LOCK TABLE statement 386
    - in SELECT clause, syntax diagram 419
  - nested table expression 565
  - non-exposed names in FROM clause 565
  - qualified column name 565
  - reference 419
  - renaming 321
  - restricting shared access, LOCK TABLE statement 386
  - revoking privileges 349
  - scalar fullselect 565
  - schemas 135
  - subquery 565
  - unique correlation names 565
  - updating by row and column, UPDATE statement 354
- TABLESPACE clause
  - COMMENT statement 83
- TCP/IP service name configuration
  - parameter 681
- termination
  - unit of work 365, 388
- terms and conditions
  - use of publications 874
- territory configuration parameter 855
- threads
  - multiple
    - using in DB2 applications 517
- THREADSAFE routines 544
- time
  - deadlock configuration parameter, interval for checking 686
  - difference among nodes, maximum 699
- TIME data types
  - CREATE TABLE statement 143
- TIMESTAMP data type
  - CREATE TABLE statement 143
- Tivoli Storage Manager (TSM)
  - management class configuration parameter 859
  - node name configuration parameter 859
  - owner name configuration parameter 860
  - password configuration parameter 860



- tm\_database configuration
  - parameter 855
- TO clause
  - GRANT statement
    - database authority 278
    - index privileges 285
    - package privileges 286
    - schema privileges 295
    - table privileges 307
- toolscat\_db configuration parameter 855
- toolscat\_inst configuration parameter 856
- toolscat\_schema configuration parameter 856
- tp\_mon\_name configuration parameter 856
- track modified pages enable configuration parameter 858
- trackmod configuration parameter 858
- transaction processing monitors
  - transaction processing monitor name configuration parameter 856
- transactions
  - description 515
  - failure recovery
    - n the failed database partition server 717
    - on active database partition server 717
    - reducing the impact of failure 713
- TRANSFER OWNERSHIP statement 458
- transformations
  - DROP statement 245
- TRIGGER clause
  - COMMENT statement 83
- triggers
  - adding comments to catalog 83
  - dropping 245
  - INSERT statement 313
  - names 565
  - updates
    - UPDATE statement 354
- troubleshooting
  - online information 874
  - security plug-ins 600
  - tutorials 874
- truncation
  - numbers 723
- trust\_allclnts configuration parameter 685
- trust\_clntauth configuration parameter 858
- tsm\_mgmtclass configuration parameter 859
- tsm\_nodename configuration parameter 859
- tsm\_owner configuration parameter 860
- tsm\_password configuration parameter 860
- tutorials
  - problem determination 874
  - troubleshooting 874
  - Visual Explain 873
- two-phase commit protocol 717
- type 2 indexes 101

- TYPE clause
  - COMMENT statement 83
  - DROP statement 245
- type mapping
  - name 565
- type name 565
- typed tables
  - names 565
- typed views
  - defining subviews 225
  - names 565

## U

- undefined reference errors 565
- UNDER clause
  - CREATE VIEW statement 225
- UNION operator
  - role in comparison of fullselect 382
- UNIQUE clause
  - ALTER TABLE statement 20
  - CREATE INDEX statement 101
  - CREATE TABLE statement 143
- unique constraints
  - adding with ALTER TABLE 20
  - ALTER TABLE statement 20
  - CREATE TABLE statement 143
  - dropping with ALTER TABLE 20
- unique correlation names
  - table designators 565
- unique keys
  - ALTER TABLE statement 20
  - CREATE TABLE statement 143
- units of work (UOW) 515
  - canceling 388
  - COMMIT statement 365
  - definition 715
  - remote 516
  - ROLLBACK statement, effect 388
  - terminating 365
  - terminating without saving changes 388
- Unqualified names 565
- UPDATE clause
  - GRANT statement (Table, View or Nickname) 307
  - REVOKE statement, removing privileges 349
- UPDATE statement
  - description 354
  - row fullselect 354
- updates
  - DB2 Information Center 870, 871
  - updatable views 225
- user data
  - directories 782
- user exit enable configuration parameter 861
- user exit status indicator configuration parameter 861
- user ID and password security
  - IBM Data Server Driver for JDBC and SQLJ 529
- user ID-only security
  - IBM Data Server Driver for JDBC and SQLJ 530

- user IDs
  - two-part user IDs 598
- user\_exit\_status configuration parameter 861
- user-defined functions
  - CREATE FUNCTION statement
    - description 100
  - description 475
  - DROP statement 245
  - REVOKE (Database Authorities) statement 324
- user-defined types
  - adding comments to catalog 83
  - distinct data types
    - CREATE TABLE statement 143
  - structured types 143
- userexit database configuration parameter 861
- USING clause
  - CREATE INDEX statement 101
- util\_heap\_sz configuration parameter 861
- util\_impact\_lim configuration parameter
  - described 862

## V

- VALUES clause
  - fullselect 382
  - loading one row 313
  - rules for number of values 313
- VARCHAR data type
  - CREATE TABLE statement 143
- vendoropt configuration parameter 863
- VIEW clause
  - CREATE VIEW statement 225
  - DROP statement 245
- VIEW HIERARCHY clause
  - DROP statement 245
- view name
  - definition 565
  - in ALTER VIEW statement 78
- views
  - adding comments to catalog 83
  - alias 245
  - column names 225
  - control privilege
    - granting 307
    - limitations on 307
  - creating 225
  - deletable 225
  - deleting using DROP statement 245
  - exposed names in FROM clause 565
  - FROM clause
    - subselect naming conventions 419
  - grant privileges 307
  - inoperative 225
  - insertable 225
  - inserting rows in viewed table 313
  - names in FROM clause 419
  - names in SELECT clause
    - syntax diagram 419
  - non-exposed names in FROM clause 565
  - preventing view definition loss, WITH CHECK OPTION 354
  - qualifying a column name 565

- views (*continued*)
  - read-only 225
  - revoking privileges 349
  - schemas 135
  - updatable 225
  - updating rows by columns 354
  - WITH CHECK OPTION 354
- Vista
  - user data directories 782
- Visual Explain
  - tutorial 873

## W

- WHENEVER statement
  - changing flow of control 491
- WHERE clause
  - DELETE statement 239
  - subselect component of fullselect 419
  - UPDATE statement 354
- WITH common table expression
  - select-statement 390
- wlm\_collect\_int database configuration
  - parameter
    - description 863
- wrappers
  - names 565

## X

- XML
  - CREATE INDEX statement 101
- XML columns
  - CREATE INDEX statement 101
- XML data
  - CREATE INDEX statement 101
- XQuery statements
  - statement heap size configuration
    - parameter 854







Printed in USA

SC14-7214-00



Spine information:

IBM DB2 Version 9.7 for Linux, UNIX, and Windows

**Common Criteria Certification: Administration and User Documentation - Volume 2**

