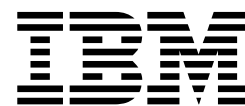
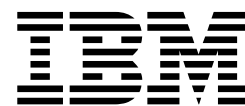


Db2 11.1 for Linux, UNIX, and Windows



Database Security Guide

Db2 11.1 for Linux, UNIX, and Windows



Database Security Guide

Notice regarding this document

This document in PDF form is provided as a courtesy to customers who have requested documentation in this format. It is provided As-Is without warranty or maintenance commitment.

Contents

Notice regarding this document	iii
--	-----

Figures	ix
-------------------	----

Tables	xi
------------------	----

Chapter 1. Db2 security model 1

Authentication	2
Authorization	3
Security considerations when installing and using the Db2 database manager	4
File permission requirements for the instance and database directories	6
Authentication details	7
Authentication methods for your server	7
Authentication considerations for remote clients	13
Partitioned database authentication considerations	13
Kerberos authentication	14
Maintaining passwords on servers	20
Authorization, privileges, and object ownership	20
Authorities overview	26
Internal system-defined routine	29
Instance level authorities	29
Database authorities	32
Privileges	41
Authorization IDs in different contexts	47
Default privileges granted on creating a database	49
Default PUBLIC privilege for built-in routines	50
Granting and revoking access	54
Controlling access for database administrators (DBAs)	60
Gaining access to data through indirect means	61
Data encryption	64
Configuring Secure Sockets Layer (SSL) support in a Db2 instance	64
IBM InfoSphere Guardium Data Encryption for encryption of data at rest	90
Database encryption using AIX encrypted file system (EFS)	93
Db2 native encryption	96
Auditing DB2 activities	116
Introduction to the Db2 audit facility	116
Audit facility management	137
Security model for the db2cluster command	141

Chapter 2. Roles 143

Creating and granting membership in roles	144
Role hierarchies	146
Effect of revoking privileges from roles	146
Delegating role maintenance by using the WITH ADMIN OPTION clause	148
Roles compared to groups	148

Using roles after migrating from IBM Informix Dynamic Server	150
--	-----

Chapter 3. Using trusted contexts and trusted connections 151

Trusted contexts and trusted connections	153
Role membership inheritance through a trusted context	156
Rules for switching the user ID on an explicit trusted connection	157
Trusted context problem determination	159

Chapter 4. Row and column access control (RCAC) overview 161

Row and column access control (RCAC) rules	162
SQL statements for managing RCAC rules	162
Built-in functions for managing RCAC permissions and masks	163
Scenario: ExampleHMO using row and column access control	163
Scenario: ExampleHMO using row and column access control - Security policies	163
Scenario: ExampleHMO using row and column access control - Database users and roles	164
Scenario: ExampleHMO using row and column access control - Database tables	165
Scenario: ExampleHMO using row and column access control - Security administration	167
Scenario: ExampleHMO using row and column access control - Row permissions	168
Scenario: ExampleHMO using row and column access control - Column masks	169
Scenario: ExampleHMO using row and column access control - Data insertion	170
Scenario: ExampleHMO using row and column access control - Data updates	171
Scenario: ExampleHMO using row and column access control - Data queries	171
Scenario: ExampleHMO using row and column access control - View creation	173
Scenario: ExampleHMO using row and column access control - Secure functions	174
Scenario: ExampleHMO using row and column access control - Secure triggers	176
Scenario: ExampleHMO using row and column access control - Revoke authority	177
Scenario: ExampleBANK using row and column access control	177
Scenario: ExampleBANK using row and column access control - Security policies	178
Scenario: ExampleBANK using row and column access control - Database users and roles	178
Scenario: ExampleBANK using row and column access control - Database tables	179

Scenario: ExampleBANK using row and column access control - Row permissions	180
Scenario: ExampleBANK using row and column access control - Column masks	181
Scenario: ExampleBANK using row and column access control - Data queries	181

Chapter 5. Label-based access control (LBAC) 183

LBAC security policies	185
LBAC security label components overview	186
LBAC security label component type: SET	187
LBAC security label component type: ARRAY	187
LBAC security label component type: TREE	188
LBAC security labels	191
Format for security label values	193
How LBAC security labels are compared	194
LBAC rule sets overview	195
LBAC rule set: DB2LBACRULES	195
LBAC rule exemptions	199
Built-in functions for managing LBAC security labels	200
Protection of data using LBAC	201
Reading of LBAC protected data	203
Inserting of LBAC protected data	205
Updating of LBAC protected data	208
Deleting or dropping of LBAC protected data	212
Removal of LBAC protection from data	215

Chapter 6. Using the system catalog for security information 217

Retrieving authorization names with granted privileges	218
Retrieving all names with DBADM authority	219
Retrieving names authorized to access a table	219
Retrieving all privileges granted to users	219
Securing the system catalog view	220

Chapter 7. Firewall support 223

Screening router firewalls	223
Application proxy firewalls	223
Circuit level firewalls	224
Stateful multi-layer inspection (SMLI) firewalls	224

Chapter 8. Security plug-ins 225

Security plug-in library locations	229
Security plug-in naming conventions	230
Security plug-in support for two-part user IDs	231
Security plug-in API versioning	233
32-bit and 64-bit considerations for security plug-ins	233
Security plug-in problem determination	233
Enabling plug-ins	235
Deploying a group retrieval plug-in	235
Deploying a user ID/password plug-in	235
Deploying a GSS-API plug-in	236
Deploying a Kerberos plug-in	238
LDAP-based authentication and group lookup support	239

Configuring transparent LDAP for authentication and group lookup (AIX)	241
Configuring transparent LDAP for authentication and group lookup (Linux)	244
Configuring the LDAP plug-in modules	245
Enabling the LDAP plug-in modules	248
Connecting with an LDAP user ID	248
Considerations for group lookup	250
Troubleshooting authenticating LDAP users or retrieving groups	250
Writing security plug-ins	251
How Db2 loads security plug-ins	251
Restrictions for developing security plug-in libraries	252
Restrictions on security plug-ins	254
Return codes for security plug-ins	256
Error message handling for security plug-ins	258
Calling sequences for the security plug-in APIs	259

Chapter 9. Security plug-in APIs 263

APIs for group retrieval plug-ins	264
db2secDoesGroupExist API - Check if group exists	265
db2secFreeErrorMsg API - Free error message memory	266
db2secFreeGroupListMemory API - Free group list memory	266
db2secGetGroupsForUser API - Get list of groups for user	267
db2secGroupPluginInit API - Initialize group plug-in	269
db2secPluginTerm - Clean up group plug-in resources	271
APIs for user ID/password authentication plug-ins	271
db2secClientAuthPluginInit API - Initialize client authentication plug-in	277
db2secClientAuthPluginTerm API - Clean up client authentication plug-in resources	278
db2secDoesAuthIDExist - Check if authentication ID exists	278
db2secFreeInitInfo API - Clean up resources held by the db2secGenerateInitialCred	279
db2secFreeToken API - Free memory held by token	280
db2secGenerateInitialCred API - Generate initial credentials	280
db2secGetAuthIDs API - Get authentication IDs	282
db2secGetDefaultLoginContext API - Get default login context	283
db2secProcessServerPrincipalName API - Process service principal name returned from server	285
db2secRemapUserid API - Remap user ID and password	286
db2secServerAuthPluginInit - Initialize server authentication plug-in	287
db2secServerAuthPluginTerm API - Clean up server authentication plug-in resources	290
db2secValidatePassword API - Validate password	290

Required APIs and definitions for GSS-API authentication plug-ins	293
Restrictions for GSS-API authentication plug-ins	294

Chapter 10. Communication buffer

exit libraries 295

Communication exit library deployment	295
Communication exit library location.	296
Naming conventions and permissions of communication exit libraries	296
Enabling communication exit libraries outside of Db2 pureScale environments	297
Enabling communication exit libraries in Db2 pureScale environments	298
Communication exit library problem determination	298
Communication exit library development	299
How a communication exit library is loaded	299
Communication exit library APIs	300
Communication buffer exit library functions structure	308
Communication buffer exit library information structure	310
Communication exit library buffer structure	311
Communication buffer exit library control over connections	311
Communication exit library API versions	311
Communication exit library error handling and return codes	312
Communication exit library development restrictions	312
Communication exit library API calling sequences	314

Chapter 11. Audit facility record

layouts 321

Audit record object types	321
-------------------------------------	-----

Audit record layout for AUDIT events	323
Audit record layout for CHECKING events	326
CHECKING access approval reasons	327
CHECKING access attempted types	329
Audit record layout for OBJMAINT events	332
Audit record layout for SECMAINT events	335
SECMAINT privileges or authorities	340
Audit record layout for SYSADMIN events	343
Audit record layout for VALIDATE events	345
Audit record layout for CONTEXT events	347
Audit record layout for EXECUTE events	348
Audit events	354

Chapter 12. Working with operating

system security 361

Db2 and Windows security	361
Authentication scenarios	362
Support for global groups (Windows)	363
User authentication and group information with DB2 on Windows	363
Defining which users hold SYSADM authority (Windows)	369
Windows LocalSystem account support	370
Extended Windows security using the DB2ADMNS and DB2USERS groups	370
Considerations for Windows7: User Access Control feature	374
Db2 and UNIX security	375
Db2 and Linux security	375
Change password support (Linux)	375
Deploying a change password plug-in (Linux)	375

Index 377

Figures

1.	Instance-level authorities	27	5.	Deploying Security Plug-ins on Db2 Clients	226
2.	Database-level authorities	28	6.	Deploying Security Plug-ins on Db2 Servers	227
3.	Object Privileges	42	7.	Authentication Using Windows Domains	362
4.	Architecture of InfoSphere Guardium Data Encryption	92			

Tables

1. Authentication Modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH Parameter Combinations.	9	23. Table T1	208
2. Internal system-defined routine needed by non-SECADM users.	29	24. Table T1 After Update	208
3. Built-in routines with no default PUBLIC privilege	51	25. Table T1 After Second Update	209
4. SSL support between the HADR primary and standby servers:	73	26. Table T1	210
5. Environment variable settings for GSKit libraries on Linux, UNIX, and Windows operating systems	99	27. Jenni's SELECT Query Result	210
6. Connection events	120	28. Jenni's UPDATE & SELECT Query Result	211
7. Audit system stored procedures and table functions	129	29. Table T1	211
8. Sequence of statements to demonstrate effect of ROLLBACK TO SAVEPOINT statement	132	30. Table T1	212
9. Example groups and users	149	31. Database Manager configuration parameters for security authentication plug-ins	228
10. Example values for a security label	193	32. Db2 security plug-ins	232
11. Summary of the DB2LBACRULES rules	195	33. Server-related values	246
12. Examples of applying the DB2LBACREADSET and DB2LBACWRITESET rules.	196	34. User-related values	246
13. Examples of applying the DB2LBACREADTREE and DB2LBACWRITETREE rules.	197	35. Group-related values	246
14. Examples of applying the DB2LBACREADARRAY rule.	198	36. Miscellaneous values	247
15. Examples of applying the DB2LBACWRITEARRAY rule.	199	37. For 64-bit UNIX and Linux systems	248
16. Example values in table T1	204	38. For 32-bit UNIX and Linux systems	248
17. Example values in view of table T1	204	39. For Windows systems (both 64-bit and 32-bit)	248
18. Example values in table T1	205	40. Security plug-in return codes	256
19. Example output from query on table T1	205	41. Required APIs and Definitions for GSS-API authentication plug-ins	293
20. Values in the example table T1 after first INSERT statement	207	42. Return codes that a communication exit library can return to the database manager.	312
21. Values in example table T1 after second INSERT statement	207	43. Audit Record Object Types Based on Audit Events	321
22. Values in example table T1 after third INSERT statement	207	44. Audit Record Layout for AUDIT Events	323
		45. Audit record layout for CHECKING events	326
		46. Audit Record Layout for OBJMAINT Events	332
		47. Audit Record Layout for SECMAINT Events	336
		48. Audit Record Layout for SYSADMIN Events	344
		49. Audit Record Layout for VALIDATE Events	346
		50. Audit Record Layout for CONTEXT Events	347
		51. Audit Record Layout for EXECUTE Events	349
		52. Successful Connection Using a Domain Controller.	369
		53. Privileges for DB2ADMNS and DB2USERS groups.	373

Chapter 1. Db2 security model

Two modes of security control access to the Db2® database system data and functions. Access to the Db2 database system is managed by facilities that reside outside the Db2 database system (authentication), whereas access within the Db2 database system is managed by the database manager (authorization).

Authentication

Authentication is the process by which a system verifies a user's identity. User authentication is completed by a security facility outside the Db2 database system, through an authentication security plug-in module. A default authentication security plug-in module that relies on operating-system-based authentication is included when you install the Db2 database system. For your convenience, the Db2 database manager also ships with authentication plug-in modules for Kerberos and lightweight directory access protocol (LDAP). To provide even greater flexibility in accommodating your specific authentication needs, you can build your own authentication security plug-in module.

The authentication process produces a Db2 authorization ID. Group membership information for the user is also acquired during authentication. Default acquisition of group information relies on an operating-system based group-membership plug-in module that is included when you install the Db2 database system. If you prefer, you can acquire group membership information by using a specific group-membership plug-in module, such as LDAP.

Authorization

After a user is authenticated, the database manager determines if that user is allowed to access Db2 data or resources. Authorization is the process whereby the Db2 database manager obtains information about the authenticated user, indicating which database operations that user can perform, and which data objects that user can access.

The different sources of permissions available to an authorization ID are as follows:

1. Primary permissions: those granted to the authorization ID directly.
2. Secondary permissions: those granted to the groups and roles in which the authorization ID is a member.
3. Public permissions: those granted to PUBLIC.
4. Context-sensitive permissions: those granted to a trusted context role.

Authorization can be given to users in the following categories:

- System-level authorization
The system administrator (SYSADM), system control (SYSCTRL), system maintenance (SYSMAINT), and system monitor (SYSMON) authorities provide varying degrees of control over instance-level functions. Authorities provide a way both to group privileges and to control maintenance and utility operations for instances, databases, and database objects.
- Database-level authorization
The security administrator (SECADM), database administrator (DBADM), database access control (ACCESSCTRL), database data access (DATAACCESS),

SQL administrator (SQLADM), workload management administrator (WLMADM), and explain (EXPLAIN) authorities provide control within the database. Other database authorities include LOAD (ability to load data into a table), and CONNECT (ability to connect to a database).

- Object-level authorization

Object level authorization involves checking privileges when an operation is performed on an object. For example, to select from a table a user must have SELECT privilege on a table (as a minimum).

- Content-based authorization

Views provide a way to control which columns or rows of a table specific users can read. Label-based access control (LBAC) determines which users have read and write access to individual rows and individual columns.

You can use these features, in conjunction with the Db2 audit facility for monitoring access, to define and manage the level of security your database installation requires.

Related information:

 [Best practices: IBM Data Server Security](#)

Authentication

Authentication of a user is completed using a security facility outside of the Db2 database system. The security facility can be part of the operating system or a separate product.

The security facility requires two items to authenticate a user: a user ID and a password. The user ID identifies the user to the security facility. By supplying the correct password, information known only to the user and the security facility, the user's identity (corresponding to the user ID) is verified.

Note: In non-root installations, operating system-based authentication must be enabled by running the **db2rfe** command.

After being authenticated:

- The user must be identified to Db2 using an SQL authorization name or *authid*. This name can be the same as the user ID, or a mapped value. For example, on UNIX operating systems, when you are using the default security plug-in module, a Db2 *authid* is derived by transforming to uppercase letters a UNIX user ID that follows Db2 naming conventions.
- A list of groups to which the user belongs is obtained. Group membership may be used when authorizing the user. Groups are security facility entities that must also map to Db2 authorization names. This mapping is done in a method similar to that used for user IDs.

The Db2 database manager uses the security facility to authenticate users in one of two ways:

- A successful security system login is used as evidence of identity, and allows:
 - Use of local commands to access local data
 - Use of remote connections when the server trusts the client authentication.
- Successful validation of a user ID and password by the security facility is used as evidence of identity and allows:
 - Use of remote connections where the server requires proof of authentication

- Use of operations where the user wants to run a command under an identity other than the identity used for login.

Note: On some UNIX systems, the Db2 database manager can log failed password attempts with the operating system, and detect when a client has exceeded the number of allowable login tries, as specified by the LOGINRETRIES parameter.

Authorization

Authorization is performed using Db2 facilities. Db2 tables and configuration files are used to record the permissions associated with each authorization name.

When an authenticated user tries to access data, these recorded permissions are compared with the permissions of:

- The authorization name of the user
- The groups to which the user belongs
- The roles granted to the user directly or indirectly through a group or a role
- The permissions acquired through a trusted context

Based on this comparison, the Db2 server determines whether to allow the requested access.

The types of permissions recorded are privileges, authority levels, and LBAC credentials.

A *privilege* defines a single permission for an authorization name, enabling a user to create or access database resources. Privileges are stored in the database catalogs.

Authority levels provide a method of grouping privileges and control over database operations. Database authorities are stored in the database catalogs; system authorities are associated with group membership, and the group names that are associated with the authority levels are stored in the database manager configuration file for a given instance.

LBAC credentials are LBAC security labels and LBAC rule exemptions that allow access to data protected by label-based access control (LBAC). LBAC credentials are stored in the database catalogs.

Groups provide a convenient means of performing authorization for a collection of users without having to grant or revoke privileges for each user individually. Unless otherwise specified, group authorization names can be used anywhere that authorization names are used for authorization purposes. In general, group membership is considered for dynamic SQL and non-database object authorizations (such as instance level commands and utilities), but is not considered for static SQL. The exception to this general case occurs when privileges are granted to PUBLIC: these are considered when static SQL is processed. Specific cases where group membership does not apply are noted throughout the Db2 documentation, where applicable.

A role is a database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC, or other roles by using a GRANT statement or to a trusted context by using a CREATE TRUSTED CONTEXT or ALTER TRUSTED CONTEXT statement. A role can be specified for the SESSION_USER ROLE connection attribute in a workload definition. When you use roles, you

associate access permissions on database objects with the roles. Users that are members of those roles then have the privileges defined for the role with which to access database objects.

Roles provide similar functionality as groups; they perform authorization for a collection of users without having to grant or revoke privileges for each user individually. One advantage of roles is that they are managed by the Db2 database system. The permissions granted to roles are taken into consideration during the authorization process for views, triggers, materialized query tables (MQTs), packages and SQL routines, unlike the permissions granted to groups. Permissions granted to groups are not considered during the authorization process for views, triggers, MQTs, packages and SQL routines, because the Db2 database system cannot discover when membership in a group changes, and so it cannot invalidate the objects mentioned previously, if appropriate.

Note: Permissions granted to roles that are granted to groups are not considered during the authorization process for views, triggers, MQTs, packages and SQL routines.

During an SQL statement processing, the permissions that the Db2 authorization model considers are the union of the following permissions:

1. The permissions granted to the primary authorization ID associated with the SQL statement
2. The permissions granted to the secondary authorization IDs (groups or roles) associated with the SQL statement
3. The permissions granted to PUBLIC, including roles that are granted to PUBLIC, directly or indirectly through other roles.
4. The permissions granted to the trusted context role, if applicable.

Security considerations when installing and using the Db2 database manager

Security considerations are important to the Db2 administrator from the moment the product is installed.

To complete the installation of the Db2 database manager, a user ID, a group name, and a password are required. The GUI-based Db2 database manager install program creates default values for different user IDs and the group. Different defaults are created, depending on whether you are installing on Linux and UNIX or Windows operating systems:

- On UNIX and Linux operating systems, if you choose to create a Db2 instance in the instance setup window, the Db2 database install program creates, by default, different users for the DAS (dasusr), the instance owner (db2inst), and the fenced user (db2fenc). Optionally, you can specify different user names

The Db2 database install program appends a number from 1-99 to the default user name, until a user ID that does not already exist can be created. For example, if the users db2inst1 and db2inst2 already exist, the Db2 database install program creates the user db2inst3. If a number greater than 10 is used, the character portion of the name is truncated in the default user ID. For example, if the user ID db2fenc9 already exists, the Db2 database install program truncates the c in the user ID, then appends the 10 (db2fen10). Truncation does not occur when the numeric value is appended to the default DAS user (for example, dasusr24).

- On Windows operating systems, the Db2 database install program creates, by default, the user db2admin for the DAS user, the instance owner, and fenced users (you can specify a different user name during setup, if you want). Unlike Linux and UNIX operating systems, no numeric value is appended to the user ID.

To minimize the risk of a user other than the administrator from learning of the defaults and using them in an improper fashion within databases and instances, change the defaults during the install to a new or existing user ID of your choice.

Note: Response file installations do not use default values for user IDs or group names. These values must be specified in the response file.

Passwords are very important when authenticating users. If no authentication requirements are set at the operating system level and the database is using the operating system to authenticate users, users will be allowed to connect. For example on Linux and UNIX operating systems, undefined passwords are treated as NULL. In this situation, any user without a defined password will be considered to have a NULL password. From the operating system's perspective, this is a match and the user is validated and able to connect to the database. Use passwords at the operating system level if you want the operating system to do the authentication of users for your database.

When working with partitioned database environments on Linux and UNIX operating systems, the Db2 database manager by default uses the rsh utility (remsh on HP-UX) to run some commands on remote members. The rsh utility transmits passwords in clear text over the network, which can be a security exposure if the Db2 server is not on a secure network. You can use the **DB2RSHCMD** registry variable to set the remote shell program to a more secure alternative that avoids this exposure. One example of a more secure alternative is ssh. See the **DB2RSHCMD** registry variable documentation for restrictions on remote shell configurations.

After installing the Db2 database manager, also review, and change (if required), the default privileges that have been granted to users. By default, the installation process grants system administration (SYSADM) privileges to the following users on each operating system:

Linux and UNIX operating systems

To a valid Db2 database user name that belongs to the primary group of the instance owner.

Windows environments

- To members of the local Administrators group.
- If the Db2 database manager is configured to enumerate groups for users at the location where the users are defined, to members of the Administrators group at the Domain Controller. You use the **DB2_GRP_LOOKUP** environment variable to configure group enumeration on Windows operating systems.
- If Windows extended security is enabled, to members of the DB2ADMNS group. The location of the DB2ADMNS group is decided during installation.
- To the LocalSystem account

By updating the database manager configuration parameter **sysadm_group**, the administrator can control which group of users possesses SYSADM privileges. You

must use the following guidelines to complete the security requirements for both the Db2 database installation and the subsequent instance and database creation.

Any group defined as the system administration group (by updating **sysadm_group**) must exist. The name of this group should allow for easy identification as the group created for instance owners. User IDs and groups that belong to this group have system administrator authority for their corresponding instances.

The administrator should consider creating an instance owner user ID that is easily recognized as being associated with a particular instance. This user ID should have as one of its groups, the name of the SYSADM group created previously. Another recommendation is to use this instance-owner user ID only as a member of the instance owner group and not to use it in any other group. This should control the proliferation of user IDs and groups that can modify the instance.

The created user ID must be associated with a password to provide authentication before being permitted entry into the data and databases within the instance. The recommendation when creating a password is to follow your organization's password naming guidelines.

Note: To avoid accidentally deleting or overwriting instance configuration or other files, administrators should consider using another user account, which does not belong to the same primary group as the instance owner, for day-to-day administration tasks that are performed on the server directly.

File permission requirements for the instance and database directories

The Db2 database system requires that your instance and database directories have a minimum level of permissions.

Note: When the instance and database directories are created by the Db2 database manager, the permissions are accurate and should not be changed.

The minimum permissions of the instance directory and the NODE000x/sqlldir directory on UNIX and Linux machines must be: u=rwx and go=rx. The meaning of the letters is explained in the following table:

Character	Represents:
u	User (owner)
g	Group
o	Other users
r	Read
w	Write
x	Execute

For example, the permissions for the instance, db2inst1, in /home are:

```
drwxr-xr-x 36 db2inst1 db2grp1          4096 Jun 15 11:13 db2inst1
```

For the directories containing the databases, each and every directory level up to and including NODE000x needs the following permissions:

```
drwxrwxr-x 11 db2inst1 db2grp1          4096 Jun 14 15:53 NODE0000/
```

For example, if a database is located in /db2/data/db2inst1/db2inst1/NODE0000 then the directories: /db2, /db2/data, /db2/data/db2inst1, /db2/data/db2inst1/db2inst1 and /db2/data/db2inst1/db2inst1/NODE0000 need drwxrwxr-x.

Within the NODE000x directory, the sqldbdir directory requires the permissions drwxrwxr-x, for example:

```
drwx-----  5 db2inst1  db2grp1          256 Jun 14 14:17 SAMPLE/
drwxr-x---   7 db2inst1  db2grp1        4096 Jun 14 13:26 SQL00001/
drwxrwxr-x   2 db2inst1  db2grp1          256 Jun 14 13:02 sqldbdir/
```

CAUTION:

To maintain the security of your files, do not change the permissions on the *DBNAME* directories (such as SAMPLE) and the *SQLxxx* directories from the permissions they are assigned when the Db2 database manager creates them.

Authentication details

Authentication methods for your server

Access to an instance or a database first requires that the user be *authenticated*. The *authentication type* for each instance determines how and where a user will be verified.

The authentication type is stored in the configuration file at the server. It is initially set when the instance is created. There is one authentication type per instance, which covers access to that database server and all the databases under its control.

If you intend to access data sources from a federated database, you must consider data source authentication processing and definitions for federated authentication types.

Note: You can check the following website for certification information about the cryptographic routines used by the Db2 database management system to perform encryption of the user ID and password when using *SERVER_ENCRYPT* authentication, and of the user ID, password, and user data when using *DATA_ENCRYPT* authentication: http://www.ibm.com/security/standards/st_evaluations.shtml.

Switching User on an Explicit Trusted Connection

For CLI/ODBC and XA CLI/ODBC applications, the authentication mechanism used when processing a switch user request that requires authentication is the same as the mechanism used to originally establish the trusted connection itself. Therefore, any other negotiated security attributes (for example, encryption algorithm, encryption keys, and plug-in names) used during the establishment of the explicit trusted connection are assumed to be the same for any authentication required for a switch user request on that trusted connection. Java™ applications allow the authentication method to be changed on a switch user request (by use of a datasource property).

Because a trusted context object can be defined such that switching user on a trusted connection does *not* require authentication, in order to take full advantage of the switch user on an explicit trusted connection feature, user-written security plug-ins must be able to:

- Accept a user ID-only token

- Return a valid Db2 authorization ID for that user ID

Note: An explicit trusted connection cannot be established if the CLIENT type of authentication is in effect.

Authentication types provided

The following authentication types are provided:

SERVER

Specifies that authentication occurs on the server through the security mechanism in effect for that configuration, for example, through a security plug-in module. The default security mechanism is that if a user ID and password are specified during the connection or attachment attempt, they are sent to the server and compared to the valid user ID and password combinations at the server to determine if the user is permitted to access the instance.

Note: The server code detects whether a connection is local or remote. For local connections, when authentication is SERVER, a user ID and password are not required for authentication to be successful.

SERVER_ENCRYPT

Specifies that the server accepts encrypted SERVER authentication schemes. If the client authentication is not specified, the client is authenticated using the method selected at the server. The user ID and password are encrypted when they are sent over the network from the client to the server.

When the resulting authentication method negotiated between the client and server is SERVER_ENCRYPT, you can choose to encrypt the user ID and password using an AES (Advanced Encryption Standard) 256-bit algorithm. To do this, set the **alternate_auth_enc** database manager configuration parameter. This configuration parameter has three settings:

- NOT_SPECIFIED (default) means that the server accepts the encryption algorithm that the client proposes, including an AES 256-bit algorithm.
- AES_CMP means that if the connecting client proposes DES but supports AES encryption, the server renegotiates for AES encryption.
- AES_ONLY means that the server accepts only AES encryption. If the client does not support AES encryption, the connection is rejected.

AES encryption can be used only when the authentication method negotiated between the client and server is SERVER_ENCRYPT.

CLIENT

Specifies that authentication occurs on the database partition where the application is invoked using operating system security. The user ID and password specified during a connection or attachment attempt are compared with the valid user ID and password combinations on the client node to determine whether the user ID is permitted access to the instance. No further authentication will take place on the database server. This is sometimes called single signon.

If the user performs a local or client login, the user is known only to that local client workstation.

If the remote instance has CLIENT authentication, two other parameters determine the final authentication type: **trust_allclnts** and **trust_clntauth**.

CLIENT level security for TRUSTED clients only:

Trusted clients are clients that have a reliable, local security system.

When the authentication type of CLIENT has been selected, an additional option might be selected to protect against clients whose operating environment has no inherent security.

To protect against unsecured clients, the administrator can select Trusted Client Authentication by setting the **trust_allclnts** parameter to NO. This implies that all trusted platforms can authenticate the user on behalf of the server. Untrusted clients are authenticated on the Server and must provide a user ID and password. You use the **trust_allclnts** configuration parameter to indicate whether you are trusting clients. The default for this parameter is YES.

Note: It is possible to trust all clients (**trust_allclnts** is YES) yet have some of those clients as those who do not have a native safe security system for authentication.

You might also want to complete authentication at the server even for trusted clients. To indicate where to validate trusted clients, you use the **trust_clntauth** configuration parameter. The default for this parameter is CLIENT.

Note: For trusted clients only, if no user ID or password is explicitly provided when attempting to CONNECT or ATTACH, then validation of the user takes place at the client. The **trust_clntauth** parameter is only used to determine where to validate the information provided on the USER or USING clauses.

To protect against all clients, including JCC type 4 clients on z/OS® and System i® but excluding native Db2 clients on z/OS, OS/390®, VM, VSE, and System i, set the **trust_allclnts** parameter to DRDAONLY. 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.

The **trust_clntauth** parameter is used to determine where the clients mentioned previously are authenticated: if **trust_clntauth** is CLIENT, authentication takes place at the client. If **trust_clntauth** is SERVER, authentication takes place at the client when no user ID and password are provided and at the server when a user ID and password are provided.

Table 1. Authentication Modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH Parameter Combinations.

trust_allclnts	trust_clntauth	Untrusted non-DRDA Client Authentication (no user ID & password)	Untrusted non-DRDA Client Authentication (with user ID & password)	Trusted non-DRDA Client Authentication (no user ID & password)	Trusted non-DRDA Client Authentication (with user ID & password)	DRDA Client Authentication (no user ID & password)	DRDA Client Authentication (with user ID & password)
YES	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT
YES	SERVER	CLIENT	SERVER	CLIENT	SERVER	CLIENT	SERVER

Table 1. Authentication Modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH Parameter Combinations. (continued)

trust_allclnts	trust_clntauth	Untrusted non-DRDA Client Authentication (no user ID & password)	Untrusted non-DRDA Client Authentication (with user ID & password)	Trusted non-DRDA Client Authentication (no user ID & password)	Trusted non-DRDA Client Authentication (with user ID & password)	DRDA Client Authentication (no user ID & password)	DRDA Client Authentication (with user ID & password)
NO	CLIENT	SERVER	SERVER	CLIENT	CLIENT	CLIENT	CLIENT
NO	SERVER	SERVER	SERVER	CLIENT	SERVER	CLIENT	SERVER
DRDAONLY	CLIENT	SERVER	SERVER	SERVER	SERVER	CLIENT	CLIENT
DRDAONLY	SERVER	SERVER	SERVER	SERVER	SERVER	CLIENT	SERVER

DATA_ENCRYPT

The server accepts encrypted SERVER authentication schemes and the encryption of user data. The authentication works the same way as that shown with SERVER_ENCRYPT. The user ID and password are encrypted when they are sent over the network from the client to the server.

The following user data are encrypted when using this authentication type:

- SQL and XQuery statements.
- SQL program variable data.
- Output data from the server processing of an SQL or XQuery statement and including a description of the data.
- Some or all of the answer set data resulting from a query.
- Large object (LOB) data streaming.
- SQLDA descriptors.

DATA_ENCRYPT_CMP

The server accepts encrypted SERVER authentication schemes and the encryption of user data. In addition, this authentication type allows compatibility with down level products not supporting 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.

KERBEROS

Used when both the Db2 client and server are on operating systems that support the Kerberos security protocol. The Kerberos security protocol performs authentication as a third party authentication service by using conventional cryptography to create a shared secret key. This key becomes a user's credential and is used to verify the identity of users during all occasions when local or network services are requested. The key eliminates the need to pass the user name and password across the network as clear text. Using the Kerberos security protocol enables the use of a single sign-on to a remote Db2 database server. The KERBEROS authentication type is supported on various operating systems, see the related information section for more information.

Kerberos authentication works as follows:

1. A user logging on to the client machine using a domain account authenticates to the Kerberos key distribution center (KDC) at the domain controller. The key distribution center issues a ticket-granting ticket (TGT) to the client.
2. During the first phase of the connection the server sends the target principal name, which is the service account name for the Db2 database server service, to the client. Using the server's target principal name and the target-granting ticket, the client requests a service ticket from the ticket-granting service (TGS) which also resides at the domain controller. If both the client's ticket-granting ticket and the server's target principal name are valid, the TGS issues a service ticket to the client. The principal name recorded in the database directory can be specified as name/instance@REALM. (This is in addition to DOMAIN\userID and userID@xxx.xxx.xxx.com formats accepted on Windows.)
3. The client sends this service ticket to the server using the communication channel (which can be, as an example, TCP/IP).
4. The server validates the client's server ticket. If the client's service ticket is valid, then the authentication is completed.

It is possible to catalog the databases on the client machine and explicitly specify the Kerberos authentication type with the server's target principal name. In this way, the first phase of the connection can be bypassed.

If a user ID and a password are specified, the client will request the ticket-granting ticket for that user account and use it for authentication.

KRB_SERVER_ENCRYPT

Specifies that the server accepts KERBEROS authentication or encrypted SERVER authentication schemes. If the client authentication is KERBEROS, the client is authenticated using the Kerberos security system. If the client authentication is SERVER_ENCRYPT, the client is authenticated using a user ID and encryption password. If the client authentication is not specified, then the client will use Kerberos if available, otherwise it will use password encryption. For other client authentication types, an authentication error is returned. The authentication type of the client cannot be specified as KRB_SERVER_ENCRYPT

Note: The Kerberos authentication types are supported on clients and servers running on specific operating systems, see the related information section for more information. For Windows operating systems, both client and server machines must either belong to the same Windows domain or belong to trusted domains. This authentication type should be used when the server supports Kerberos and some, but not all, of the client machines support Kerberos authentication.

GSSPLUGIN

Specifies that the server uses a GSS-API plug-in to perform authentication. If the client authentication is not specified, the server returns a list of server-supported plug-ins, including any Kerberos plug-in that is listed in the **srvcon_gssplugin_list** database manager configuration parameter, to the client. The client selects the first plug-in found in the client plug-in directory from the list. If the client does not support any plug-in in the list, the client is authenticated using the Kerberos authentication scheme (if it is

returned). If the client authentication is the GSSPLUGIN authentication scheme, the client is authenticated using the first supported plug-in in the list.

GSS_SERVER_ENCRYPT

Specifies that the server accepts plug-in authentication or encrypted server authentication schemes. If client authentication occurs through a plug-in, the client is authenticated using the first client-supported plug-in in the list of server-supported plug-ins.

If the client authentication is not specified and an implicit connect is being performed (that is, the client does not supply a user ID and password when making the connection), the server returns a list of server-supported plug-ins, the Kerberos authentication scheme (if one of the plug-ins in the list is Kerberos-based), and the encrypted server authentication scheme. The client is authenticated using the first supported plug-in found in the client plug-in directory. If the client does not support any of the plug-ins that are in the list, the client is authenticated using the Kerberos authentication scheme. If the client does not support the Kerberos authentication scheme, the client is authenticated using the encrypted server authentication scheme, and the connection will fail because of a missing password. A client supports the Kerberos authentication scheme if a Db2 supplied Kerberos plug-in exists for the operating system, or a Kerberos-based plug-in is specified for the **srvcon_gssplugin_list** database manager configuration parameter.

If the client authentication is not specified and an explicit connection is being performed (that is, both the user ID and password are supplied), the authentication type is equivalent to SERVER_ENCRYPT. In this case, 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.

Note:

1. Do not inadvertently lock yourself out of your instance when you are changing the authentication information, since access to the configuration file itself is protected by information in the configuration file. The following database manager configuration file parameters control access to the instance:

- **authentication ***
- **sysadm_group ***
- **trust_allclnts**
- **trust_clntauth**
- **sysctrl_group**
- **sysmaint_group**

* Indicates the two most important parameters.

There are some things that can be done to ensure this does not happen: If you do accidentally lock yourself out of the Db2 database system, you have a fail-safe option available on all platforms that will allow you to override the usual Db2 database security checks to update the database manager configuration file using a highly privileged local operating system security user. This user *always* has the privilege to update the database manager configuration file and thereby correct the problem. However, this security bypass is restricted to a local update of the database manager configuration file. You cannot use a fail-safe user remotely or for any other Db2 database command. This special user is identified as follows:

- UNIX platforms: the instance owner
- Windows platform: someone belonging to the local “Administrators” group
- Other platforms: there is no local security on the other platforms, so all users pass local security checks anyway

Authentication considerations for remote clients

When you catalog a database for remote access, you can specify the authentication type in the database directory entry.

The authentication type is not required. If it is not specified the client will try to connect using the SERVER_ENCRYPT authentication type first. If the server does not support SERVER_ENCRYPT, the server returns a list of the authentication types that it supports. The client will use the first authentication type listed to connect to the server. While unspecified, the database catalog listed using the LIST DATABASE DIRECTORY command will not show an authentication type. If the authentication type is not specified in the database directory entry then the client may take longer to connect. If an authentication type is specified, authentication can begin immediately provided that value specified matches that at the server. If a mismatch is detected, Db2 database attempts to recover. Recovery may result in more flows to reconcile the difference, or in an error if the Db2 database cannot recover. In the case of a mismatch, the value at the server is assumed to be correct.

The authentication type DATA_ENCRYPT_CMP is designed to allow clients from a previous release that does not support data encryption to connect to a server using SERVER_ENCRYPT authentication instead of DATA_ENCRYPT. This authentication does not work when the following statements are true:

- The client level is Version 7.2.
- The gateway level is Version 8 FixPak 7 or later.
- The server is Version 8 FixPak 7 or later.

When these are all true, the client cannot connect to the server. To allow the connection, you must either upgrade your client to Version 8 or later, or have your gateway level at Version 8 FixPak 6 or earlier.

The determination of the authentication type used when connecting is made by specifying the appropriate authentication type as a database catalog entry at the gateway. This is true for both Db2 Connect scenarios and for clients and servers in a partitioned database environment where the client has set the **DB2NODE** registry variable. You will catalog the authentication type at the catalog partition with the intent to “hop” to the appropriate partition. In this scenario, the authentication type cataloged at the gateway is not used because the negotiation is solely between the client and the server.

You may have a need to catalog multiple database aliases at the gateway using different authentication types if they need to have clients that use differing authentication types. When deciding which authentication type to catalog at a gateway, you can keep the authentication type the same as that used at the client and server; or, you can use the NOTSPEC authentication type with the understanding that NOTSPEC defaults to SERVER.

Partitioned database authentication considerations

In a partitioned database, each partition of the database must have the same set of users and groups defined. If the definitions are not the same, the user may be authorized to do different things on different partitions.

Consistency across all partitions is recommended.

Kerberos authentication

Kerberos is a third-party network authentication protocol that employs a system of shared secret keys to securely authenticate a user in an unsecured network environment. The Db2 database system provides support for the Kerberos authentication protocol on AIX®, HP-UX, Solaris, Linux IA32 and AMD64, and Windows operating systems.

Introduction

Kerberos authentication is managed by a three-tiered system in which encrypted service tickets, rather than a plain-text user ID and password pair, are exchanged between the application server and client. These encrypted service tickets, called *credentials*, are provided by a separate server called the Kerberos Key Distribution Center (KDC). Credentials have a finite lifetime and are understood only by the client and the server. These features reduce the risk of a security exposure, even if the ticket is intercepted from the network. Each user, or *principal* in Kerberos terms, possesses a private encryption key that is shared with the KDC. Collectively, the principals and computers that are registered with a KDC are known as a *realm*.

One key feature of Kerberos is that it provides a single sign-on environment: a user must verify identity only once to access the resources within the Kerberos realm. This single sign-on environment means that a user can connect or attach to a Db2 database server without providing a user ID or password. Another advantage is that the administration of user identification is simplified because Kerberos uses a central repository for principals. Finally, Kerberos supports mutual authentication, which enables the client to validate the identity of the server.

Setup

Before you can use Kerberos with a Db2 database system, you must install and configure the Kerberos layer on all computers. For a typical configuration, you must meet the following requirements:

- Create the appropriate principals.
- Ensure that the client and server computers and principals belong to the same realm or to trusted realms. Trusted realms are known as trusted domains in Windows terminology.
- Where appropriate, create server keytab files.
- Synchronize the time clocks on all computers. Kerberos typically permits a 5-minute time skew; if there is more than a 5-minute time skew, a preauthentication error occurs during an attempt to obtain credentials.

Setting up Kerberos for a Db2 server

Before you can use Kerberos authentication with a Db2 database system, you must install and configure the Kerberos layer on all computers. For a typical configuration, you must follow the instructions on this page.

Before you begin

If you are using a Linux, Sun Solaris, or HP-UX operating system, ensure that no Kerberos libraries other than the `krb5` library are installed on your system. Otherwise, Kerberos authentication fails, and a message is logged in the `db2diag` log files.

If you are using a Linux or Sun Solaris operating system, uninstall any instances of the IBM® Network Authentication Service (NAS) Toolkit, and remove any reference to the NAS installation path locations from the system **PATH** variable.

About this task

The use of Kerberos authentication by a Db2 database depends on whether the security authentication was successfully created using the credentials provided by the connecting application. Furthermore, whenever available, Kerberos mutual authentication is supported, where the client and server must both prove their identities to use Kerberos. However, other Kerberos features, such as the signing or encryption of messages, are unavailable.

For additional details on installing and configuring Kerberos products on your systems, see <http://www.ibm.com/developerworks/data/library/techarticle/dm-0603see/index.html>, or the documentation provided with your Kerberos product.

Kerberos support for a Db2 database system is provided through the IBMkrb5 GSS-API security plug-in. This plug-in is used for both server and client authentication. The plug-in library is installed during Db2 installation in the following locations:

- On UNIX and Linux 32-bit operating systems: the `sqllib/security32/plugin/IBM/client` and `sqllib/security32/plugin/IBM/server` directories
- On UNIX and Linux 64-bit operating systems: the `sqllib/security64/plugin/IBM/client` and `sqllib/security64/plugin/IBM/server` directories
- On Windows operating systems: the `sqllib\security\plugin\IBM\client` and `sqllib\security\plugin\IBM\server` directories

The source code for the UNIX and Linux plug-in, `IBMkrb5.C`, is available in the `sqllib/samples/security/plugins` directory. For 64-bit Windows operating systems, the plug-in library is called `IBMkrb564.dll`.

Kerberos and groups

Kerberos does not possess the concept of groups. As a result, the Db2 database instance relies upon the local operating system to obtain a group list for a Kerberos principal. For UNIX and Linux operating systems, this reliance requires an equivalent system account for each principal. For example, for the principal `name@REALM`, the Db2 database product collects group information by querying the local operating system for all group names to which the operating system user `name` belongs. If an operating system user `name` does not exist, the AUTHID belongs only to the PUBLIC group.

On Windows operating systems, a domain account is automatically associated with a Kerberos principal. The additional step of creating a separate operating system account is not required.

Kerberos keytab files

To accept security context requests, every Kerberos service on a UNIX or Linux operating system must place its credentials in a *keytab* file. This requirement applies to those principals that the Db2 database instance uses as server principals. Only the default keytab file is searched for the server key. For instructions on adding a key to the keytab file, see the documentation provided with the Kerberos product.

There is no concept of a keytab file on Windows operating systems; the system automatically handles storing and acquiring the credentials for a principal.

You can specify the default keytab file name by using the **KRB5_KTNAME** environment variable. However, because the server plug-in runs within a Db2 database engine process, this environment variable might not be accessible. To avoid this situation, add the **KRB5_KTNAME** environment variable to the **DB2ENVLIST** registry variable using the **db2set** command:

```
db2set DB2ENVLIST=KRB5_KTNAME
```

As keytab files are not used by Kerberos for Windows, this option is only available for a Linux or UNIX server.

Procedure

To set up Kerberos for a Db2 server:

1. Install Kerberos by performing one of the following steps:
 - For AIX operating systems, install the NAS (Network Authentication Services) Toolkit for Db2 on AIX, Version 1.4 or later. You can download the NAS package from <https://www.ibm.com/services/forms/preLogin.do?source=dm-nas>.
 - For Linux and HP-UX (64-bit only) operating systems, install the Kerberos package, **krb5**, that is included on your operating system installation media.
 - For Sun Solaris operating systems, the Kerberos service is included in the Solaris 10 release. No additional installation is required.
 - For Windows operating systems, enable the Active Directory on your domain controller.
2. Configure the Db2 product to use the Kerberos plug-in. See “Deploying a Kerberos plug-in” on page 238.
3. Restart the Db2 server.

Naming and mapping for Kerberos

Before you can use Kerberos with a Db2 database system, you must ensure that the client and server computers and principals belong to the same realm or to trusted realms.

Client principals

Any unique identity that can receive Kerberos tickets for authentication is known as a *principal*. A Kerberos principal identity is defined by either a two-part or multipart format, either *name@REALM* or *name/instance@REALM*. Because the *name* component is used in the authorization ID (AUTHID) mapping, the *name* must adhere to the Db2 database naming rules. Those rules limit a name to 128 characters and restrict the choice of characters.

Note: Windows operating systems directly associate a Kerberos principal identity with a domain user. An implication is that Kerberos authentication is unavailable to Windows operating systems that are not associated with a domain or realm. Furthermore, Windows operating systems support only the two-part format for defining principal identities, that is, *name@domain*.

Authorization ID mapping

Unlike operating system user IDs, whose scope of existence is usually restricted to a single computer, Kerberos principals can be authenticated in realms other than their own. You can avoid the potential problem of duplicate principal names by using the realm name to fully qualify the principal name. In Kerberos, a fully qualified principal name takes the following form:

name/instance@REALM

where *instance* can be multiple instance names separated by a forward slash (/), for example, *name/instance1/instance2@REALM*. Alternatively, you can omit the *instance* field.

The realm name must be unique within all the realms that are defined within a network. A one-to-one mapping is needed between the authorization ID and the principal name, that is, the *name* field in the fully qualified principal. This simple mapping is needed because the authorization ID is used as the default schema by the Db2 database manager and should be easily and logically derived. Be aware of the potential issues caused by the following mappings:

- Principals with the same name but from different realms are mapped to the same authorization ID. For example, the following two principal names both map to an authorization ID of gregor1x:
 - gregor1x@EXAMPLE.COM
 - gregor1x@WWW.COM
- Principals with the same name but on different instances are mapped to the same authorization ID. For example, the following two principal names both map to an authorization ID of gregor1x:
 - gregor1x/bigmachine@EXAMPLE.COM
 - gregor1x/littlemachine@EXAMPLE.COM

Therefore, follow these guidelines:

- Maintain a unique namespace for a name in all the trusted realms that access the Db2 database server.
- Make all principals with the same *name* field, regardless of the instance, belong to the same user.

Server principals

On UNIX and Linux operating systems, the server principal name for the Db2 database instance is assumed to be *instance name/fully qualified hostname@REALM*. This principal must be able to accept Kerberos security contexts, and it must exist before you start the Db2 database instance, because the server name is reported to the Db2 database instance by the plug-in at initialization time.

On Windows operating systems, the server principal is usually identified by the domain account that is used to start the Db2 database service. An exception to this situation is when the instance is started by the LocalSystem account. In this case, the server principal name is reported as *host/hostname*. This identity is valid only if both the client and server belong to Windows domains.

Windows operating systems do not support names that have more than two parts. For example: *component/component@REALM*. This creates an issue when a Windows client attempts to connect to a UNIX server. As a result, if you require

interoperability with UNIX Kerberos, you must create a mapping between the Kerberos principal and a Windows account in the Windows domain. For instructions, see the appropriate Windows documentation.

You can override the Kerberos server principal name that is used by the Db2 server on UNIX and Linux operating systems by setting the **DB2_KRB5_PRINCIPAL** environment variable to the fully qualified server principal name. The replacement server principal name is recognized by the Db2 database system only after you restart the instance by issuing the **db2start** command.

Kerberos authentication enablement

Before you can use Kerberos with a Db2 database system, you must enable Kerberos authentication.

Enabling Kerberos authentication on the client

To enable Kerberos authentication on the client, set the **clnt_krb_plugin** database manager configuration parameter to the name of the Kerberos plug-in that you are using.

For local authorizations, the client will use Kerberos if the **authentication** configuration parameter is set to **KERBEROS** or **KRB_SERVER_ENCRYPT**. Otherwise, no client-side Kerberos support is assumed.

Important: No checks are performed to validate that Kerberos support is available.

To enable Kerberos authentication on outbound connections to a Db2 server, you instead specify Kerberos as the authentication type when you catalog the database, as shown in the following example:

```
CATALOG DATABASE testdb AT NODE testnode
AUTHENTICATION KERBEROS TARGET PRINCIPAL
service/host@REALM
```

However, if you do not provide authentication information, the server sends the name of the server principal to the client.

Enabling Kerberos authentication on the server

To enable Kerberos authentication on the server, include the specific Kerberos plug-in name in the list of plug-ins that you specify for the **srvcon_gssplugin_list** database manager configuration parameter on the server. Having the Kerberos plug-in name in this list enables the client to scan the server and select the Kerberos authentication method when making a connection.

If this configuration parameter is left empty and you set the **authentication** configuration parameter to **KERBEROS** or **KRB_SERVER_ENCRYPT**, the default Kerberos plug-in, **IBMkrb5**, is used instead. You can specify only one Kerberos plug-in.

Finally, to use Kerberos for authorization of incoming connections only, set the **svrcon_auth** parameter to one of the following two options:

- **KERBEROS** to use only Kerberos authentication; or
- **KRB_SERVER_ENCRYPT** to use Kerberos and **SERVER_ENCRYPT** authorization.

If you want to use Kerberos for incoming connections and local authorizations, leave the **svrcon_auth** configuration parameter empty and set the value of the **authentication** configuration parameter to one of the Kerberos options.

Kerberos plug-in creation

To customize the behavior of Kerberos authentication on a Db2 database system, you can develop your own Kerberos authentication plug-ins.

Consider the following points when creating a Kerberos plug-in:

- Write the Kerberos plug-in as a GSS-API plug-in, but in the initialization function, set the *plugintype* variable to DB2SEC_PLUGIN_TYPE_KERBEROS for the function pointer array that is returned to the Db2 database instance.
- Under certain conditions, the server reports the server principal name to the client. The Kerberos plug-in must specify principals in the GSS_C_NT_USER_NAME format (that is, *server/host@REALM*). The GSS_C_NT_HOSTBASED_SERVICE format (that is, *service@host*) is not supported.

Kerberos compatibility

Db2 Kerberos authentication is compatible with IBM System z®, IBM i, and Windows systems.

IBM System z and IBM i compatibility

To connect to a database on an IBM System z or IBM i system, you must catalog the database by using the **AUTHENTICATION** and **KERBEROS TARGET PRINCIPAL** parameters of the **CATALOG DATABASE** command.

Neither IBM System z nor IBM i operating systems support the mutual authentication security feature of Kerberos.

Windows issues

When you are using Kerberos on Windows operating systems, be aware of the following issues:

- Due to the manner in which Windows operating systems detect and report some errors, the following conditions result in a client security plug-in error.
 - Expired account
 - Invalid password
 - Expired password
 - Password change forced by administrator
 - Disabled account

Furthermore, in all cases, the Db2 administration log or the **db2diag** log files contain Logon failed or Logon denied messages.

- If a domain account name is also defined locally, connections explicitly specifying the domain name and password fail with the following error: The Local Security Authority cannot be contacted. The error is a result of the Windows operating system locating the local user first. The solution is to fully qualify the user in the connection string, for example *name@DOMAIN.IBM.COM*.
- Windows accounts cannot include the at sign (@) character in their names because the Db2 Kerberos plug-in assumes that the character is the domain name separator.
- If the client and server are both on the Windows operating system, you can start the Db2 service using the LocalSystem account. However, if the client and server are in different domains, the connection can fail with an invalid target principal name error. To avoid this error, explicitly catalog the target principal on the client with the **CATALOG DATABASE** command, using the fully qualified server host name and the fully qualified domain name. Use the following format:

host/server hostname@server domain name. For example, `host248/server34.toronto.ibm.com@TORONTO.IBM.COM`. An alternative to using the `LocalSystem` account is to use a valid domain account.

Maintaining passwords on servers

You might be required to perform password maintenance tasks. Because such tasks are typically required at the server, and many users are not able or comfortable working with the server environment, performing these tasks can pose a significant challenge. The Db2 database system provides a way to update and verify passwords without having to be at the server.

You can assign new passwords when you connect to databases 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 IBM i V6R1.

For example, if an error message SQL1404N “Password expired” or SQL30082N “Security processing failed with reason 1 (PASSWORD EXPIRED)” is received, use the `CONNECT` statement to change the password as follows:

```
CONNECT TO database USER userid USING  
      password NEW new_password CONFIRM new_password
```

Authorization, privileges, and object ownership

Users (identified by an authorization ID) can successfully execute operations only if they have the authority to perform the specified function. To create a table, a user must be authorized to create tables; to alter a table, a user must be authorized to alter the table; and so forth.

The database manager requires that each user be specifically authorized to use each database function needed to perform a specific task. A user can acquire the necessary authorization through a grant of that authorization to their user ID or through membership in a role or a group that holds that authorization.

There are three forms of authorization, *administrative authority*, *privileges*, and *LBAC credentials*. In addition, ownership of objects brings with it a degree of authorization on the objects created. These forms of authorization are discussed in the following section.

Administrative authority

The person or persons holding administrative authority are charged with the task of controlling the database manager and are responsible for the safety and integrity of the data.

System-level authorization

The system-level authorities provide varying degrees of control over instance-level functions:

- **SYSADM** (system administrator) authority

The **SYSADM** (system administrator) authority provides control over all the resources created and maintained by the database manager. The system administrator possesses all the authorities of **SYSCTRL**, **SYSMAINT**, and **SYSMON** authority. The user who has **SYSADM** authority is responsible both for controlling the database manager, and for ensuring the safety and integrity of the data.

- **SYSCTRL authority**
The SYSCTRL authority provides control over operations that affect system resources. For example, a user with SYSCTRL authority can create, update, start, stop, or drop a database. This user can also start or stop an instance, but cannot access table data. Users with SYSCTRL authority also have SYSMON authority.
- **SYSMAINT authority**
The SYSMAINT authority provides the authority required to perform maintenance operations on all databases associated with an instance. A user with SYSMAINT authority can update the database configuration, backup a database or table space, restore an existing database, and monitor a database. Like SYSCTRL, SYSMAINT does not provide access to table data. Users with SYSMAINT authority also have SYSMON authority.
- **SYSMON (system monitor) authority**
The SYSMON (system monitor) authority provides the authority required to use the database system monitor.

Database-level authorization

The database level authorities provide control within the database:

- **DBADM (database administrator)**
The DBADM authority level provides administrative authority over a single database. This database administrator possesses the privileges required to create objects and issue database commands.
The DBADM authority can be granted only by a user with SECADM authority. The DBADM authority cannot be granted to PUBLIC.
- **SECADM (security administrator)**
The SECADM authority level provides administrative authority for security over a single database. The security administrator authority possesses the ability to manage database security objects (database roles, audit policies, trusted contexts, security label components, and security labels) and grant and revoke all database privileges and authorities. A user with SECADM authority can transfer the ownership of objects that they do not own. They can also use the AUDIT statement to associate an audit policy with a particular database or database object at the server.
The SECADM authority has no inherent privilege to access data stored in tables. It can only be granted by a user with SECADM authority. The SECADM authority cannot be granted to PUBLIC.
- **SQLADM (SQL administrator)**
The SQLADM authority level provides administrative authority to monitor and tune SQL statements within a single database. It can be granted by a user with ACCESSCTRL or SECADM authority.
- **WLMADM (workload management administrator)**
The WLMADM authority provides administrative authority to manage workload management objects, such as service classes, work action sets, work class sets, and workloads. It can be granted by a user with ACCESSCTRL or SECADM authority.
- **EXPLAIN (explain authority)**
The EXPLAIN authority level provides administrative authority to explain query plans without gaining access to data. It can only be granted by a user with ACCESSCTRL or SECADM authority.

- ACCESSCTRL (database access control authority)

The ACCESSCTRL authority level provides administrative authority to issue the following GRANT (and REVOKE) statements.

- GRANT (Database Authorities)

ACCESSCTRL authority does not give the holder the ability to grant ACCESSCTRL, DATAACCESS, DBADM, or SECADM authority. Only a user who has SECADM authority can grant these authorities.

- GRANT (Global Variable Privileges)
- GRANT (Index Privileges)
- GRANT (Module Privileges)
- GRANT (Package Privileges)
- GRANT (Routine Privileges)
- GRANT (Schema Privileges)
- GRANT (Sequence Privileges)
- GRANT (Server Privileges)
- GRANT (Table, View, or Nickname Privileges)
- GRANT (Table Space Privileges)
- GRANT (Workload Privileges)
- GRANT (XSR Object Privileges)

ACCESSCTRL authority can only be granted by a user with SECADM authority. The ACCESSCTRL authority cannot be granted to PUBLIC.

- DATAACCESS (database data access authority)

The DATAACCESS authority level provides the following privileges and authorities.

- LOAD authority
- SELECT, INSERT, UPDATE, DELETE privilege on tables, views, nicknames, and materialized query tables
- EXECUTE privilege on packages
- EXECUTE privilege on modules
- EXECUTE privilege on routines
Except on the audit routines: AUDIT_ARCHIVE, AUDIT_LIST_LOGS, AUDIT_DELIM_EXTRACT.
- READ privilege on all global variables and WRITE privilege on all global variables except variables which are read-only
- USAGE privilege on all XSR objects
- USAGE privilege on all sequences

It can be granted only by a user who holds SECADM authority. The DATAACCESS authority cannot be granted to PUBLIC.

- Database authorities (non-administrative)

To perform activities such as creating a table or a routine, or for loading data into a table, specific database authorities are required. For example, the LOAD database authority is required for use of the **load** utility to load data into tables (a user must also have the privilege to insert data into the table).

Privileges

A privilege is a permission to perform an action or a task. Authorized users can create objects, have access to objects they own, and can pass on privileges on their own objects to other users by using the GRANT statement.

Privileges may be granted to individual users, to groups, or to PUBLIC. PUBLIC is a special group that consists of all users, including future users. Users that are members of a group will indirectly take advantage of the privileges granted to the group, where groups are supported.

The CONTROL privilege: Possessing the CONTROL privilege on an object allows a user to access that database object, and to grant and revoke privileges to or from other users on that object.

Note: The CONTROL privilege only applies to tables, views, nicknames, indexes, and packages.

If a different user requires the CONTROL privilege to that object, a user with SECADM or ACCESSCTRL authority could grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked from the object owner, however, the object owner can be changed by using the TRANSFER OWNERSHIP statement.

Individual privileges: Individual privileges can be granted to allow a user to carry out specific tasks on specific objects. Users with the administrative authorities ACCESSCTRL or SECADM, or with the CONTROL privilege, can grant and revoke privileges to and from users.

Individual privileges and database authorities allow a specific function, but do not include the right to grant the same privileges or authorities to other users. The right to grant table, view, schema, package, routine, and sequence privileges to others can be extended to other users through the WITH GRANT OPTION on the GRANT statement. However, the WITH GRANT OPTION does not allow the person granting the privilege to revoke the privilege once granted. You must have SECADM authority, ACCESSCTRL authority, or the CONTROL privilege to revoke the privilege.

Privileges on objects in a package or routine: When a user has the privilege to execute a package or routine, they do not necessarily require specific privileges on the objects used in the package or routine. If the package or routine contains static SQL or XQuery statements, the privileges of the owner of the package are used for those statements. If the package or routine contains dynamic SQL or XQuery statements, the authorization ID used for privilege checking depends on the setting of the **DYNAMICRULES BIND** option of the package issuing the dynamic query statements, and whether those statements are issued when the package is being used in the context of a routine (except on the audit routines: AUDIT_ARCHIVE, AUDIT_LIST_LOGS, AUDIT_DELIM_EXTRACT).

A user or group can be authorized for any combination of individual privileges or authorities. When a privilege is associated with an object, that object must exist. For example, a user cannot be given the SELECT privilege on a table unless that table has previously been created.

Note: Care must be taken when an authorization name representing a user or a group is granted authorities and privileges and there is no user, or group created

with that name. At some later time, a user or a group can be created with that name and automatically receive all of the authorities and privileges associated with that authorization name.

The REVOKE statement is used to revoke previously granted privileges. The revoking of a privilege from an authorization name revokes the privilege granted by all authorization names.

Revoking a privilege from an authorization name does not revoke that same privilege from any other authorization names that were granted the privilege by that authorization name. For example, assume that CLAIRE grants SELECT WITH GRANT OPTION to RICK, then RICK grants SELECT to BOBBY and CHRIS. If CLAIRE revokes the SELECT privilege from RICK, BOBBY and CHRIS still retain the SELECT privilege.

LBAC credentials

Label-based access control (LBAC) lets the security administrator decide exactly who has write access and who has read access to individual rows and individual columns. The security administrator configures the LBAC system by creating security policies. A security policy describes the criteria used to decide who has access to what data. Only one security policy can be used to protect any one table but different tables can be protected by different security policies.

After creating a security policy, the security administrator creates database objects, called security labels and exemptions that are part of that policy. A security label describes a certain set of security criteria. An exemption allows a rule for comparing security labels not to be enforced for the user who holds the exemption, when they access data protected by that security policy.

Once created, a security label can be associated with individual columns and rows in a table to protect the data held there. Data that is protected by a security label is called protected data. A security administrator allows users access to protected data by granting them security labels. When a user tries to access protected data, that user's security label is compared to the security label protecting the data. The protecting label blocks some security labels and does not block others.

Object ownership

When an object is created, one authorization ID is assigned *ownership* of the object. Ownership means the user is authorized to reference the object in any applicable SQL or XQuery statement.

When an object is created within a schema, the authorization ID of the statement must have the required privilege to create objects in the implicitly or explicitly specified schema. That is, the authorization name must either be the owner of the schema, or possess the CREATEIN privilege on the schema.

Note: This requirement is not applicable when creating table spaces, buffer pools or database partition groups. These objects are not created in schemas.

When an object is created, the authorization ID of the statement is the definer of that object and by default becomes the owner of the object after it is created.

Note: One exception exists. If the AUTHORIZATION option is specified for the CREATE SCHEMA statement, any other object that is created as part of the

CREATE SCHEMA operation is owned by the authorization ID specified by the AUTHORIZATION option. Any objects that are created in the schema after the initial CREATE SCHEMA operation, however, are owned by the authorization ID associated with the specific CREATE statement.

For example, the statement `CREATE SCHEMA SCOTTSTUFF AUTHORIZATION SCOTT` `CREATE TABLE T1 (C1 INT)` creates the schema SCOTTSTUFF and the table SCOTTSTUFF.T1, which are both owned by SCOTT. Assume that the user BOBBY is granted the CREATEIN privilege on the SCOTTSTUFF schema and creates an index on the SCOTTSTUFF.T1 table. Because the index is created after the schema, BOBBY owns the index on SCOTTSTUFF.T1.

Privileges are assigned to the object owner based on the type of object being created:

- The CONTROL privilege is implicitly granted on newly created tables, indexes, and packages. This privilege allows the object creator to access the database object, and to grant and revoke privileges to or from other users on that object. If a different user requires the CONTROL privilege to that object, a user with ACCESSCTRL or SECADM authority must grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked by the object owner.
- The CONTROL privilege is implicitly granted on newly created views if the object owner has the CONTROL privilege on all the tables, views, and nicknames referenced by the view definition.
- Other objects like triggers, routines, sequences, table spaces, and buffer pools do not have a CONTROL privilege associated with them. The object owner does, however, automatically receive each of the privileges associated with the object and those privileges are with the WITH GRANT OPTION, where supported. Therefore the object owner can provide these privileges to other users by using the GRANT statement. For example, if USER1 creates a table space, USER1 automatically has the USEAUTH privilege with the WITH GRANT OPTION on this table space and can grant the USEAUTH privilege to other users. In addition, the object owner can alter, add a comment on, or drop the object. These authorizations are implicit for the object owner and cannot be revoked.

Certain privileges on the object, such as altering a table, can be granted by the owner, and can be revoked from the owner by a user who has ACCESSCTRL or SECADM authority. Certain privileges on the object, such as commenting on a table, cannot be granted by the owner and cannot be revoked from the owner. Use the TRANSFER OWNERSHIP statement to move these privileges to another user. When an object is created, the authorization ID of the statement is the definer of that object and by default becomes the owner of the object after it is created. However, when you use the **BIND** command to create a package and you specify the **OWNER** *authorization id* option, the owner of objects created by the static SQL statements in the package is the value of *authorization id*. In addition, if the AUTHORIZATION clause is specified on a CREATE SCHEMA statement, the authorization name specified after the AUTHORIZATION keyword is the owner of the schema.

A security administrator or the object owner can use the TRANSFER OWNERSHIP statement to change the ownership of a database object. An administrator can therefore create an object on behalf of an authorization ID, by creating the object using the authorization ID as the qualifier, and then using the TRANSFER OWNERSHIP statement to transfer the ownership that the administrator has on the object to the authorization ID.

Authorities overview

Various administrative authorities exist at the instance level and at the database level. These administrative authorities group together certain privileges and authorities so that you can grant them to the users who are responsible for these tasks in your database installation.

Instance level authorities

Instance level authorities enable you to perform instance-wide functions, such as creating and upgrading databases, managing table spaces, and monitoring activity and performance on your instance. No instance-level authority provides access to data in database tables. The following diagram summarizes the abilities given by each of the instance level administrative authorities:

- SYSADM -for users managing the instance as a whole
- SYSCTRL -for users administering a database manager instance
- SYSMANT -for users maintaining databases within an instance
- SYSMON -for users monitoring the instance and its databases

A user with a higher-level authority also has the abilities given by the lower level authorities. For example, a user with SYSCTRL authority can perform the functions of users with SYSMANT and SYSMON authority as well.

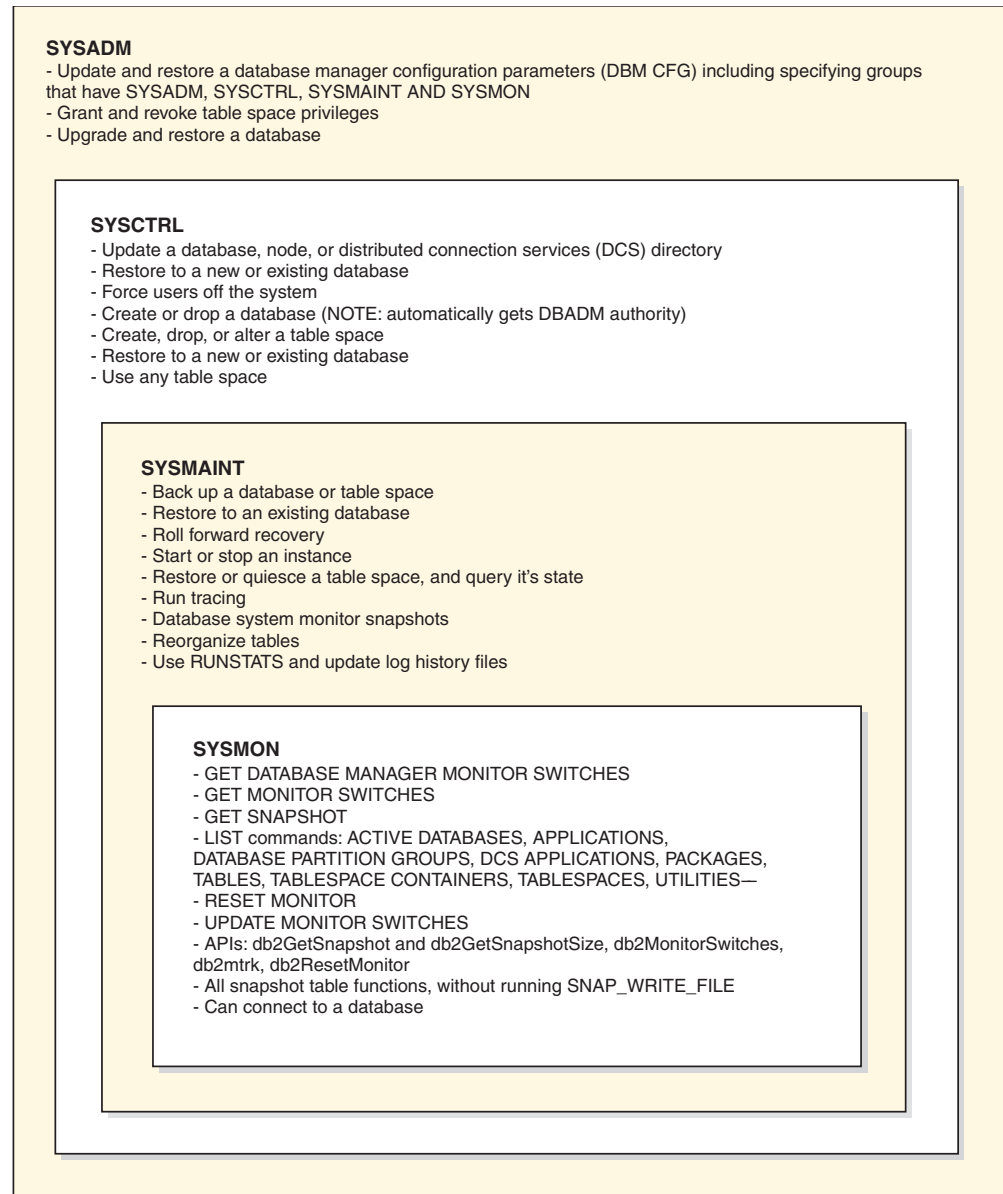


Figure 1. Instance-level authorities

Database level authorities

Database level authorities enable you to perform functions within a specific database, such as granting and revoking privileges, inserting, selecting, deleting and updating data, and managing workloads. The following diagram summarizes the abilities given by each of the database level authorities. The administrative database authorities are:

- SECADM - for users managing security within a database
- DBADM - for users administering a database
- ACCESSCTRL - for users who need to grant and revoke authorities and privileges (except for SECADM, DBADM, ACCESSCTRL, and DATAACCESS authority, SECADM authority is required to grant and revoke these authorities)
- DATAACCESS - for users who need to access data
- SQLADM - for users who monitor and tune SQL queries

- WLMADM - for users who manage workloads
- EXPLAIN - for users who need to explain query plans (EXPLAIN authority does not give access to the data itself)

The following diagram shows, where appropriate, which higher level authorities include the abilities given by a lower level authority. For example, a user with DBADM authority can perform the functions of users with SQLADM and EXPLAIN authority, and all functions except granting USAGE privilege on workloads, of users with WLMADM authority.

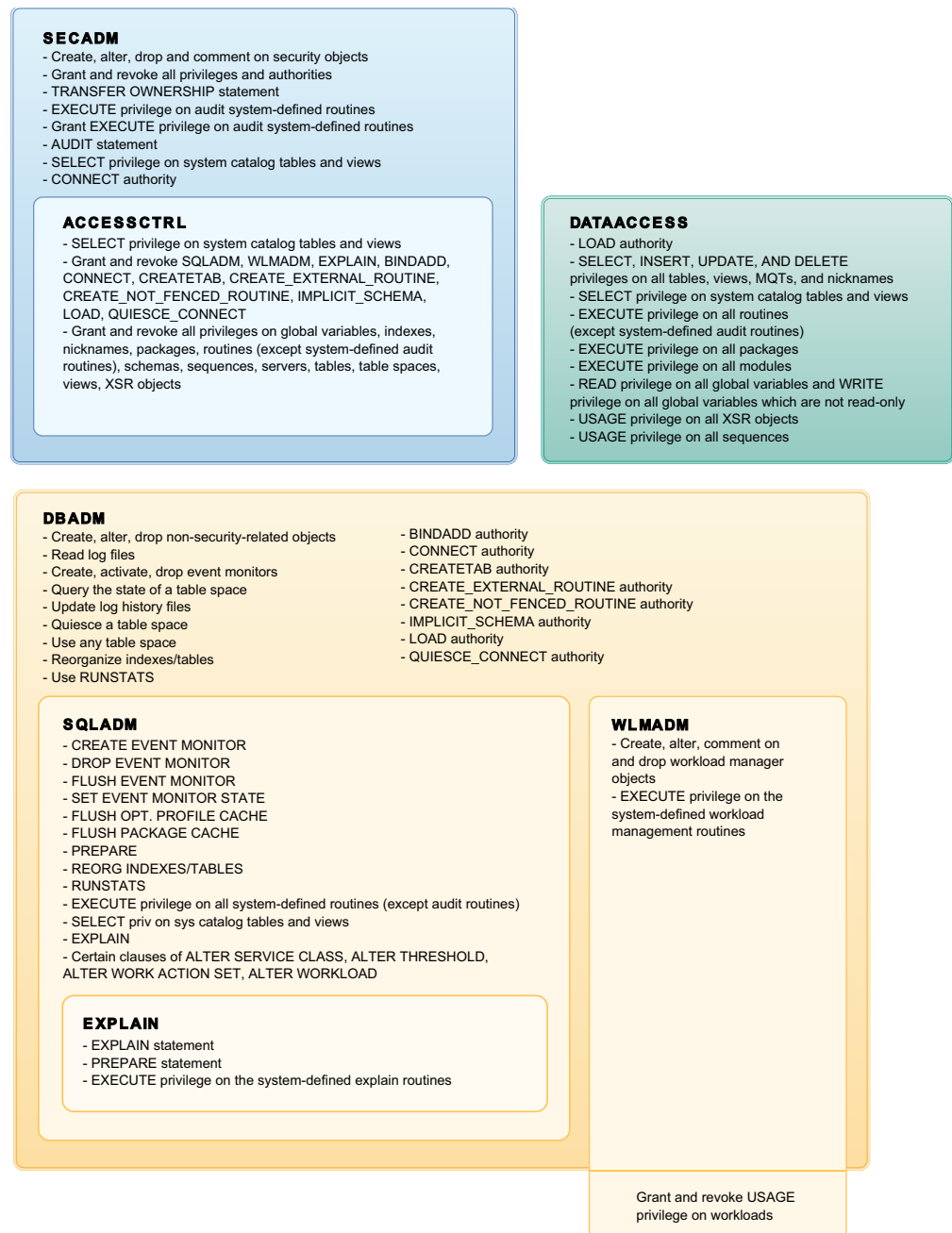


Figure 2. Database-level authorities

Internal system-defined routine

When Security Administrator (SECADM) users GRANT privileges to individual routines for users, SECADM users might come across certain internal routines. When users do not have the required privileges for these internal routines, operations that require the privilege of these internal routines might fail.

This table can be useful when deploying a restrictive database. Users can encounter missing privilege errors on certain internal routines. SECADM must consult this table and the routine description to decide whether they need to grant EXECUTE privilege on the specific internal routine that is failing with an authorization error.

This table can also be useful when the SECADM is trying to harden/secure a non-restrictive database. After you receive the report of privileges on internal routines that are granted to the special group PUBLIC, a SECADM user can consult this table to decide which internal routines still need EXECUTE privilege granted to specific users, roles, or groups.

These internal routines, their respective description, and the appropriate criteria to use them in a GRANT statement are as follows:

Table 2. Internal system-defined routine needed by non-SECADM users

Routine Name	Description

Instance level authorities

System administration authority (SYSADM)

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 assigned to the group specified by the **sysadm_group** configuration parameter. Membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSADM authority can perform the following functions:

- Upgrade a database
- Restore a database
- Change the database manager configuration file (including specifying the groups having SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority)

A user with SYSADM authority can grant and revoke table space privileges and can also use any table space.

Note: When a user with SYSADM authority creates a database, that user is automatically granted ACCESSCTRL, DATAACCESS, DBADM and SECADM authority on the database. If you want to prevent that user from accessing that database as a database administrator or a security administrator, you must explicitly revoke these database authorities from the user.

In releases before Version 9.7, SYSADM authority included implicit DBADM authority and also provided the ability to grant and revoke all authorities and privileges. In Version 9.7, the Db2 authorization model has been updated to clearly separate the duties of the system administrator, the database administrator, and the security administrator. As part of this enhancement, the abilities given by the SYSADM authority have been reduced.

In Version 9.7, only SECADM authority provides the ability to grant and revoke all authorities and privileges.

For a user holding SYSADM authority to obtain the same capabilities as in Version 9.5 (other than the ability to grant SECADM authority), the security administrator must explicitly grant the user DBADM authority and grant the user the new DATAACCESS and ACCESSCTRL authorities. These new authorities can be granted by using the GRANT DBADM ON DATABASE statement with the WITH DATAACCESS and WITH ACCESSCTRL options of that statement, which are default options. The DATAACCESS authority is the authority that allows access to data within a specific database, and the ACCESSCTRL authority is the authority that allows a user to grant and revoke privileges and non-administrative authorities within a specific database.

Considerations for the Windows LocalSystem account

On Windows systems, when the **sysadm_group** database manager configuration parameter is not specified, the LocalSystem account is considered a system administrator (holding SYSADM authority). Any Db2 application that is run by LocalSystem is affected by the change in scope of SYSADM authority in Version 9.7. These applications are typically written in the form of Windows services and run under the LocalSystem account as the service logon account. If there is a need for these applications to perform database actions that are no longer within the scope of SYSADM, you must grant the LocalSystem account the required database privileges or authorities. For example, if an application requires database administrator capabilities, grant the LocalSystem account DBADM authority using the GRANT (Database Authorities) statement. Note that the authorization ID for the LocalSystem account is SYSTEM.

System control authority (SYSCTRL)

SYSCTRL authority is the highest level of system control authority. This authority provides the ability to perform maintenance and utility operations against the database manager instance and its databases. These operations can affect system resources, but they do not allow direct access to data in the databases.

System control authority is designed for users administering a database manager instance containing sensitive data.

SYSCTRL authority is assigned to the group specified by the **sysctrl_group** configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSCTRL authority or higher can perform the following actions:

- Update a database, node, or distributed connection services (DCS) directory
- Create or drop a database
- Drop, create, or alter a table space
- Use any table space

- Restore to a new or an existing database.

In addition, a user with SYSCTRL authority can perform the functions of users with system maintenance authority (SYSMAINT) and system monitor authority (SYSMON).

Users with SYSCTRL authority also have the implicit privilege to connect to a database.

Note: When users with SYSCTRL authority create databases, they are automatically granted explicit ACCESSCTRL, DATAACCESS, DBADM, and SECADM authorities on the database. If the database creator is removed from the SYSCTRL group, and if you want to also prevent them from accessing that database as an administrator, you must explicitly revoke the four administrative authorities mentioned previously.

System maintenance authority (SYSMAINT)

SYSMAINT authority is the second level of system control authority. This authority provides the ability to perform maintenance and utility operations against the database manager instance and its databases. These operations can affect system resources, but they do not allow direct access to data in the databases.

System maintenance authority is designed for users maintaining databases within a database manager instance that contains sensitive data.

SYSMAINT authority is assigned to the group specified by the **sysmaint_group** configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSMAINT or higher system authority can perform the following actions:

- Back up a database or table space
- Restore to an existing database
- Perform roll forward recovery
- Force users off the system
- Start or stop an instance
- Restore a table space
- Run a trace, using the **db2trc** command
- Take database system monitor snapshots of a database manager instance or its databases.

A user with SYSMAINT authority can perform the following actions:

- Query the state of a table space
- Update log history files
- Quiesce a table space
- Reorganize a table
- Collect catalog statistics using the **RUNSTATS** utility.

Users with SYSMAINT authority also have the implicit privilege to connect to a database, and can perform the functions of users with system monitor authority (SYSMON).

System monitor authority (SYSMON)

SYSMON authority provides the ability to take database system monitor snapshots of a database manager instance or its databases.

SYSMON authority is assigned to the group specified by the **sysmon_group** configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

SYSMON authority enables the user to run the following commands:

- **GET DATABASE MANAGER MONITOR SWITCHES**
- **GET MONITOR SWITCHES**
- **GET SNAPSHOT**
- **LIST** (some commands):
 - **LIST ACTIVE DATABASES**
 - **LIST APPLICATIONS**
 - **LIST DATABASE PARTITION GROUPS**
 - **LIST DCS APPLICATIONS**
 - **LIST PACKAGES**
 - **LIST TABLES**
 - **LIST TABLESPACE CONTAINERS**
 - **LIST TABLESPACES**
 - **LIST UTILITIES**
- **RESET MONITOR**
- **UPDATE MONITOR SWITCHES**

SYSMON authority enables the user to use the following APIs:

- db2GetSnapshot - Get Snapshot
- db2GetSnapshotSize - Estimate Size Required for db2GetSnapshot() Output Buffer
- db2MonitorSwitches - Get/Update Monitor Switches
- db2mtrk - Memory tracker
- db2ResetMonitor - Reset Monitor

SYSMON authority enables the user use the following SQL table functions:

- All snapshot table functions without previously running `SYSPROC.SNAP_WRITE_FILE`
`SYSPROC.SNAP_WRITE_FILE` takes a snapshot and saves its content into a file. If any snapshot table functions are called with null input parameters, the file content is returned instead of a real-time system snapshot.

Important: The `SYSPROC.SNAP_WRITE_FILE` procedure is deprecated in Version 10.5 and might be removed in a future release. For more information, see “`SNAP_WRITE_FILE` procedure” in *Administrative Routines and Views*.

Database authorities

Each database authority allows the authorization ID holding it to perform some particular type of action on the database as a whole. Database authorities are different from privileges, which allow a certain action to be taken on a particular database object, such as a table or an index.

These are the database authorities.

ACCESSCTRL

Allows the holder to grant and revoke all object privileges and database

authorities except for privileges on the audit routines, and ACCESSCTRL, DATAACCESS, DBADM, and SECADM authority.

BINDADD

Allows the holder to create new packages in the database.

CONNECT

Allows the holder to connect to the database.

CREATETAB

Allows the holder to create new tables in the database.

CREATE_EXTERNAL_ROUTINE

Allows the holder to create a procedure for use by applications and other users of the database.

CREATE_NOT_FENCED_ROUTINE

Allows the holder to create a user-defined function (UDF) or procedure that is *not fenced*. CREATE_EXTERNAL_ROUTINE is automatically granted to any user who is granted CREATE_NOT_FENCED_ROUTINE.

Attention: The database manager does not protect its storage or control blocks from UDFs or procedures that are not fenced. A user with this authority must, therefore, be very careful to test their UDF extremely well before registering it as not fenced.

DATAACCESS

Allows the holder to access data stored in database tables.

DBADM

Allows the holder to act as the database administrator. In particular it gives the holder all of the other database authorities except for ACCESSCTRL, DATAACCESS, and SECADM.

EXPLAIN

Allows the holder to explain query plans without requiring them to hold the privileges to access data in the tables referenced by those query plans.

IMPLICIT_SCHEMA

Allows any user to create a schema implicitly by creating an object using a CREATE statement with a schema name that does not already exist. SYSIBM becomes the owner of the implicitly created schema and PUBLIC is given the privilege to create objects in this schema.

LOAD

Allows the holder to load data into a table

QUIESCE_CONNECT

Allows the holder to access the database while it is quiesced.

SECADM

Allows the holder to act as a security administrator for the database.

SQLADM

Allows the holder to monitor and tune SQL statements.

WLMADM

Allows the holder to act as a workload administrator. In particular, the holder of WLMADM authority can create and drop workload manager objects, grant and revoke workload manager privileges, and execute workload manager routines.

Only authorization IDs with the SECADM authority can grant the ACCESSCTRL, DATAACCESS, DBADM, and SECADM authorities. All other authorities can be granted by authorization IDs that hold ACCESSCTRL or SECADM authorities.

To remove any database authority from PUBLIC, an authorization ID with ACCESSCTRL or SECADM authority must explicitly revoke it.

Security administration authority (SECADM)

SECADM authority is the security administration authority for a specific database. This authority allows you to create and manage security-related database objects and to grant and revoke all database authorities and privileges. Additionally, the security administrator can execute, and manage who else can execute, the audit system routines.

SECADM authority has the ability to SELECT from the catalog tables and catalog views, but cannot access data stored in user tables.

SECADM authority can be granted only by the security administrator (who holds SECADM authority) and can be granted to a user, a group, or a role. PUBLIC cannot obtain the SECADM authority directly or indirectly.

The database must have at least one authorization ID of type USER with the SECADM authority. The SECADM authority cannot be revoked from every authorization ID of type USER.

SECADM authority gives a user the ability to perform the following operations:

- Create, alter, comment on, and drop:
 - Audit policies
 - Security label components
 - Security policies
 - Trusted contexts
- Create, comment on, and drop:
 - Roles
 - Security labels
- Grant and revoke database privileges and authorities
- Execute the following audit routines to perform the specified tasks:
 - The SYSPROC.AUDIT_ARCHIVE stored procedure and table function archive audit logs.
 - The SYSPROC.AUDIT_LIST_LOGS table function allows you to locate logs of interest.
 - The SYSPROC.AUDIT_DELIM_EXTRACT stored procedure extracts data into delimited files for analysis.

Also, the security administrator can grant and revoke EXECUTE privilege on these routines, therefore enabling the security administrator to delegate these tasks, if required. Only the security administrator can grant EXECUTE privilege on these routines. EXECUTE privilege WITH GRANT OPTION cannot be granted for these routines (SQLSTATE 42501).

- Change the settings of the **enclib** and **encropts** database configuration parameters.
- Execute the following security routines to perform the specified tasks:
 - The SYSPROC.ADMIN_ROTATE_MASTER_KEY stored procedure to rotate the master key for an encrypted database.

- The SYSPROC.ADMIN_GET_ENCRYPTION_INFO table function to return the encryption information about the database.
- Use of the AUDIT statement to associate an audit policy with a particular database or database object at the server
- Use of the TRANSFER OWNERSHIP statement to transfer objects not owned by the authorization ID of the statement

No other authority gives these abilities.

Only the security administrator has the ability to grant other users, groups, or roles the ACCESSCTRL, DATAACCESS, DBADM, and SECADM authorities.

In Version 9.7, the Db2 authorization model has been updated to clearly separate the duties of the system administrator, the database administrator, and the security administrator. As part of this enhancement, the abilities given by the SECADM authority have been extended. In releases before Version 9.7, SECADM authority did not provide the ability to grant and revoke all privileges and authorities. Also, SECADM authority could be granted only to a user, not to a role or a group. Additionally, SECADM authority did not provide the ability to grant EXECUTE privilege to other users on the audit built-in procedures and table function.

Database administration authority (DBADM)

DBADM authority is an administrative authority for a specific database. The database administrator possesses the privileges that are required to create objects and issue database commands. DBADM authority has SELECT privileges on system catalog tables and views, and can run all built-in Db2 routines, except audit routines and the SET_MAINT_MODE_RECORD_NO_TEMPORALHISTORY procedure.

DBADM authority can only be granted or revoked by the security administrator (who holds SECADM authority) and can be granted to a user, a group, or a role. PUBLIC cannot obtain the DBADM authority either directly or indirectly.

Holding the DBADM authority for a database allows a user to perform these actions on that database:

- Create, alter, and drop non-security related database objects
- Read log files
- Create, activate, and drop event monitors
- Query the state of a table space
- Update log history files
- Quiesce a table space
- Use any table space
- Reorganize a table
- Collect catalog statistics using the **RUNSTATS** utility

SQLADM authority and WLMADM authority are subsets of the DBADM authority. WLMADM authority has the additional ability to grant the USAGE privilege on workloads.

Granting DATAACCESS authority with DBADM authority

The security administrator can specify whether a database administrator can access data within the database. DATAACCESS authority is the authority that allows access to data within a specific database. The security administrator can use the

WITH DATAACCESS option of the GRANT DBADM ON DATABASE statement to provide a database administrator with this ability. If neither the WITH DATAACCESS or WITHOUT DATAACCESS options are specified, by default DATAACCESS authority is granted.

To grant database administrator authority without DATAACCESS authority, use GRANT DBADM WITHOUT DATAACCESS in your SQL statement.

Granting ACCESSCTRL authority with DBADM authority

The security administrator can specify whether a database administrator can grant and revoke privileges within the database. ACCESSCTRL authority is the authority that allows a user to grant and revoke privileges and non-administrative authorities within a specific database. The security administrator can use the WITH ACCESSCTRL option of the GRANT DBADM ON DATABASE statement to provide a database administrator with this ability. If neither the WITH ACCESSCTRL or WITHOUT ACCESSCTRL options are specified, by default ACCESSCTRL authority is granted.

To grant database administrator authority without ACCESSCTRL authority, use GRANT DBADM WITHOUT ACCESSCTRL in your SQL statement.

Revoking DBADM authority

If a security administrator has granted DBADM authority that includes DATAACCESS or ACCESSCTRL authority, to revoke these authorities, the security administrator must explicitly revoke DATAACCESS or ACCESSCTRL authority. For example, if the security administrator grants DBADM authority to a user:

```
GRANT DBADM ON DATABASE TO user1
```

By default, DATAACCESS and ACCESSCTRL authority are also granted to user1.

Later, the security administrator revokes DBADM authority from user1:

```
REVOKE DBADM ON DATABASE FROM user1
```

Now user1 no longer holds DBADM authority, but still has both DATAACCESS and ACCESSCTRL authority.

To revoke these remaining authorities, the security administrator needs to revoke them explicitly:

```
REVOKE ACCESSCTRL, DATAACCESS ON DATABASE FROM user1
```

Differences for DBADM authority in prior releases

In Version 9.7, the Db2 authorization model has been updated to clearly separate the duties of the system administrator, the database administrator, and the security administrator. As part of this enhancement, the abilities given by the DBADM authority have changed. In releases before Version 9.7, DBADM authority automatically included the ability to access data and to grant and revoke privileges for a database. In Version 9.7, these abilities are given by the new authorities, DATAACCESS and ACCESSCTRL as explained earlier.

Also, in releases before Version 9.7, granting DBADM authority automatically granted the following authorities too:

- BINDADD

- CONNECT
- CREATETAB
- CREATE_EXTERNAL_ROUTINE
- CREATE_NOT_FENCED_ROUTINE
- IMPLICIT_SCHEMA
- QUIESCE_CONNECT
- LOAD

Before Version 9.7, when DBADM authority was revoked these authorities were not revoked.

In Version 9.7, these authorities are now part of DBADM authority. When DBADM authority is revoked in Version 9.7, these authorities are lost.

However, if a user held DBADM authority when you upgraded to Version 9.7, these authorities are not lost if DBADM authority is revoked. Revoking DBADM authority in Version 9.7 causes a user to lose these authorities only if they acquired them through holding DBADM authority that was granted in Version 9.7.

Access control administration authority (ACCESSCTRL)

ACCESSCTRL authority is the authority required to grant and revoke privileges on objects within a specific database. ACCESSCTRL authority has no inherent privilege to access data stored in tables, except the catalog tables and views.

ACCESSCTRL authority can only be granted by the security administrator (who holds SECADM authority). It can be granted to a user, a group, or a role. PUBLIC cannot obtain the ACCESSCTRL authority either directly or indirectly. ACCESSCTRL authority gives a user the ability to perform the following operations:

- Grant and revoke the following administrative authorities:
 - EXPLAIN
 - SQLADM
 - WLMADM
- Grant and revoke the following database authorities:
 - BINDADD
 - CONNECT
 - CREATETAB
 - CREATE_EXTERNAL_ROUTINE
 - CREATE_NOT_FENCED_ROUTINE
 - IMPLICIT_SCHEMA
 - LOAD
 - QUIESCE_CONNECT
- Grant and revoke all privileges on the following objects, regardless who granted the privilege:
 - Global Variable
 - Index
 - Nickname
 - Package
 - Routine (except audit routines)
 - Schema

- Sequence
- Server
- Table
- Table Space
- View
- XSR Objects
- SELECT privilege on the system catalog tables and views

This authority is a subset of security administrator (SECADM) authority.

Data access administration authority (DATAACCESS)

DATAACCESS is the authority that allows access to data within a specific database.

DATAACCESS authority can be granted only by the security administrator (who holds SECADM authority). It can be granted to a user, a group, or a role. PUBLIC cannot obtain the DATAACCESS authority either directly or indirectly.

For all tables, views, materialized query tables, and nicknames it gives these authorities and privileges:

- LOAD authority on the database
- SELECT privilege (including system catalog tables and views)
- INSERT privilege
- UPDATE privilege
- DELETE privilege

In addition, DATAACCESS authority provides the following privileges:

- EXECUTE on all packages
- EXECUTE on all routines (except audit routines, the SET_MAINT_MODE_RECORD_NO_TEMPORALHISTORY procedure, the ADMIN_SET_MAINT_MODE procedure, and the encryption related routines ADMIN_ROTATE_MASTER_KEY and ADMIN_GET_ENCRYPTION_INFO)
- EXECUTE on all modules
- READ on all global variables and WRITE on all global variables except variables which are read-only
- USAGE on all XSR objects
- USAGE on all sequences

SQL administration authority (SQLADM)

SQLADM authority is the authority required to monitor and tune SQL statements.

SQLADM authority can be granted by the security administrator (who holds SECADM authority) or a user who possesses ACCESSCTRL authority. SQLADM authority can be granted to a user, a group, a role, or to PUBLIC. SQLADM authority gives a user the ability to perform the following functions:

- Execution of the following SQL statements:
 - CREATE EVENT MONITOR
 - DROP EVENT MONITOR
 - EXPLAIN
 - FLUSH EVENT MONITOR

- FLUSH OPTIMIZATION PROFILE CACHE
- FLUSH PACKAGE CACHE
- PREPARE
- REORG INDEXES/TABLE
- RUNSTATS
- SET EVENT MONITOR STATE

Note: If the **DB2AUTH** registry variable is set to **SQLADM_NO_RUNSTATS_REORG**, users with SQLADM authority will not be able to perform reorg or runstats operations.

- Execution of certain clauses of the following workload manager SQL statements:
 - The following clauses of the ALTER SERVICE CLASS statement:
 - COLLECT AGGREGATE ACTIVITY DATA
 - COLLECT AGGREGATE REQUEST DATA
 - COLLECT REQUEST METRICS
 - The following clause of the ALTER THRESHOLD statement
 - WHEN EXCEEDED COLLECT ACTIVITY DATA
 - The following clauses of the ALTER WORK ACTION SET statement that allow you to alter a work action:
 - ALTER WORK ACTION ... COLLECT ACTIVITY DATA
 - ALTER WORK ACTION ... COLLECT AGGREGATE ACTIVITY DATA
 - ALTER WORK ACTION ... WHEN EXCEEDED COLLECT ACTIVITY DATA
 - The following clauses of the ALTER WORKLOAD statement:
 - COLLECT ACTIVITY METRICS
 - COLLECT AGGREGATE ACTIVITY DATA
 - COLLECT LOCK TIMEOUT DATA
 - COLLECT LOCK WAIT DATA
 - COLLECT UNIT OF WORK DATA
- SELECT privilege on the system catalog tables and views
- EXECUTE privilege on all built-in Db2 routines (except audit routines and the SET_MAINT_MODE_RECORD_NO_TEMPORALHISTORY procedure)

SQLADM authority is a subset of the database administrator (DBADM) authority.

EXPLAIN authority is a subset of the SQLADM authority.

Workload administration authority (WLMADM)

WLMADM authority is the authority required to manage workload objects for a specific database. This authority allows you to create, alter, drop, comment on, and grant and revoke access to workload manager objects.

WLMADM authority can be granted by the security administrator (who holds SECADM authority) or a user who possesses ACCESSCTRL authority. WLMADM authority can be granted to a user, a group, a role, or to PUBLIC. WLMADM authority gives a user the ability to perform the following operations:

- Create, alter, comment on, and drop the following workload manager objects:
 - Histogram templates

- Service classes
- Thresholds
- Work action sets
- Work class sets
- Workloads
- Grant and revoke workload privileges
- Execute the built-in workload management routines.

WLMADM authority is a subset of the database administrator authority, DBADM.

Explain administration authority (EXPLAIN)

EXPLAIN authority is the authority required to explain query plans without gaining access to data for a specific database. This authority is a subset of the database administrator authority and has no inherent privilege to access data stored in tables.

EXPLAIN authority can be granted by the security administrator (who holds SECADM authority) or by a user who possesses ACCESSCTRL authority. The EXPLAIN authority can be granted to a user, a group, a role, or to PUBLIC. It gives the ability to execute the following SQL statements:

- EXPLAIN
- PREPARE
- DESCRIBE on output of a SELECT statement or of an XQuery statement

EXPLAIN authority also provides EXECUTE privilege on the built-in explain routines.

EXPLAIN authority is a subset of the SQLADM authority.

LOAD authority

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can use the **LOAD** command to load data into a table.

Note: Having DATAACCESS authority gives a user full access to the LOAD command.

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can **LOAD RESTART** or **LOAD TERMINATE** if the previous load operation is a load to insert data.

Users having LOAD authority at the database level, as well as the INSERT and DELETE privileges on a table, can use the **LOAD REPLACE** command.

If the previous load operation was a load replace, the DELETE privilege must also have been granted to that user before the user can **LOAD RESTART** or **LOAD TERMINATE**.

If the exception tables are used as part of a load operation, the user must have INSERT privilege on the exception tables.

The user with this authority can perform **QUIESCE TABLESPACES FOR TABLE**, **RUNSTATS**, and **LIST TABLESPACES** commands.

Implicit schema authority (IMPLICIT_SCHEMA) considerations


When a new database is created, PUBLIC is given IMPLICIT_SCHEMA database authority, unless the **RESTRICTIVE** option is specified on the **CREATE DATABASE** command.

With the IMPLICIT_SCHEMA authority, a user can create a schema by creating an object and specifying a schema name that does not exist. SYSIBM becomes the owner of the implicitly created schema and PUBLIC is given the privilege to create objects in this schema. When the database is restrictive, PUBLIC does not have the CREATEIN privilege on the schema. The user who implicitly creates the schema has CREATEIN privilege on the schema.

If control of who can implicitly create schema objects is required for the database, the database must be created with the RESTRICTIVE option specified. If the database is not restrictive, IMPLICIT_SCHEMA database authority must be revoked from PUBLIC. In this scenario, there are only three ways that a schema object is created:

- Any user can create a schema with their own authorization name on a CREATE SCHEMA statement.
- Any user with DBADM authority can explicitly create any schema which does not exist, and can optionally specify another user as the owner of the schema.
- Any user with DBADM authority has IMPLICIT_SCHEMA database authority, so that they can implicitly create a schema with any name at the time they are creating other database objects.

Related information:

 Best practices: A practical guide to restrictive databases

Privileges

Authorization ID privileges: SETSESSIONUSER

Authorization ID privileges involve actions on authorization IDs. There is currently only one such privilege: the SETSESSIONUSER privilege.

The SETSESSIONUSER privilege can be granted to a user or to a group and allows the holder to switch identities to any of the authorization IDs on which the privilege was granted. The identity switch is made by using the SQL statement SET SESSION AUTHORIZATION. The SETSESSIONUSER privilege can only be granted by a user holding SECADM authority.

Note: When you upgrade a Version 8 database to Version 9.1, or later, authorization IDs with explicit DBADM authority on that database are automatically granted SETSESSIONUSER privilege on PUBLIC. This prevents breaking applications that rely on authorization IDs with DBADM authority being able to set the session authorization ID to any authorization ID. This does not happen when the authorization ID has SYSADM authority but has not been explicitly granted DBADM.

Schema privileges

Schema privileges are in the object privilege category.

Object privileges are shown in Figure 3 on page 42.

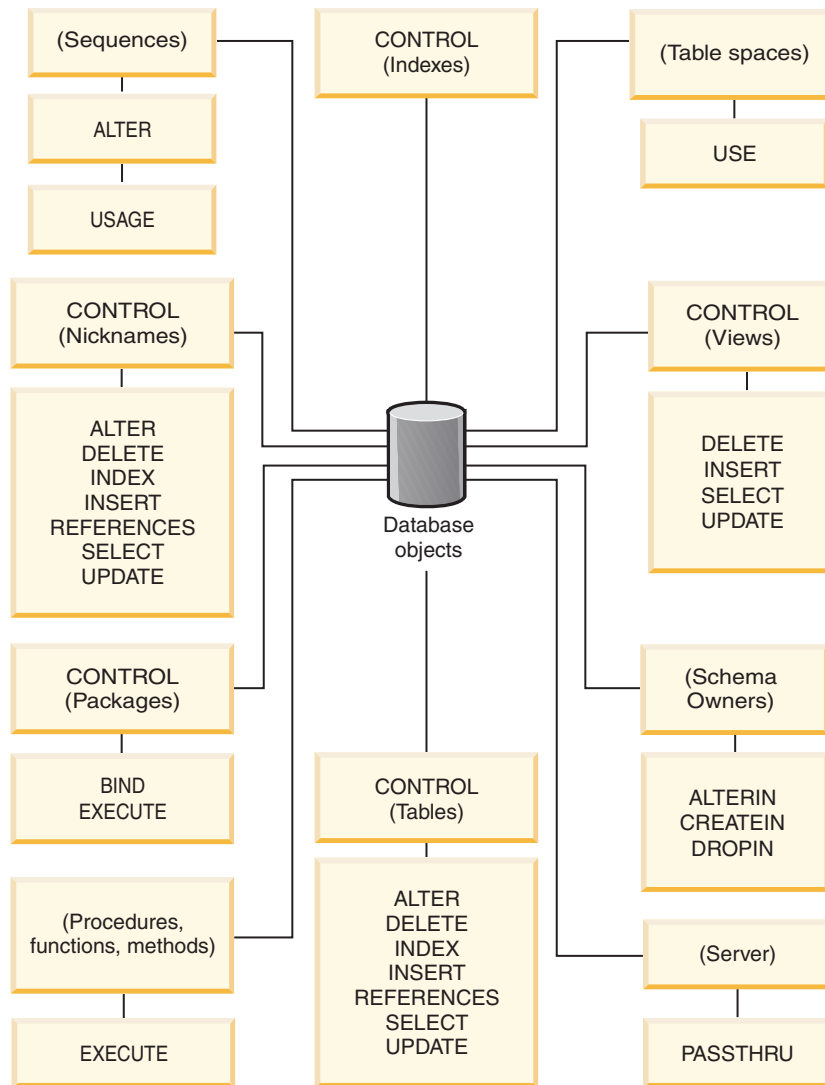


Figure 3. Object Privileges

Schema privileges involve actions on schemas in a database. A user, group, role, or PUBLIC can be granted any of the following privileges:

- CREATEIN allows the user to create objects within the schema.
- ALTERIN allows the user to alter objects within the schema.
- DROPIN allows the user to drop objects from within the schema.

The owner of the schema has all of these privileges and the ability to grant them to others. The objects that are manipulated within the schema object include: tables, views, indexes, packages, data types, functions, triggers, procedures, and aliases.

Table space privileges

The table space privileges involve actions on the table spaces in a database. A user can be granted the USE privilege for a table space, which then allows them to create tables within the table space.

The owner of the table space is granted USE privilege with the WITH GRANT OPTION on the table space when it is created. Also, users who hold SECADM or ACCESSCTRL authority have the ability to grant USE privilege on the table space.

Users who hold SYSADM or SYSCTRL authority are able to use any table space.

Upon creating a non-restrictive database, by default, the USE privilege for the table space USERSPACE1 is granted to PUBLIC. This privilege can be later revoked.

You cannot GRANT the USE privilege to SYSCATSPACE and any other system temporary table spaces.

Table and view privileges

Table and view privileges involve actions on tables or views in a database.

A user must have CONNECT authority on the database to use any of the following privileges:

- CONTROL provides the user with all privileges for a table or view including the ability to drop it, and to grant and revoke individual table privileges. You must have ACCESSCTRL or SECADM authority to grant CONTROL. The creator of a table automatically receives CONTROL privilege on the table. The creator of a view automatically receives CONTROL privilege only if they have CONTROL privilege on all tables, views, and nicknames referenced in the view definition.
- ALTER allows the user to modify on a table, for example, to add columns or a unique constraint to the table. A user with ALTER privilege can also COMMENT ON a table, or on columns of the table. For information about the possible modifications that can be performed on a table, see the ALTER TABLE and COMMENT statements.
- DELETE allows the user to delete rows from a table or view.
- INDEX allows the user to create an index on a table. Creators of indexes automatically have CONTROL privilege on the index.
- INSERT allows the user to insert a row into a table or view, and to run the **IMPORT** utility.
- REFERENCES allows the user to create and drop a foreign key, specifying the table as the parent in a relationship. The user might have this privilege only on specific columns.
- SELECT allows the user to retrieve rows from a table or view, to create a view on a table, and to run the **EXPORT** utility.
- UPDATE allows the user to change an entry in a table, a view, or for one or more specific columns in a table or view. The user may have this privilege only on specific columns.

The privilege to grant these privileges to others may also be granted using the WITH GRANT OPTION on the GRANT statement.

Note: When a user or group is granted CONTROL privilege on a table, all other privileges on that table are automatically granted WITH GRANT OPTION. If you subsequently revoke the CONTROL privilege on the table from a user, that user will still retain the other privileges that were automatically granted. To revoke all the privileges that are granted with the CONTROL privilege, you must either explicitly revoke each individual privilege or specify the ALL keyword on the REVOKE statement, for example:

```
REVOKE ALL
ON EMPLOYEE FROM USER HERON
```

When working with typed tables, there are implications regarding table and view privileges.

Note: Privileges may be granted independently at every level of a table hierarchy. As a result, a user granted a privilege on a supertable within a hierarchy of typed tables may also indirectly affect any subtables. However, a user can only operate directly on a subtable if the necessary privilege is held on that subtable.

The supertable/subtable relationships among the tables in a table hierarchy mean that operations such as SELECT, UPDATE, and DELETE will affect the rows of the operation's target table and all its subtables (if any). This behavior can be called *substitutability*. For example, suppose that you have created an Employee table of type Employee_t with a subtable Manager of type Manager_t. A manager is a (specialized) kind of employee, as indicated by the type/subtype relationship between the structured types Employee_t and Manager_t and the corresponding table/subtable relationship between the tables Employee and Manager. As a result of this relationship, the SQL query:

```
SELECT * FROM Employee
```

will return the object identifier and Employee_t attributes for both employees and managers. Similarly, the update operation:

```
UPDATE Employee SET Salary = Salary + 1000
```

will give a thousand dollar raise to managers as well as regular employees.

A user with SELECT privilege on Employee will be able to perform this SELECT operation even if they do not have an explicit SELECT privilege on Manager. However, such a user will not be permitted to perform a SELECT operation directly on the Manager subtable, and will therefore not be able to access any of the non-inherited columns of the Manager table.

Similarly, a user with UPDATE privilege on Employee will be able to perform an UPDATE operation on Manager, thereby affecting both regular employees and managers, even without having the explicit UPDATE privilege on the Manager table. However, such a user will not be permitted to perform UPDATE operations directly on the Manager subtable, and will therefore not be able to update non-inherited columns of the Manager table.

Package privileges

A package is a database object that contains the information needed by the database manager to access data in the most efficient way for a particular application program. Package privileges enable a user to create and manipulate packages.

The user must have CONNECT authority on the database to use any of the following privileges:

- CONTROL provides the user with the ability to rebind, drop, or execute a package as well as the ability to extend those privileges to others. The creator of a package automatically receives this privilege. A user with CONTROL privilege is granted the BIND and EXECUTE privileges, and can also grant these privileges to other users by using the GRANT statement. (If a privilege is granted using WITH GRANT OPTION, a user who receives the BIND or EXECUTE privilege can, in turn, grant this privilege to other users.) To grant CONTROL privilege, the user must have ACCESSCTRL or SECADM authority.
- BIND privilege on a package allows the user to rebind or bind that package and to add new package versions of the same package name and creator.
- EXECUTE allows the user to execute or run a package.

Note: All package privileges apply to all VERSIONs that share the same package name and creator.

In addition to these package privileges, the BINDADD database authority allows users to create new packages or rebind an existing package in the database.

Objects referenced by nicknames need to pass authentication checks at the data sources containing the objects. In addition, package users must have the appropriate privileges or authority levels for data source objects at the data source.

It is possible that packages containing nicknames might require additional authorization steps because Db2 database uses dynamic queries when communicating with Db2 Family data sources. The authorization ID running the package at the data source must have the appropriate authority to execute the package dynamically at that data source.

Index privileges

The creator of an index or an index specification automatically receives CONTROL privilege on the index. CONTROL privilege on an index is really the ability to drop the index. To grant CONTROL privilege on an index, a user must have ACCESSCTRL or SECADM authority.

The table-level INDEX privilege allows a user to create an index on that table.

The nickname-level INDEX privilege allows a user to create an index specification on that nickname.

Privileges on expression-based indexes:

Special consideration must be given to privileges when you use expression-based indexes.

The authorization that is required to create an index with an expression-based key is the same authorization that is required for a regular index. For details, refer to the “Authorization” section of the CREATE INDEX topic in SQL Reference Volume 2.

When you create an expression-based index, two more database objects are system-generated and associated with the index. The first is a statistical view, and the second is a package. These additional objects are not system-generated when you create a regular index. A restricted set of privileges is granted on these additional objects.

Statistical view privileges

Normally, the authorization ID must hold either SELECT or DATAACCESS privilege on the table to create a statistical view. The same privilege is required to ALTER the same table to enable query optimization for the view.

For a system-generated statistical view that is associated with an index, these privileges are not required. The statistical view is automatically created if the authorization ID has the required authority to create an index on the table. However, the set of privileges that is granted on the statistical view that is associated with an index differ from a set of privileges on a normal statistical view. Namely, no privileges are granted to any authorization ID on the statistical view, including the owner of the index. The owner of the index is also the owner of the

statistical view. No one, including authorization IDs with the SECADM or DBADM authority can modify privileges on a statistical view. An attempt to GRANT or REVOKE a privilege on the statistical view results in an error (SQLSTATE 42501).

The ability to issue RUNSTATS on the statistical view or manually update its statistics is governed by the authorities and privileges on the underlying table.

The TRANSFER OWNERSHIP operation on the statistical view is not allowed and results in SQL20344N, reason code 7. However, TRANSFER OWNERSHIP of an index with an expression-based key implicitly transfers the ownership of the associated statistical view.

Package privileges

No extra privileges are required to run any statement or command in the system-generated package. When an index is created with an expression-based key, any user with privileges on the table can use the package. That is, any user with INSERT, UPDATE, DELETE, or SELECT on the table has EXECUTE privilege on that package. This authorization is implicit as part of the statement or command that is run.

The TRANSFER OWNERSHIP operation on the package is not allowed and results in SQL20344N, reason code 5. However, TRANSFER OWNERSHIP of an index with an expression-based key implicitly transfers the ownership of the associated the package.

Sequence privileges

The creator of a sequence automatically receives the USAGE and ALTER privileges on the sequence. The USAGE privilege is needed to use NEXT VALUE and PREVIOUS VALUE expressions for the sequence.

To allow other users to use the NEXT VALUE and PREVIOUS VALUE expressions, sequence privileges must be granted to PUBLIC. This allows all users to use the expressions with the specified sequence.

ALTER privilege on the sequence allows the user to perform tasks such as restarting the sequence or changing the increment for future sequence values. The creator of the sequence can grant the ALTER privilege to other users, and if WITH GRANT OPTION is used, these users can, in turn, grant these privileges to other users.

Routine privileges

Execute privileges involve actions on all types of routines such as functions, procedures, and methods within a database. Once having EXECUTE privilege, a user can then invoke that routine, create a function that is sourced from that routine (applies to functions only), and reference the routine in any DDL statement such as CREATE VIEW or CREATE TRIGGER.

The user who defines the externally stored procedure, function, or method receives EXECUTE WITH GRANT privilege. If the EXECUTE privilege is granted to another user via WITH GRANT OPTION, that user can, in turn, grant the EXECUTE privilege to another user.

Usage privilege on workloads

To enable use of a workload, a user who holds ACCESSCTRL, SECADM, or WLMADM authority can grant USAGE privilege on that workload to a user, a group, or a role using the GRANT USAGE ON WORKLOAD statement.

When the Db2 database system finds a matching workload, it checks whether the session user has USAGE privilege on that workload. If the session user does not have USAGE privilege on that workload, then the Db2 database system searches for the next matching workload in the ordered list. In other words, the workloads that the session user does not have USAGE privilege on are treated as if they do not exist.

The USAGE privilege information is stored in the catalogs and can be viewed through the SYSCAT.WORKLOADAUTH view.

The USAGE privilege can be revoked using the REVOKE USAGE ON WORKLOAD statement.

Users with the ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM authority implicitly have the USAGE privilege on all workloads.

The SYSDEFAULTUSERWORKLOAD workload and the USAGE privilege

USAGE privilege on SYSDEFAULTUSERWORKLOAD is granted to PUBLIC at database creation time, if the database is created without the RESTRICT option. Otherwise, the USAGE privilege must be explicitly granted by a user with ACCESSCTRL, WLMADM, or SECADM authority.

If the session user does not have USAGE privilege on any of the workloads, including SYSDEFAULTUSERWORKLOAD, an SQL error is returned.

The SYSDEFAULTADMWORKLOAD workload and the USAGE privilege

USAGE privilege on SYSDEFAULTADMWORKLOAD cannot be explicitly granted to any user. Only users who issue the **SET WORKLOAD TO SYSDEFAULTADMWORKLOAD** command and whose session authorization ID has ACCESSCTRL, DATAACCESS, DBADM, WLMADM or SECADM authority are allowed to use this workload.

The GRANT USAGE ON WORKLOAD and REVOKE USAGE ON WORKLOAD statements do not have any effect on SYSDEFAULTADMWORKLOAD.

Authorization IDs in different contexts

An authorization ID is used for two purposes: identification and authorization checking. For example, the session authorization ID is used for initial authorization checking.

When referring to the use of an authorization ID in a specific context, the reference to the authorization is qualified to identify the context, as shown in the following section.

Contextual reference to authorization ID

Definition

System authorization ID

The authorization ID used to do any initial authorization checking, such as

checking for CONNECT privilege during CONNECT processing. As part of the authentication process during CONNECT processing, an authorization ID compatible with Db2 naming requirements is produced that represents the external user ID within the Db2 database system. The system authorization ID represents the user that created the connection. Use the SYSTEM_USER special register to see the current value of the system authorization ID. The system authorization ID cannot be changed for a connection.

Session authorization ID

The authorization ID used for any session authorization checking subsequent to the initial checks performed during CONNECT processing. The default value of the session authorization ID is the value of the system authorization ID. Use the SESSION_USER special register to see the current value of the session authorization ID. The USER special register is a synonym for the SESSION_USER special register. The session authorization ID can be changed by using the SET SESSION AUTHORIZATION statement.

Package authorization ID

The authorization ID used to bind a package to the database. This authorization ID is obtained from the value of the **OWNER** *authorization id* option of the **BIND** command. The package authorization ID is sometimes referred to as the package binder or package owner.

Routine owner authorization ID

The authorization ID listed in the system catalogs as the owner of the SQL routine that has been invoked.

Routine invoker authorization ID

The authorization ID that is the statement authorization ID for the statement that invoked an SQL routine.

Statement authorization ID

The authorization ID associated with a specific SQL statement that is to be used for any authorization requirements as well as for determining object ownership (where appropriate). It takes its value from the appropriate source authorization ID, depending on the type of SQL statement:

- Static SQL
The package authorization ID is used.
- Dynamic SQL (from non-routine context)
The table shows which authorization ID is used in each case:

Value of DYNAMICRULES option for issuing the package	Authorization ID used
RUN	Session authorization ID
BIND	Package authorization ID
DEFINERUN, INVOKERUN	Session authorization ID
DEFINEBIND, INVOKEBIND	Package authorization ID

- Dynamic SQL (from routine context)
The table shows which authorization ID is used in each case:

Value of DYNAMICRULES option for issuing the package	Authorization ID used
DEFINERUN, DEFINEBIND	Routine owner authorization ID

Value of DYNAMICRULES option for issuing the package	Authorization ID used
INVOKERUN, INVOKEBIND	Routine invoker authorization ID

Use the CURRENT_USER special register to see the current value of the statement authorization ID. The statement authorization ID cannot be changed directly; it is changed automatically by the Db2 database system to reflect the nature of each SQL statement.

Default privileges granted on creating a database

When you create a database, default database level authorities and default object level privileges are granted to you within that database.

The authorities and privileges that you are granted are listed according to the system catalog views where they are recorded:

1. SYSCAT.DBAUTH

- The database creator is granted the following authorities:
 - ACCESSCTRL
 - DATAACCESS
 - DBADM
 - SECADM
- In a non-restrictive database, the special group PUBLIC is granted the following authorities:
 - CREATETAB
 - BINDADD
 - CONNECT
 - IMPLICIT_SCHEMA

2. SYSCAT.TABAUTH

In a non-restrictive database, the special group PUBLIC is granted the following privileges:

- SELECT on all SYSCAT and SYSIBM tables
- SELECT and UPDATE on all SYSSTAT tables
- SELECT on the following views in schema SYSIBMADM:
 - ALL_*
 - USER_*
 - ROLE_*
 - SESSION_*
 - DICTIONARY
 - TAB

3. SYSCAT.ROUTINEAUTH


In a non-restrictive database, the special group PUBLIC is granted the following privileges:

- EXECUTE with GRANT on all procedures in schema SQLJ
- EXECUTE with GRANT on all functions and procedures in schema SYSFUN
- EXECUTE with GRANT on most functions and procedures in schema SYSPROC, for a list of exceptions see “Default PUBLIC privilege for built-in routines” on page 50
- EXECUTE on all table functions in schema SYSIBM

- EXECUTE on all other procedures in schema SYSIBM
4. SYSCAT.MODULEAUTH
In a non-restrictive database, the special group PUBLIC is granted the following privileges:
 - EXECUTE on the following modules in schema SYSIBMADM:
 - DBMS_DDL
 - DBMS_JOB
 - DBMS_LOB
 - DBMS_OUTPUT
 - DBMS_SQL
 - DBMS_STANDARD
 - DBMS_UTILITY
 5. SYSCAT.PACKAGEAUTH
 - The database creator is granted the following privileges:
 - CONTROL on all packages created in the NULLID schema
 - BIND with GRANT on all packages created in the NULLID schema
 - EXECUTE with GRANT on all packages created in the NULLID schema
 - In a non-restrictive database, the special group PUBLIC is granted the following privileges:
 - BIND on all packages created in the NULLID schema
 - EXECUTE on all packages created in the NULLID schema
 6. SYSCAT.SCHEMAAUTH
In a non-restrictive database, the special group PUBLIC is granted the following privileges:
 - CREATEIN on schema SQLJ
 - CREATEIN on schema NULLID
 7. SYSCAT.TBSPACEAUTH
In a non-restrictive database, the special group PUBLIC is granted the USE privilege on table space USERSPACE1.
 8. SYSCAT.WORKLOADAUTH
In a non-restrictive database, the special group PUBLIC is granted the USAGE privilege on SYSDEFAULTUSERWORKLOAD.
 9. SYSCAT.VARIABLEAUTH
In a non-restrictive database, the special group PUBLIC is granted the READ privilege on schema global variables in the SYSIBM schema, except for the following variables:
 - SYSIBM.CLIENT_ORIGUSERID
 - SYSIBM.CLIENT_USRSECTOKEN

A non-restrictive database is a database created without the RESTRICTIVE option on the CREATE DATABASE command.

Related information:

 Best practices: A practical guide to restrictive databases

Default PUBLIC privilege for built-in routines

When a non-restrictive database is created, the special group PUBLIC is granted EXECUTE with GRANT to the majority of built-in routines.

The exceptions are listed in Table 1. All the listed routines are in the schema SYSPROC.

Table 3. Built-in routines with no default PUBLIC privilege

Routine Name	Routine Type
ADMIN_GET_INTRA_PARALLEL	Function
ADMIN_GET_MEM_USAGE	Function
ADMIN_GET_STORAGE_PATHS	Function
ADMIN_GET_TAB_COMPRESS_INFO	Function
ADMIN_GET_TAB_COMPRESS_INFO_V97	Function
ADMIN_GET_TAB_DICTIONARY_INFO	Function
ADMIN_SET_INTRA_PARALLEL	Procedure
AUDIT_ARCHIVE	Function
AUDIT_ARCHIVE	Procedure
AUDIT_DELIM_EXTRACT	Procedure
AUDIT_LIST_LOGS	Function
AUTOMAINT_GET_POLICYFILE	Procedure
AUTOMAINT_GET_POLICY	Procedure
AUTOMAINT_SET_POLICYFILE	Procedure
AUTOMAINT_SET_POLICY	Procedure
DB2_GET_CLUSTER_HOST_STATE	Function
DB2_GET_INSTANCE_INFO	Function
ENV_GET_DB2_EDU_SYSTEM_RESOURCES	Function
ENV_GET_DB2_SYSTEM_RESOURCES	Function
ENV_GET_NETWORK_RESOURCES	Function
ENV_GET_REG_VARIABLES	Function
ENV_GET_SYS_RESOURCES	Function
ENV_GET_SYSTEM_RESOURCES	Function
EVMON_UPGRADE_TABLES	Procedure
EXPLAIN_FROM_ACTIVITY	Procedure
EXPLAIN_FROM_CATALOG	Procedure
EXPLAIN_FROM_DATA	Procedure
EXPLAIN_FROM_SECTION	Procedure
MON_CAPTURE_ACTIVITY_IN_PROGRESS	Procedure
MON_COLLECT_STATS	Procedure
MON_FORMAT_LOCK_NAME	Function
MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW	Function
MON_FORMAT_XML_METRICS_BY_ROW	Function
MON_FORMAT_XML_WAIT_TIMES_BY_ROW	Function
MON_GET_ACTIVITY_DETAILS	Function
MON_GET_ACTIVITY	Function
MON_GET_AGENT	Function
MON_GET_APPL_LOCKWAIT	Function
MON_GET_AUTO_MAINT_QUEUE	Function
MON_GET_AUTO_RUNSTATS_QUEUE	Function
MON_GET_BUFFERPOOL	Function
MON_GET_CF_CMD	Function
MON_GET_CF	Function

Table 3. Built-in routines with no default PUBLIC privilege (continued)

Routine Name	Routine Type
MON_GET_CF_WAIT_TIME	Function
MON_GET_CONNECTION_DETAILS	Function
MON_GET_CONNECTION	Function
MON_GET_CONTAINER	Function
MON_GET_DATABASE_DETAILS	Function
MON_GET_DATABASE	Function
MON_GET_EXTENDED_LATCH_WAIT	Function
MON_GET_EXTENT_MOVEMENT_STATUS	Function
MON_GET_FCM_CONNECTION_LIST	Function
MON_GET_FCM	Function
MON_GET_GROUP_BUFFERPOOL	Function
MON_GET_HADR	Function
MON_GET_INDEX	Function
MON_GET_INDEX_USAGE_LIST	Function
MON_GET_INSTANCE	Function
MON_GET_LOCKS	Function
MON_GET_MEMORY_POOL	Function
MON_GET_MEMORY_SET	Function
MON_GET_PAGE_ACCESS_INFO	Function
MON_GET_PKG_CACHE_STMT_DETAILS	Function
MON_GET_PKG_CACHE_STMT	Function
MON_GET_QUEUE_STATS	Function
MON_GET_REBALANCE_STATUS	Function
MON_GET_ROUTINE_DETAILS	Function
MON_GET_ROUTINE_EXEC_LIST	Function
MON_GET_ROUTINE	Function
MON_GET_RTS_RQST	Function
MON_GET_SECTION	Function
MON_GET_SECTION_OBJECT	Function
MON_GET_SECTION_ROUTINE	Function
MON_GET_SERVERLIST	Function
MON_GET_SERVICE_SUBCLASS_DETAILS	Function
MON_GET_SERVICE_SUBCLASS	Function
MON_GET_SERVICE_SUBCLASS_STATS	Function
MON_GET_SERVICE_SUPERCLASS_STATS	Function
MON_GET_TABLE	Function
MON_GET_TABLESPACE	Function
MON_GET_TABLESPACE QUIESCER	Function
MON_GET_TABLESPACE_RANGE	Function
MON_GET_TABLE_USAGE_LIST	Function
MON_GET_TRANSACTION_LOG	Function
MON_GET_UNIT_OF_WORK_DETAILS	Function
MON_GET_UNIT_OF_WORK	Function
MON_GET_USAGE_LIST_STATUS	Function
MON_GET_UTILITY	Function

Table 3. Built-in routines with no default PUBLIC privilege (continued)

Routine Name	Routine Type
MON_GET_WORK_ACTION_SET_STATS	Function
MON_GET_WORKLOAD_DETAILS	Function
MON_GET_WORKLOAD	Function
MON_GET_WORKLOAD_STATS	Function
MON_INCREMENT_INTERVAL_ID	Procedure
MON_SAMPLE_SERVICE_CLASS_METRICS	Function
MON_SAMPLE_WORKLOAD_METRICS	Function
SET_MAINT_MODE_RECORD_NO_TEMPORALHISTORY	Procedure
SYSTS_ALTER	Procedure
SYSTS_CLEANUP	Procedure
SYSTS_CLEAR_COMMANDLOCKS	Procedure
SYSTS_CLEAR_EVENTS	Procedure
SYSTS_CONFIGURE	Procedure
SYSTS_CREATE	Procedure
SYSTS_DISABLE	Procedure
SYSTS_DROP	Procedure
SYSTS_ENABLE	Procedure
SYSTS_UPDATE	Procedure
SYSTS_UPGRADE_CATALOG	Procedure
SYSTS_UPGRADE_INDEX	Procedure
WLM_ALTER_MEMBER_SUBSET	Procedure
WLM_CANCEL_ACTIVITY	Procedure
WLM_CAPTURE_ACTIVITY_IN_PROGRESS	Procedure
WLM_COLLECT_STATS	Procedure
WLM_CREATE_MEMBER_SUBSET	Procedure
WLM_DROP_MEMBER_SUBSET	Procedure
WLM_GET_ACTIVITY_DETAILS	Function
WLM_GET_CONN_ENV	Function
WLM_GET_QUEUE_STATS	Function
WLM_GET_SERVICE_CLASS_AGENTS	Function
WLM_GET_SERVICE_CLASS_AGENTS_V97	Function
WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES	Function
WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97	Function
WLM_GET_SERVICE_SUBCLASS_STATS	Function
WLM_GET_SERVICE_SUBCLASS_STATS_V97	Function
WLM_GET_SERVICE_SUPERCLASS_STATS	Function
WLM_GET_WORK_ACTION_SET_STATS	Function
WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES	Function
WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97	Function
WLM_GET_WORKLOAD_STATS	Function
WLM_GET_WORKLOAD_STATS_V97	Function
WLM_SET_CLIENT_INFO	Procedure
WLM_SET_CONN_ENV	Procedure

Granting and revoking access

Granting privileges

To grant privileges on most database objects, you must have ACCESSCTRL authority, SECADM authority, or CONTROL privilege on that object; or, you must hold the privilege WITH GRANT OPTION. Additionally, users with SYSADM or SYSCTRL authority can grant table space privileges. You can grant privileges only on existing objects.

About this task

To grant CONTROL privilege to someone else, you must have ACCESSCTRL or SECADM authority. To grant ACCESSCTRL, DATAACCESS, DBADM or SECADM authority, you must have SECADM authority.

The GRANT statement allows an authorized user to grant privileges. A privilege can be granted to one or more authorization names in one statement; or to PUBLIC, which makes the privileges available to all users. Note that an authorization name can be either an individual user or a group.

On operating systems where users and groups exist with the same name, you should specify whether you are granting the privilege to the user or group. Both the GRANT and REVOKE statements support the keywords USER, GROUP, and ROLE. If these optional keywords are not used, the database manager checks the operating system security facility to determine whether the authorization name identifies a user or a group; it also checks whether an authorization ID of type role with the same name exists. If the database manager cannot determine whether the authorization name refers to a user, a group, or a role, an error is returned. The following example grants SELECT privileges on the EMPLOYEE table to the user HERON:

```
GRANT SELECT
ON EMPLOYEE TO USER HERON
```

The following example grants SELECT privileges on the EMPLOYEE table to the group HERON:

```
GRANT SELECT
ON EMPLOYEE TO GROUP HERON
```

Revoking privileges

The REVOKE statement allows authorized users to revoke privileges previously granted to other users.

About this task

To revoke privileges on database objects, you must have ACCESSCTRL authority, SECADM authority, or CONTROL privilege on that object. Table space privileges can also be revoked by users with SYSADM and SYSCTRL authority. Note that holding a privilege WITH GRANT OPTION is not sufficient to revoke that privilege. To revoke CONTROL privilege from another user, you must have ACCESSCTRL, or SECADM authority. To revoke ACCESSCTRL, DATAACCESS, DBADM or SECADM authority, you must have SECADM authority. Table space privileges can be revoked only by a user who holds SYSADM, or SYSCTRL authority. Privileges can only be revoked on existing objects.

Note: A user without ACCESSCTRL authority, SECADM authority, or CONTROL privilege is not able to revoke a privilege that they granted through their use of the

WITH GRANT OPTION. Also, there is no cascade on the revoke to those who have received privileges granted by the person being revoked.

If an explicitly granted table (or view) privilege is revoked from a user with DBADM authority, privileges will not be revoked from other views defined on that table. This is because the view privileges are available through the DBADM authority and are not dependent on explicit privileges on the underlying tables.

If a privilege has been granted to a user, a group, or a role with the same name, you must specify the GROUP, USER, or ROLE keyword when revoking the privilege. The following example revokes the SELECT privilege on the EMPLOYEE table from the user HERON:

```
REVOKE SELECT
ON EMPLOYEE FROM USER HERON
```

The following example revokes the SELECT privilege on the EMPLOYEE table from the group HERON:

```
REVOKE SELECT
ON EMPLOYEE FROM GROUP HERON
```

Note that revoking a privilege from a group may not revoke it from all members of that group. If an individual name has been directly granted a privilege, it will keep it until that privilege is directly revoked.

If a table privilege is revoked from a user, privileges are also revoked on any view created by that user which depends on the revoked table privilege. However, only the privileges implicitly granted by the system are revoked. If a privilege on the view was granted directly by another user, the privilege is still held.

If a table privilege is revoked from a user, privileges are also revoked on any view created by that user which depends on the revoked table privilege. However, only the privileges implicitly granted by the system are revoked. If a privilege on the view was granted directly by another user, the privilege is still held.

You may have a situation where you want to GRANT a privilege to a group and then REVOKE the privilege from just one member of the group. There are only a couple of ways to do that without receiving the error message SQL0556N:

- You can remove the member from the group; or, create a new group with fewer members and GRANT the privilege to the new group.
- You can REVOKE the privilege from the group and then GRANT it to individual users (authorization IDs).

Note: When CONTROL privilege is revoked from a user on a table or a view, the user continues to have the ability to grant privileges to others. When given CONTROL privilege, the user also receives all other privileges WITH GRANT OPTION. Once CONTROL is revoked, all of the other privileges remain WITH GRANT OPTION until they are explicitly revoked.

All packages that are dependent on revoked privileges are marked invalid, but can be validated if rebound by a user with appropriate authority. Packages can also be rebuilt if the privileges are subsequently granted again to the binder of the application; running the application will trigger a successful implicit rebind. If privileges are revoked from PUBLIC, all packages bound by users having only been able to bind based on PUBLIC privileges are invalidated. If DBADM authority is revoked from a user, all packages bound by that user are invalidated including those associated with database utilities. Attempting to use a package that

has been marked invalid causes the system to attempt to rebound the package. If this rebound attempt fails, an error occurs (SQLCODE -727). In this case, the packages must be explicitly rebound by a user with:

- Authority to rebound the packages
- Appropriate authority for the objects used within the packages

These packages should be rebound at the time the privileges are revoked.

If you define a trigger or SQL function based on one or more privileges and you lose one or more of these privileges, the trigger or SQL function cannot be used.

Managing implicit authorizations by creating and dropping objects

The database manager implicitly grants certain privileges to a user that creates a database object such as a table or a package. Privileges are also granted when objects are created by users with DBADM authority. Similarly, privileges are removed when an object is dropped.

About this task

When the created object is a table, nickname, index, or package, the user receives CONTROL privilege on the object. When the object is a view, the CONTROL privilege for the view is granted implicitly only if the user has CONTROL privilege for all tables, views, and nicknames referenced in the view definition.

When the object explicitly created is a schema, the schema owner is given ALTERIN, CREATEIN, and DROPIN privileges WITH GRANT OPTION. An implicitly created schema has CREATEIN granted to PUBLIC.

Establishing ownership of a package

The **BIND** and **PRECOMPILE** commands create or change an application package. On either one, use the **OWNER** option to name the owner of the resulting package.

About this task

There are simple rules for naming the owner of a package:

- Any user can name themselves as the owner. This is the default if the **OWNER** option is not specified.
- A user ID with DBADM authority can name any authorization ID as the owner using the **OWNER** option.

Not all operating systems that can bind a package using Db2 database products support the **OWNER** option.

Implicit privileges through a package

Access to data within a database can be requested by application programs, as well as by persons engaged in an interactive workstation session. A package contains statements that allow users to perform a variety of actions on many database objects. Each of these actions requires one or more privileges.

Privileges granted to individuals binding the package and to PUBLIC, as well as to the roles granted to the individuals and to PUBLIC, are used for authorization checking when static SQL and XQuery statements are bound. Privileges granted through groups, and the roles granted to groups, are not used for authorization checking when static SQL and XQuery statements are bound.

Unless **VALIDATE RUN** is specified when binding the package, the user with a valid authorization ID who binds a package must either:

- Have been granted all the privileges required to execute the static SQL or XQuery statements in the package.
- Have acquired the necessary privileges through membership in one or more of:
 - **PUBLIC**
 - The roles granted to **PUBLIC**
 - The roles granted to the user

If **VALIDATE RUN** is specified at **BIND** time, all authorization failures for any static SQL or XQuery statements within this package will not cause the **BIND** to fail, and those SQL or XQuery statements are revalidated at run time. **PUBLIC**, group, role, and user privileges are all used when checking to ensure the user has the appropriate authorization (**BIND** or **BINDADD** privilege) to bind the package.

Packages may include both static and dynamic SQL and XQuery statements. To process a package with static queries, a user need only have **EXECUTE** privilege on the package. This user can then implicitly obtain the privileges of the package binder for any static queries in the package but only within the restrictions imposed by the package.

If the package includes dynamic SQL or XQuery statements, the required privileges depend on the value that was specified for **DYNAMICRULES** when the package was precompiled or bound. For more information, see the topic that describes the effect of **DYNAMICRULES** on dynamic queries.

Indirect privileges through a package containing nicknames

When a package contains references to nicknames, authorization processing for package creators and package users is slightly more complex.

When a package creator successfully binds packages that contain nicknames, the package creator does not have to pass authentication checking or privilege checking for the tables and views that the nicknames reference at the data source. However, the package executor must pass authentication and authorization checking at data sources.

For example, assume that a package creator's .SQC file contains several SQL or XQuery statements. One static statement references a local table. Another dynamic statement references a nickname. When the package is bound, the package creator's authid is used to verify privileges for the local table and the nickname, but no checking is done for the data source objects that the nickname identifies. When another user executes the package, assuming they have the **EXECUTE** privilege for that package, that user does not have to pass any additional privilege checking for the statement referencing the table. However, for the statement referencing the nickname, the user executing the package must pass authentication checking and privilege checking at the data source.

When the .SQC file contains only dynamic SQL and XQuery statements and a mixture of table and nickname references, Db2 database authorization checking for local objects and nicknames is similar. Package users must pass privilege checking for any local objects (tables, views) within the statements and also pass privilege checking for nickname objects (package users must pass authentication and privilege checking at the data source containing the objects that the nicknames identify). In both cases, users of the package must have the **EXECUTE** privilege.

The authorization ID and password of the package executor is used for all data source authentication and privilege processing. This information can be changed by creating a user mapping.

Note: Nicknames cannot be specified in static SQL and XQuery statements. Do not use the **DYNAMICRULES** option (set to BIND) with packages containing nicknames.

It is possible that packages containing nicknames might require additional authorization steps because Db2 database uses dynamic SQL when communicating with Db2 Family data sources. The authorization ID running the package at the data source must have the appropriate authority to execute the package dynamically at that data source.

Controlling access to data with views

A view provides a means of controlling access or extending privileges to a table.

Using a view allows the following kinds of control over access to a table:

- Access only to designated columns of the table.
For users and application programs that require access only to specific columns of a table, an authorized user can create a view to limit the columns addressed only to those required.
- Access only to a subset of the rows of the table.
By specifying a WHERE clause in the subquery of a view definition, an authorized user can limit the rows addressed through a view.
- Access only to a subset of the rows or columns in data source tables or views. If you are accessing data sources through nicknames, you can create local Db2 database views that reference nicknames. These views can reference nicknames from one or many data sources.

Note: Because you can create a view that contains nickname references for more than one data source, your users can access data in multiple data sources from one view. These views are called *multi-location views*. Such views are useful when joining information in columns of sensitive tables across a distributed environment or when individual users lack the privileges needed at data sources for specific objects.

To create a view, a user must have DATAACCESS authority, or CONTROL or SELECT privilege for each table, view, or nickname referenced in the view definition. The user must also be able to create an object in the schema specified for the view. That is, DBADM authority, CREATEIN privilege for an existing schema, or IMPLICIT_SCHEMA authority on the database if the schema does not already exist.

If you are creating views that reference nicknames, you do not need additional authority on the data source objects (tables and views) referenced by nicknames in the view; however, users of the view must have SELECT authority or the equivalent authorization level for the underlying data source objects when they access the view.

If your users do not have the proper authority at the data source for underlying objects (tables and views), you can:

1. Create a data source view over those columns in the data source table that are OK for the user to access
2. Grant the SELECT privilege on this view to users

3. Create a nickname to reference the view

Users can then access the columns by issuing a SELECT statement that references the new nickname.

The following scenario provides a more detailed example of how views can be used to restrict access to information.

Many people might require access to information in the STAFF table, for different reasons. For example:

- The personnel department needs to be able to update and look at the entire table.

This requirement can be easily met by granting SELECT and UPDATE privileges on the STAFF table to the group PERSONNL:

```
GRANT SELECT,UPDATE ON TABLE STAFF TO GROUP PERSONNL
```

- Individual department managers need to look at the salary information for their employees.

This requirement can be met by creating a view for each department manager. For example, the following view can be created for the manager of department number 51:

```
CREATE VIEW EMP051 AS
  SELECT NAME,SALARY,JOB FROM STAFF
  WHERE DEPT=51
GRANT SELECT ON TABLE EMP051 TO JANE
```

The manager with the authorization name JANE would query the EMP051 view just like the STAFF table. When accessing the EMP051 view of the STAFF table, this manager views the following information:

NAME	SALARY	JOB
Fraye	45150.0	Mgr
Williams	37156.5	Sales
Smith	35654.5	Sales
Lundquist	26369.8	Clerk
Wheeler	22460.0	Clerk

- All users need to be able to locate other employees. This requirement can be met by creating a view on the NAME column of the STAFF table and the LOCATION column of the ORG table, and by joining the two tables on their corresponding DEPT and DEPTNUMB columns:

```
CREATE VIEW EMPLOCS AS
  SELECT NAME, LOCATION FROM STAFF, ORG
  WHERE STAFF.DEPT=ORG.DEPTNUMB
GRANT SELECT ON TABLE EMPLOCS TO PUBLIC
```

Users who access the employee location view will see the following information:

NAME	LOCATION
Molinare	New York
Lu	New York
Daniels	New York
Jones	New York
Hanes	Boston
Rothman	Boston

NAME	LOCATION
Ngan	Boston
Kermisch	Boston
Sanders	Washington
Pernal	Washington
James	Washington
Sneider	Washington
Marenghi	Atlanta
O'Brien	Atlanta
Quigley	Atlanta
Naughton	Atlanta
Abrahams	Atlanta
Koonitz	Chicago
Plotz	Chicago
Yamaguchi	Chicago
Scoutten	Chicago
Fraye	Dallas
Williams	Dallas
Smith	Dallas
Lundquist	Dallas
Wheeler	Dallas
Lea	San Francisco
Wilson	San Francisco
Graham	San Francisco
Gonzales	San Francisco
Burke	San Francisco
Quill	Denver
Davis	Denver
Edwards	Denver
Gafney	Denver

Controlling access for database administrators (DBAs)

You may want to monitor, control, or prevent access to data by database administrators (users holding DBADM authority).

Monitoring access to data

You can use the Db2 audit facility to monitor access by database administrators. To do so, follow these steps:

1. Create an audit policy that monitors the events you want to capture for users who hold DBADM authority.
2. Associate this audit policy with the DBADM authority.

Controlling access to data

You can use trusted contexts in conjunction with a role to control access by database administrators. To do so, follow these steps:

1. Create a role and grant DBADM authority to that role.
2. Define a trusted context and make the role the default role for this trusted context.
Do not grant membership in the role to any authorization ID explicitly. This way, the role is available only through this trusted context and a user acquires DBADM capability only when they are within the confines of the trusted context.
3. There are two ways you can control how users access the trusted context:
 - Implicit access: Create a unique trusted context for each user. When the user establishes a regular connection that matches the attributes of the trusted context, they are implicitly trusted and gain access to the role.
 - Explicit access: Create a trusted context using the WITH USE FOR clause to define all users who can access it. Create an application through which those users can make database requests. The application establishes an explicit trusted connection, and when a user issues a request, the application switches to that user ID and executes the request as that user on the database.

If you want to monitor the use of this trusted context, you can create an audit policy that captures the events you are interested in for users of this trusted context. Associate this audit policy with the trusted context.

Preventing access to data

To prevent access to data in tables, choose one of these options:

- To prevent access to data in all tables, revoke DATAACCESS from your DBADM user, role or group. Alternatively, you could grant DBADM to the user, role or group of interest without the DATAACCESS option
- To prevent access to data in one particular table, follow these steps:
 - Assign a security label to every column in the table.
 - Grant that security label to a role.
 - Grant that role to all users (or roles) that have a legitimate need to access the table.

No user, regardless of their authority, will be able to access data in that table unless they are a member in that role.

Gaining access to data through indirect means

To successfully manage security, you need to be aware of indirect ways that users can gain access to data.

The following list represents the indirect means through which users can gain access to data they might not be authorized to access:

- **Catalog views:** The Db2 database system catalog views store metadata and statistics about database objects. Users with SELECT access to the catalog views can gain some knowledge about data that they might not be qualified for. For better security, make sure that only qualified users have access to the catalog views.

Note: In Db2 Universal Database Version 8, or earlier, SELECT access on the catalog views was granted to PUBLIC by default. In Db2 Version 9.1, or later, database systems, users can choose whether SELECT access to the catalog views is granted to PUBLIC or not by using the new **RESTRICTIVE** option on the **CREATE DATABASE** command.

- **Explain snapshot:** The explain snapshot is compressed information that is collected when an SQL or XQuery statement is explained. It is stored as a binary large object (BLOB) in the EXPLAIN_STATEMENT table, and contains column statistics that can reveal information about table data. For better security, access to the explain tables should be granted to qualified users only.
- **Section explain:** The section explain procedures (EXPLAIN_FROM_SECTION, EXPLAIN_FROM_CATALOG, EXPLAIN_FROM_ACTIVITY and EXPLAIN_FROM_DATA) can populate explain tables with information from any section that resides in the package cache. This information includes statement text which may contain input data values. For better security, access to the section explain procedures and explain tables should be granted to qualified users only.
- **Log reader functions:** A user authorized to run a function that reads the logs can gain access to data they might not be authorized for if they are able to understand the format of a log record. These functions read the logs:

Function	Authority needed in order to execute the function
db2ReadLog	SYSADM or DBADM
db2ReadLogNoConn	None.

- **Replication:** When you replicate data, even the protected data is reproduced at the target location. For better security, make sure that the target location is at least as secure as the source location.
- **Exception tables:** When you specify an exception table while loading data into a table, users with access to the exception table can gain information that they might not be authorized for. For better security, only grant access to the exception table to authorized users and drop the exception table as soon as you are done with it.
- **Backup table space or database:** Users with the authority to run the **BACKUP DATABASE** command can take a backup of a database or a table space, including any protected data, and restore the data somewhere else. The backup can include data that the user might not otherwise have access to.
The **BACKUP DATABASE** command can be executed by users with SYSADM, SYSCTRL, or SYSMAINT authority.
- **Set session authorization:** In Db2 Universal Database Version 8, or earlier, a user with DBADM authority could use the SET SESSION AUTHORIZATION SQL statement to set the session authorization ID to any database user. In Db2 Version 9.1, or later, database systems a user must be explicitly authorized through the GRANT SETSESSIONUSER statement before they can set the session authorization ID.

When upgrading an existing Version 8 database to a Db2 Version 9.1, or later, database system, however, a user with existing explicit DBADM authority (for example, granted in SYSCAT.DBAUTH) will keep the ability to set the session authorization to any database user. This is allowed so that existing applications will continue to work. Being able to set the session authorization potentially allows access to all protected data. For more restrictive security, you can override this setting by executing the REVOKE SETSESSIONUSER SQL statement.

- **Lock monitoring:** As part of the lock monitoring activity of Db2 database management systems, values associated with parameter markers are written to the monitoring output when the HIST_AND_VALUES collection level is specified. Values may also be embedded in the statement text captured by the lock event monitor. A user with access to the monitoring output can gain access to information for which they might not be authorized.
- **Activity monitoring:** As part of monitoring activities in a Db2 database management system using an activity event monitor, the values associated with parameter markers are written to the monitoring output when the VALUES clause is specified, and the statement text (which may contain input data values) is written to the monitoring output when the WITH DETAILS clause is specified. A user with access to the monitoring output can gain access to information for which they might not be authorized. For better security, access to the CREATE EVENT MONITOR statement and any event monitor tables should be granted to qualified users only.
- **Package cache monitoring:** As part of monitoring the package cache in a Db2 database management system using a package cache event monitor, the statement text (which may contain input data values) is written to the monitoring output whenever a section is ejected from the package cache. For better security, access to the CREATE EVENT MONITOR statement and any event monitor tables should be granted to qualified users only.
- **Monitor table functions, views and reports:** The following monitor table functions, views and reports expose statement text for either currently executing statements or statements in the package cache:
 - SYSPROC.MON_GET_ACTIVITY_DETAILS
 - SYSPROC.MON_GET_PKG_CACHE_STMT
 - SYSPROC.MON_GET_PKG_CACHE_STMT_DETAILS
 - SYSIBMADM.MON_PKG_CACHE_SUMMARY
 - SYSIBMADM.MON_CURRENT_SQL
 - SYSIBMADM.MON_LOCKWAITS
 - SYSIBMADM.MONREPORT.LOCKWAIT
 - SYSIBMADM.MONREPORT.CURRENTSQL
 - SYSIBMADM.MONREPORT.PKGCACHE

The statement text may contain input data values. For better security, EXECUTE privilege on these table functions and reports and SELECT privilege on these views should be granted to qualified users only.

- **Traces:** A trace can contain table data. A user with access to such a trace can gain access to information that they might not be authorized for.
- **Dump files:** To help in debugging certain problems, Db2 database products might generate memory dump files in the sql1ib\db2dump directory. These memory dump files might contain table data. If they do, users with access to the files can gain access to information that they might not be authorized for. For better security you should limit access to the sql1ib\db2dump directory.
- **db2dart:** The **db2dart** tool examines a database and reports any architectural errors that it finds. The tool can access table data and Db2 does not enforce access control for that access. A user with the authority to run the **db2dart** tool or with access to the **db2dart** output can gain access to information that they might not be authorized for.
- **REOPT bind option:** When the **REOPT** bind option is specified, explain snapshot information for each reoptimizable incremental bind SQL statement is placed in the explain tables at run time. The explain will also show input data values.

- **db2cat:** The **db2cat** tool is used to dump a table's packed descriptor. The table's packed descriptor contains statistics that can reveal information about a table's contents. A user who runs the **db2cat** tool or has access to the output can gain access to information that they might not be authorized for.

Data encryption

The Db2 database system offers several ways to encrypt data, both while in storage, and while in transit over the network.

Encrypting data in storage

You have the following options for encrypting data in storage:

- You can use Db2 native encryption to encrypt your databases and backup images.
- You can use IBM InfoSphere® Guardium® Data Encryption to encrypt the underlying operating system data and backup files.
- If you are running a Db2 system on the AIX operating system, and you are interested in file-level encryption only, you can use encrypted file system (EFS) to encrypt your operating system data and backup files.

Encrypting data in transit

To encrypt data in-transit between clients and Db2 databases, you can use the `DATA_ENCRYPT` authentication type, or, the Db2 database system support of Secure Sockets Layer (SSL).

Note: `DATA_ENCRYPT` and `SERVER_ENCRYPT` with DES use algorithms that are not compliant with NIST SP 800-131A. If you must comply with NIST SP 800-131A, they must not be used. If compliance to NIST SP 800-131A is not an issue, they are still valid.

Configuring Secure Sockets Layer (SSL) support in a Db2 instance

The Db2 database system supports SSL, which means that a Db2 client application that also supports SSL can connect to a Db2 database by using a SSL socket. CLI, CLP, and .Net Data Provider client applications and applications that use the IBM Data Server Driver for JDBC and SQLJ (type 4 connections) support SSL.

Before you begin

This procedure helps you to configure client applications to communicate with Db2 using SSL. In addition, starting in Db2 Version 11.1, databases configured with High Availability Disaster Recovery (HADR) can also be configured to transmit transaction log data between primary and standby database servers using SSL. For more information, see *Configuring SSL for the communication between primary and standby HADR servers*.

Before you configure SSL support, perform the following steps:

- Ensure that the path to the IBM Global Security Kit (GSKit) libraries appears in the **PATH** environment variable on Windows platforms and the **LIBPATH**, **SHLIB_PATH** or **LD_LIBRARY_PATH** environment variables on Linux and UNIX platforms. GSKit is automatically included when you install the Db2 database system.

On Windows 32-bit platforms, the GSKit libraries are located in C:\Program Files\IBM\GSK8\lib. In this case, the system **PATH** must include C:\Program Files\IBM\GSK8\lib. On Windows 64-bit platforms, the 64-bit GSKit libraries are located in C:\Program Files\IBM\GSK8\lib64 and the 32-bit GSKit libraries are located in C:\Program Files (x86)\IBM\GSK8\lib.

On UNIX and Linux platforms, the GSKit libraries are located in sqllib/lib/gskit.

On non-Windows platforms, the Db2 database manager installs GSKit locally, and for a given instance, the GSKit libraries would be located in sqllib/lib/gskit or sqllib/lib64/gskit. It is unnecessary to have another copy of GSKit installed in a global location to bring up the instance. If a global copy of GSKit does exist, keep the version of the global GSKit at the same version of the local GSKit.

- Ensure that the connection concentrator is not activated. SSL support will not be enabled in the Db2 instance if connection concentrator is running.

To determine whether connection concentrator is activated, issue the **GET DATABASE MANAGER CONFIGURATION** command. If the configuration parameter **max_connections** is set to a value greater than the value of **max_coordagents**, connection concentrator is activated.

About this task

The SSL communication will always be in FIPS mode.

SSL support for Db2 Connect

If you are using Db2 Connect for System i, Db2 Connect for System z, or Db2 Enterprise Server Edition on an intermediate server computer to connect Db2 clients to a host or System i database, SSL support is available in any of the following configurations:

- Between the client and the Db2 Connect server
- Between the Db2 Connect server and the server
- Between both the client and the Db2 Connect server and the Db2 Connect server and the server

Note: For SSL support to be enabled on all paths in the configuration, each client or server must fulfill all requirements for SSL support. For example, if the Db2 Connect connection concentrator is on, the inbound request to the Db2 Connect server cannot use SSL. However, the outbound request to the target server can use SSL.

SSL support for databases that are configured with High Availability Disaster Recovery (HADR)

SSL is supported between clients and the HADR primary or standby database servers, including standby databases enabled for Reads on Standby. Clients connecting to the HADR primary database by using SSL are able to reroute to the HADR standby database server configured with SSL. In addition, starting in Version 11.1, SSL is also supported for transaction log data transmission between primary and standby database servers. For more information, see *Configuring SSL for the communication between primary and standby HADR servers*.

Documentation for the GSKit tool: GSKCapiCmd

For information about the GSKit tool GSKCapiCmd, see the *GSKCapiCmd User's Guide*, available at ftp://ftp.software.ibm.com/software/webserver/appserv/library/v80/GSK_CapiCmd_UserGuide.pdf.

Configuring SSL support

To configure SSL support, first, you create a key database to manage your digital certificates. These certificates and encryption keys are used for establishing the SSL connections. Second, the Db2 instance owner must configure the Db2 instance for SSL support.

Procedure

1. Create a key database and set up your digital certificates.

- a. Use the GSKCapiCmd tool to create your key database. It must be a Certificate Management System (CMS) type key database. The GSKCapiCmd is a non-Java-based command-line tool, and Java does not need to be installed on your system to use this tool.

You invoke GSKCapiCmd using the **gskcapiCmd** command, as described in the *GSKCapiCmd User's Guide*. The path for the command is `sqlib/gskit/bin` on Linux and UNIX platforms, and `C:\Program Files\IBM\GSK8\bin` on both 32-bit and 64-bit Windows platforms. (On 64-bit platforms, the 32-bit GSKit executable files and libraries are also present; in this case, the path for the command is `C:\Program Files (x86)\IBM\GSK8\bin`.) Ensure `PATH` (on the Windows platform) includes the proper GSKit library path, and `LIBPATH`, `SHLIB_PATH`, or `LD_LIBRARY_PATH` (on UNIX or Linux platforms) include the proper GSKit library path, such as `sqlib/lib64/gskit`.

For example, the following command creates a key database called `mydbserver.kdb` and a stash file called `mydbserver.sth`:

```
gsk8capiCmd_64 -keydb -create -db "mydbserver.kdb" -pw "myServerPassw0rdpw0"
               -stash
```

The **-stash** option creates a stash file at the same path as the key database, with a file extension of `.sth`. At instance start-up, GSKit uses the stash file to obtain the password to the key database.

Note: You should use strong file system protection on the stash file. By default, only the instance owner has access to this file (both read and write access).

When you create a key database, it is automatically populated with signer certificates from a few certificate authorities (CAs), such as Verisign.

- b. Add a certificate for your server to your key database. The server sends this certificate to clients during the SSL handshake to provide authentication for the server. To obtain a certificate, you can either use GSKCapiCmd to create a new certificate request and submit it to a CA to be signed, or you can create a self-signed certificate for testing purposes.

For example, to create a self-signed certificate with a label of `myselfsigned`, use the **GSKCapiCmd** command as shown in the following example:

```
gsk8capiCmd_64 -cert -create -db "mydbserver.kdb" -pw "myServerPassw0rdpw0"
               -label "myselfsigned" -dn "CN=myhost.mycompany.com,O=myOrganization,
               OU=myOrganizationUnit,L=myLocation,ST=ON,C=CA" -size 2048 -sigalg SHA256_WITH_RSA
```

Where:

- `-cert` is the certificate request command which creates a new RSA private-public key pair and a PKCS10 certificate request in the specified key database
- `-db` is the fully qualified path name of a key database

- `-pw` is the password for the key database identified by the `-db` or `-tokenlabel` tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password
 - `-label` is the label to be attached to the certificate request on creation. The user uses this label to uniquely identify the certificate request
 - `-dn <dist_name>` is the X.500 distinguished name that will uniquely identify the certificate. The input must be a quoted string of the following format (only **CN** is required):
 - CN = common name
 - O = organization
 - OU = organization unit
 - L = location
 - ST = State, Province
 - C = country
 - DC = domain component
 - EMAIL =email address
 - `-size <key_size>` is the size, in bits, of the new key pair to be created.
 - `-sigalg | -sig_alg <signature_algorithm>` is the signing algorithm to be used during the creation of the self-signed certificate. This algorithm is used to create the signature associated with the new self-signed certificate. The generated key type will be chosen to match this signing algorithm
- c. Extract the certificate you just created to a file, so that you can distribute it to computers running clients that will be establishing SSL connections to your Db2 server.
- For example, the following GSKCapiCmd command extracts the certificate to a file called `mydbserver.arm`:
- ```
gsk8capiCmd_64 -cert -extract -db "mydbserver.kdb" -pw "myServerPassw0rdpw0"
 -label "myselfsigned" -target "mydbserver.arm" -format ascii -fips
```
2. To set up your Db2 server for SSL support, log in as the Db2 instance owner and set the following configuration parameters and the **DB2COMM** registry variable.
- a. Set the **ssl\_svr\_keydb** configuration parameter to the fully qualified path of the key database file. For example:
- ```
db2 update dbm cfg using SSL_SVR_KEYDB
      /home/test/sql1ib/security/keystore/key.kdb
```
- If **ssl_svr_keydb** is null (unset), SSL support is not enabled.
- b. Set the **ssl_svr_stash** configuration parameter to the fully qualified path of the stash file. For example:
- ```
db2 update dbm cfg using SSL_SVR_STASH
 /home/test/sql1ib/security/keystore/mydbserver.sth
```
- If **ssl\_svr\_stash** is null (unset), SSL support is not enabled.
- c. Set the **ssl\_svr\_label** configuration parameter to the label of the digital certificate of the server, which you added in Step 1. If **ssl\_svr\_label** is not set, the default certificate in the key database is used. If there is no default certificate in the key database, SSL is not enabled. For example: `db2 update dbm cfg using SSL_SVR_LABEL myselfsigned` where *myselfsigned* is a sample label.

- d. Set the **ssl\_svcname** configuration parameter to the port that the Db2 database system should listen on for SSL connections. If TCP/IP and SSL are both enabled (the **DB2COMM** registry variable is set to 'TCPIP, SSL'), you must set **ssl\_svcname** to a different port than the port to which **svcname** is set. The **svcname** configuration parameter sets the port that the Db2 database system listens on for TCP/IP connections. If you set **ssl\_svcname** to the same port as **svcname**, neither TCP/IP or SSL will be enabled. If **ssl\_svcname** is null (unset), SSL support is not enabled.

**Note:** In HADR environments, do not set **hadr\_local\_svc** on the primary or standby database system to the same value as you set for **ssl\_svcname**. Also, do not set **hadr\_local\_svc** to the same value as **svcname**, or **svcname** plus one.

**Note:** When the **DB2COMM** registry variable is set to 'TCPIP,SSL', if TCPIP support is not properly enabled, for example due to the **svcname** configuration parameter being set to null, the error SQL5043N is returned and SSL support is not enabled.

- e. (Optional) If you want to specify which cipher suites the server can use, set the **ssl\_cipherspecs** configuration parameter. If you leave **ssl\_cipherspecs** as null (unset), this allows GSKit to pick the strongest available cipher suite that is supported by both the client and the server. See “Supported cipher suites” on page 87 for information about which cipher suites are available.
- f. Add the value SSL to the **DB2COMM** registry variable. For example:

```
db2set -i db2inst1 DB2COMM=SSL
```

where *db2inst1* is the Db2 instance name. The database manager can support multiple protocols at the same time. For example, to enable both TCP/IP and SSL communication protocols:

```
db2set -i db2inst1 DB2COMM=SSL,TCPIP
```

- g. Restart the Db2 instance. For example:

```
db2stop
db2start
```

## Example

The following example demonstrates how to display a certificate. This example uses the self-signed certificate created by the following command:

```
gsk8capicmd_64 -cert -create -db "mydbserver.kdb" -pw "mydbserverpw0"
-label "myselfsigned" -dn "CN=myhost.mycompany.com,O=myOrganization,
OU=myOrganizationUnit,L=myLocation,ST=ON,C=CA" -size 2048 -sigalg SHA256_WITH_RSA
```

To display the certificate, issue the following command:

```
gsk8capicmd_64 -cert -details -db "mydbserver.kdb" -pw "mydbserverpw0"
-label "myselfsigned"
```

The output is displayed as follows:

```
label : myselfsigned
key size : 2048
version : X509 V3
serial : 45095be96766f560
issue:CN=myhost.mycompany.com,OU=myOrganizationUnit,O=myOrganization,L=myLocation,ST=ON,C=CA
subject:CN=myhost.mycompany.com,OU=myOrganizationUnit,O=myOrganization,L=myLocation,ST=ON,C=CA
not before : June 13, 2016 2:01:19 PM EDT
not after : June 14, 2017 2:01:19 PM EDT
public Key
30 82 01 22 30 0D 06 09 2A 86 48 86 F7 0D 01 01
```

```

01 05 00 03 82 01 0F 00 30 82 01 0A 02 82 01 01
00 A2 E0 48 05 A2 B3 37 82 43 08 68 7D 9C 87 73
34 81 5A 3B AC 5E 34 69 39 17 CA DB 08 E7 F6 26
94 63 67 F0 9D 73 7B 49 68 23 FB A6 E1 12 B8 13
3E 30 56 62 40 8B 73 EC CC 01 BA B7 22 D7 5C BF
7E EC 9B FC D8 81 34 DD A8 96 19 4A C4 CC 95 9C
3E 02 3D 1C D8 18 AA 88 99 15 24 5B 4A BB 8E 05
40 B8 CF 06 EF 6D 5B 6D 23 D1 72 BE 3F 3B 7F 55
98 41 FB C3 71 8A 31 89 0F 35 C5 74 91 3D 80 36
09 6D 9A 48 86 8F 20 06 89 7E 15 DE A6 65 C7 AA
C5 91 3C 89 9F 41 D2 7D 26 AA D7 78 E7 50 7F 95
D2 E1 ED A9 C8 02 5C 3D 4D 8C 21 9C 09 1F 2D 10
57 A4 D8 E6 DC FF 52 B9 1A D7 D4 C2 B3 E5 38 1C
89 75 9F 2E 68 98 88 C6 40 81 12 75 AA DF 7B F2
8B 44 2F AE C7 8A A8 87 66 4E FA CE D5 F9 C8 AD
A8 21 82 4D A0 83 33 E2 AA 23 A3 9B CB FB 38 A0
22 43 9B C0 AC 74 E5 57 99 E6 C1 03 96 93 AD C0
AF 02 03 01 00 01
public key type : RSA (1.2.840.113549.1.1.1)
finger print : SHA1 :
 63 51 15 86 4D 9C 6F 34 4D B0 1D 35 D2 D9 B1 74
 06 5E F8 7B
Fingerprint : MD5 :
 64 5C 62 42 D1 3E AC 02 C6 73 BF FC F6 D5 D5 E8
Fingerprint : SHA256 :
 22 A7 E8 93 2D D6 75 92 69 D6 FF 41 A4 27 92 8D
 20 FB F2 88 F4 3C B5 7C 2A 82 47 D4 37 36 98 4D
Extensions
 SubjectKeyIdentifier
 keyIdentifier:
 15 5C F0 3E E8 35 CF 14 C8 31 1F C6 BB D1 5D 68
 0D 38 19 B4
 AuthorityKeyIdentifier
 keyIdentifier:
 15 5C F0 3E E8 35 CF 14 C8 31 1F C6 BB D1 5D 68
 0D 38 19 B4
 authorityIdentifier:
 authorityCertSerialNumber:
Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)
value
 7D 76 1D 2E F6 49 CD A0 46 84 37 88 96 FF A8 7A
 06 39 E8 C9 87 D4 D8 5A 6E E4 45 6C BC 12 DA 7E
 C4 1F E3 87 88 34 62 C1 BE 0A D3 AF 59 96 2D 1E
 EF FB 4C 0A D0 73 90 67 8B CE FC F3 F0 6B F8 38
 52 E2 A8 41 45 D0 A4 98 A3 E5 36 47 E1 C7 0F B0
 D7 DF BD 9E D8 90 C0 84 CE B5 22 75 FC 5A D5 8E
 D7 C5 EB 65 52 4B CD 69 B0 B7 71 8F 18 35 81 D9
 F4 7F 0C EF 81 70 BA 5B EE A1 55 DD 3C 82 52 4D
 E7 4C 38 B3 5E 7E FE CC CE 9A 5E 77 8B 8C 6A 56
 E3 2A 41 C7 5D AE F5 6E B5 FA 21 46 EE 3F 8A A6
 20 C4 A7 28 2E EB B8 8B 90 31 D9 F8 C5 A8 A6 6C
 BF 91 29 1D 44 A5 5D F4 39 A3 3C D4 31 5E 65 48
 F6 73 A8 45 16 75 EA 84 4C A6 30 DE B5 0C 08 44
 03 1F 53 C2 A5 FD AD 7B A0 7E 96 61 8B 6B C5 93
 27 0A EB AA 37 8A 83 AC 14 33 65 55 C0 94 21 39
 F7 61 F1 6F 12 EF A3 AE 26 57 57 CD B0 67 4B A7
Trust Status : Enabled

```

To obtain a CA-signed certificate for your server (instead of a self-signed certificate), you need to generate a certificate signing request and pay the well known CA, such as VeriSign, to sign the certificate. After you get the signed certificate back, you need to receive it into the server key database. The following example demonstrates how to request and receive a certificate. It uses a trial version of a certificate.

1. First, create a Certificate Signing Request (CSR) for mydbserver.kdb The following command creates a new RSA private-public key pair and a PKCS10 certificate request in the specified key database. For CMS key databases, the certificate request information is stored in the file with the ".rdb" extension. The file specified by the **-file** option is the one that needs to be sent to a CA.

```
gsk8capicmd_64 -certreq -create -db "mydbserver.kdb" -pw "mydbserverpw0"
-label "mycert" -dn "CN=myhost.mycompany.com,
O=myOrganization,OU=myOrganizationUnit,L=myLocation,ST=ON,C=CA",
-file "mycertRequestNew" -size 2048 -sigalg SHA256_WITH_RSA
```

The following command lists the detailed information of the certificate request for my db server:

```
gsk8capicmd_64 -certreq -details -showOID -db "mydbserver.kdb"
-pw "mydbserverpw0" -label "mycert"
```

The output would display as follows:

```
label : mycert
key size : 2048
subject : Common Name (CN):
Type : 2.5.4.3
Value: myhost.mycompany.com
Organizational Unit (OU):
Type : 2.5.4.11
Value: myOrganizationUnit
OrganizationName (O) :
Type : 2.5.4.10
Value: myOrganization
Locality (L):
Type : 2.5.4.7
Value: myLocation
StateOrProvinceName (ST) :
Type : 2.5.4.8
Value: ON
CountryName (C) :
Type : 2.5.4.6
Value: CA
public Key
30 82 01 22 30 0D 06 09 2A 86 48 86 F7 0D 01 01
01 05 00 03 82 01 0F 00 30 82 01 0A 02 82 01 01
00 CE F0 92 0E 7C 78 AD C9 26 7A C8 82 CE DA A3
CC 88 35 37 9B 27 3F 49 E8 09 90 ED 25 AD 56 AF
E0 BB 3A 13 E3 EE ED 28 32 0A D8 82 BC F5 7F F7
54 2A DC EE 32 1C BB 84 53 0F D5 3B A6 58 E4 9E
F0 5E 0A 65 22 83 81 63 4C 3D 40 D4 52 7D 2D 8B
9A B9 35 4A 4F BE 9D 33 B9 01 54 1B 8F 22 62 C4
D0 41 AB A9 14 D8 BF 7A 1B 74 6D 03 82 EE DD E8
0A 16 F1 1F 94 90 C6 36 D7 3A 81 63 1D 8F 45 84
04 D1 D3 D5 7F FB 25 91 FB D3 7F CC 86 16 5A BC
B8 7D DC C4 FA 08 C2 0A B1 FF 58 F2 15 73 F7 D5
CD 40 63 3C 37 A2 B3 9D 27 A5 D2 B1 48 FE 51 22
43 61 6A FA 1B D2 CF 22 67 71 E0 CD 74 C5 29 7E
30 46 AE E7 FB B1 6C CD E0 A3 0D 10 EA A3 EA 64
AE D0 64 49 26 DC 58 A4 C8 28 D2 76 94 3F 14 3D
D6 92 A5 B3 8D 37 AE 92 1D 00 CF 10 3F 28 E9 11
C3 55 7C C4 16 92 3E D4 F2 2D 37 CC 4B 3A DD 84
3F 02 03 01 00 01
Public Key Type : RSA (1.2.840.113549.1.1.1)
Finger print :
eaad5783226fbfee46a5ef11a4605a76
04cc89b9
Attributes
Signature Algorithm : SHA1WithRSAEncryption (1.2.840.113549.1.1.5)
Value
77 91 EB 56 61 BD 2E DB 83 45 68 D9 39 44 DD E2
```

```

FA 67 C5 90 AE 9E 82 8D 4E C1 E6 BF 33 AE CA 51
33 AA 9D 91 92 00 9C 0C 07 FD FC 4D DC 4C 03 20
48 76 25 1B 2C 0F A1 FA 8A F6 4C 22 5B D8 00 73
A8 2D 23 ED 7C 9E EE 1B E7 79 39 B7 7A 6A B7 E4
3A 82 BB 5B 7B 0F BA 5B 58 89 AE 3C F7 19 EE 43
40 77 F0 57 B4 B4 0A 36 1D C9 E9 18 26 23 83 DF
C8 1B 3F 37 E7 E9 01 76 8A C7 8A 31 41 FF 39 10
08 CC 02 92 F4 0C 9A E2 17 C3 8B 02 BD 46 02 22
74 34 90 9E 21 7D 64 67 5B 97 F4 6C EB B3 B2 A0
AD 99 C7 7C 99 67 49 E0 C5 66 61 A3 C4 3D 61 D5
34 C7 1F 40 58 98 61 5A 4D 76 6C 60 6B C1 40 D0
34 18 26 68 CE BD C4 F1 8D 4E 7E D4 79 45 0D 46
5E 6D 27 1D 07 50 83 FD 8E 96 64 EB A3 B5 A2 B3
8C AD 57 84 B2 F6 BE E7 55 4B B6 38 F8 2E 5D E1
96 2C 96 2D AE 14 09 A1 9F 7E 43 7A 53 EF 4F 4E

```

To display the certificate request file:

```
$ cat mycertRequestNew
```

```

-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBrjCCARcCAQAwbjELMAkGA1UEBhMCQ0ExEDAOBgNVBAGTB09udGFyaW8xEDAO
BgNVBACTB01hcmt0YW0xDDAKBgNVBAoTA01CTEMMAoGA1UECxmDREIyMR8wHQYD
VQQDEExZnaWxlcmludG9yb2xhYi5pYm0uY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GN
ADCBiQKBgQCctGI8iQJ0sNjqC7jMcGNKWR8P/ZiaGjmU40PB3rNIUdX2YZvEbiR
CKzj4iEy/kmFB8n1QGs+TVY1BWLWeAvjlyj3JzGkbb7y0kRr2NH/HttpZY+ZJUj1F
nB60zNqh2Q861gImXIkjLu4xZY2Hjr1hxm8pdvrAxbmM4UUAzLd8QIDAQABAAAw
DQYJKoZIhvcNAQEFBQADgYEATwa04x8AtIGQzKKZSgJo0IS1fzML8ATVfUxyc1zT
N3fibRAXUBnQf2HHyFR7281vR59+f1rMZCCF1ahex3379Ip/S3RvCsbvCecKFRfM
HdJd7QKhvh388mXrDeKtVihMTHN2Fp8bEjt6Ac/gY5fo0AL7R+7yF1RmTfd/nhPa
dqI=
-----END NEW CERTIFICATE REQUEST-----

```

In case you need to delete the certificate request, use a command similar to the following example:

```
gsk8capicmd_64 -certreq -delete -db "mydbserver.kdb" -pw "mydbserverpw0"
-label "mycert"
```

2. Next, go to the VeriSign website, register and you will be asked to cut and paste the request file to submit the request. For trial version, you would receive an email that contains the signed certificate. The email also contains links for downloading the trial root CA certificate and the trial intermediate CA certificate. Use notepad or vi to save all three certificates into files:

- RootCert.arm
- IntermediateCert.arm
- MyCertificate.arm

These three are in a chain of trust.

Add the trial Root CA Certificate into mydbserver.kdb with the following command:

```
gsk8capicmd_64 -cert -add -db "mydbserver.kdb" -pw "mydbserverpw0"
-label "trialRootCACert" -file RootCert.arm -format ascii
```

Add the trial Intermediate CA Certificate into mydbserver.kdb with the following command:

```
gsk8capicmd_64 -cert -add -db "mydbserver.kdb" -pw "mydbserverpw0"
-label "trialIntermediateCACert" -file IntermediateCert.arm -format ascii
```

Receive the trial Certificate into mydbserver.kdb with the following command:

```
$ cat SSLCertificate.cer2
```

```
-----BEGIN CERTIFICATE-----
```



## Before you begin

### Configuring HADR to use SSL

To configure HADR to use SSL communications, procedures are similar to the ones described in Configuring Secure Sockets Layer (SSL) support in a DB2® instance. In particular, the steps that describe how to set up your SSL key database and certificate must be done for all instances in the HADR configuration. The steps for configuring the HADR environment by using a self-signed certificate are described in the following section.

An activated connection concentrator does not inhibit the use of SSL for HADR communications. For SSL support to be enabled, each primary and standby database in the HADR configuration must fulfill all requirements for SSL support.

### Prior to configuring SSL support, perform the following steps on each primary and standby in the HADR configuration

Ensure that the path to the IBM Global Security Kit (GSKit) libraries appears in the *LIBPATH*, *SHLIB\_PATH*, or *LD\_LIBRARY\_PATH* environment variables on Linux and UNIX operating systems. GSKit is automatically included when you install a Db2 database server product.

On UNIX and Linux operating systems, the GSKit libraries are located in *sqllib/lib/gskit*.

On Linux platforms, the GSKit is installed locally when Db2 is installed. The GSKit libraries are located in *sqllib/lib/gskit* or *sqllib/lib64/gskit*. It is unnecessary to have another copy of GSKit installed in a global location to start the instance. If a global copy of GSKit does exist, keep the version of the global GSKit at the same version of the local GSKit.

For information about the GSKit tool *GSKCapiCmd*, see the *GSKCapiCmd User's Guide*, available at [ftp://ftp.software.ibm.com/software/webserver/appserv/library/v80/GSK\\_CapiCmd\\_UserGuide.pdf](ftp://ftp.software.ibm.com/software/webserver/appserv/library/v80/GSK_CapiCmd_UserGuide.pdf).

## About this task

### Configuring SSL support

The general steps for configuring SSL support are:

1. Create a key database on the primary and each standby instance to manage your digital certificates. These certificates and encryption keys are used for establishing the SSL connections.
2. Configure the Db2 instance for SSL support. This step is done by Db2 instance owner.
3. Configured SSL for the particular database for which SSL is to be used.

The procedure section details this configuration process for the communication between primary and standby HADR servers.

### Restrictions

Table 4. SSL support between the HADR primary and standby servers:

| Platform                       | Supported starting in Db2 Version |
|--------------------------------|-----------------------------------|
| Linux on AMD64 and Intel EM64T | 11.1.1.1                          |
| All other platforms            | 11.1.3.3                          |

## Procedure

1. Create a key database and set up your digital certificates on the primary and standby instances.

- a. Use the **gskcapicmd** command to create your key database. The key database type must be a Certificate Management System (CMS). GSKCapiCmd is a non Java based command-line tool, and Java does not need to be installed on your system to use this tool.

The **gskcapicmd** command is described in the *GSKCapiCmd User's Guide*. The path for the command is `sqllib/gskit/bin` on Linux and UNIX operating systems. On Linux and UNIX, ensure that the `LIBPATH`, `SHLIB_PATH`, or `LD_LIBRARY_PATH` environment variables include the proper GSKit library path, such as `sqllib/lib64/gskit`.

For example, the following command creates a key database that is called `myprimary.kdb` and a stash file that is called `myprimary.sth`:

```
gsk8capicmd_64 -keydb -create -db "primary.kdb" -pw "myPrimaryPassw0rdpw0" -stash
```

The `-stash` option creates a stash file at the same path as the key database, with a file extension of `.sth`. At instance start-up, GSKit uses the stash file to obtain the password to the key database.

When you create a key database, it is automatically populated with signer certificates from a few certificate authorities (CAs), such as Verisign.

**Note:** You should use strong file system protection on the stash file. By default, only the instance owner has read and write access to this file. Since this file is a user-managed file, it is not to be stored in the `Db2 sqllib` directory. Create a keystore directory under each instance's home directory to store the key database and stash files. For example,

```
mkdir /home/test/keystore
```

- b. Add a certificate for your primary instance to your key database. The standby instance sends this certificate to the primary instance during the SSL handshake to provide authentication for the standby instance. To obtain a certificate, you can either use the **gskcapicmd** command to create a new certificate request and submit it to a CA to be signed, or you can create a self-signed certificate. The following examples are for a self-signed certificate:

To create a self-signed certificate with a label of *myPrimarysigned*, use the following **gskcapicmd**:

```
gsk8capicmd_64 -cert -create -db "primary.kdb" -pw "myPrimaryPassw0rdpw0" -label "myPrimarysigned" -dn "CN=myhost.mycompany.com,O=myOrganization, OU=myOrganizationUnit,L=myLocation,ST=ON,C=US"
```

To use a CA signed certificate, you must obtain one, as described in *Configuring Secure Sockets Layer (SSL) support in a DB2 instance*.

- c. Extract the certificate that you created to a file so that you can distribute it to each standby instance.

For example, the following **gskcapicmd** command extracts the certificate to a file called `primary.arm`:

```
gsk8capicmd_64 -cert -extract -db "primary.kdb" -pw "myPrimaryPassw0rdpw0" -label "myPrimarysigned" -target "primary.arm" -format ascii -fips
```

- d. Repeat steps 1a through 1c on each standby database.

To create a keydb on the standby:

```
gsk8capicmd_64 -keydb -create -db "standby1.kdb" -pw "myStandby1Passw0rdpw0" -stash
```

Create the certificate on the standby:

```
gsk8capicmd_64 -cert -create -db "standby1.kdb" -pw "myStandby1Passw0rdpw0" -label "myStandby1signed" -dn "CN=myhost.mycompany.com,O=myOrganization, OU=myOrganizationUnit,L=myLocation,ST=ON,C=US"
```



Extract the standby certificate into a file called standby1.arm:

```
gsk8capicmd_64 -cert -extract -db "standby1.kdb" -pw "myStandby1Passw0rdpw0" -label "myStar"
-target "standby1.arm" -format ascii -fips
```

2. Add the primary and standby certificates to the key database at each primary and standby instance.

- a. FTP the file that contains the primary instance's certificate to the standby instance. This file was extracted in a previous step into a file called primary.arm. Also, FTP the file that contains the standby instance's certificate, standby1.arm, to the primary instance. Place these files into the directory where you created your key database on each instance.

- b. Add the primary instance's certificate into the standby's key database.

For example, the following **gsk8capicmd** command imports the certificate from the file primary.arm into the key database called standby1.kdb:

```
gsk8capicmd_64 -cert -add -db "standby1.kdb" -pw "myStandby1Passw0rdpw0" -label "myPrimary"
-file "primary.arm" -format ascii -fips
```

- c. On the primary instance, add the standby's certificate into the primary's key database.

For example, the following **gsk8capicmd** command imports the certificate from the file standby1.arm into the key database called primary.kdb.

```
gsk8capicmd_64 -cert -add -db "primary.kdb" -pw "myPrimaryPassw0rdpw0" -label "myStandby1"
-file "standby1.arm" -format ascii -fips
```

- d. If multiple standby databases exist, the certificate from each instance in the HADR configuration must be imported into the key database of each instance, in the same manner described in steps 2a through 2c.

3. Set up your Db2 instances for SSL support.

To set up your Db2 instances for SSL support, log in as the Db2 instance owner and set the following configuration parameters. This step must be done on the Db2 instance of the primary and all standby databases.

- a. Set the ssl\_svr\_keydb configuration parameter to the fully qualified path of the key database file on the instance. For example, on the primary instance:

```
db2 update dbm cfg using SSL_SVR_KEYDB /home/test/keystore/primary.kdb
```

On the standby instance:

```
db2 update dbm cfg using SSL_SVR_KEYDB /home/test/keystore/standby1.kdb
```

If ssl\_svr\_keydb is null (unset) on any instance in the HADR configuration, SSL support fails.

The paths do not have to be the same on the primary and each standby.

- b. Set the ssl\_svr\_stash configuration parameter to the fully qualified path of the stash file. For example, on the primary instance:

```
db2 update dbm cfg using SSL_SVR_STASH /home/test/keystore/primary.sth
```

On the standby instance:

```
db2 update dbm cfg using SSL_SVR_STASH /home/test/keystore/standby1.sth
```

If ssl\_svr\_stash is null (unset) on any instance in the HADR configuration, SSL support fails.

The paths do not have to be the same on the primary and each standby.

- c. Restart each primary and standby Db2 instance.

```
db2stop
```

```
db2start
```

4. Enable SSL communications for each primary and standby database.

On the primary and each standby database, set the `hadr_ssl_label` database configuration parameter to the label of the digital certificate, which you added in steps 1 on page 74 and 2 on page 75. For example, on the primary database:

```
db2 update db cfg for db2db using HADR_SSL_LABEL myPrimarysigned
```

Where `db2db` is the database name and `myPrimarysigned` is the label that is created in step 1 on page 74.

On the secondary database:

```
db2 update db cfg for db2db using HADR_SSL_LABEL myStandby1signed
```

If `hadr_ssl_label` is set for one primary or standby, then it must be set for all primary and standby databases in the configuration. If `hadr_ssl_label` is not set for all databases, then some HADR connections between primary and standby databases fail.

The label does not have to be the same on each primary and standby.

If the `hadr_ssl_label` is set, then both the `ssl_svr_keydb` and `ssl_svr_stash` must be set. If not, then HADR cannot be started, or some HADR connections between primary and standby databases fail.

#### Related reference:

HADR\_SSL\_LABEL - Label name in the key file for SSL communication between HADR primary and standby instances configuration parameter

## Configuring Secure Sockets Layer (SSL) support in federation server for DRDA wrapper

The Db2 database system supports SSL, which means that a Db2 client application that also supports SSL can connect to a Db2 database using an SSL socket.

### Before you begin

- Ensure to update DB2COMM registry variable

```
db2set -i db2inst1 DB2COMM=SSL
```
- Ensure to enable both TCP/IP and SSL communication protocols

```
db2set -i db2inst1 DB2COMM=SSL,TCPIP
```

**Note:** Specify different service ports for SSL and TCPIP.

- Ensure to enable SSL connection on data source side. For more information, see “Configuring Secure Sockets Layer (SSL) support in a Db2 instance” on page 64. Some important configuration parameters are:
  - SSL\_SVR\_KEYDB
  - SSL\_SVR\_STASH
  - SSL\_SVR\_LABEL
  - SSL\_SVCENAME

### Procedure

To configure SSL support in a federation server:

1. Obtain the signer certificate of the server digital certificate on the client. The server certificate can either be a self-signed certificate or a certificate signed by a certificate authority (CA).
  - If your server certificate is a self-signed certificate, you must extract its signer certificate to a file on the server computer and then distribute it to computers running clients that will be establishing SSL connections to that server. See “Configuring Secure Sockets Layer (SSL) support in a Db2 instance” on page 64 for information about how to extract the certificate to a file.

- If your server certificate is signed by a well known CA, your client key database might already contain the CA certificate that signed your server certificate. If it does not, you must obtain the CA certificate, which is usually done by visiting the website of the CA.
2. On the Db2 client system, use the GSKCapiCmd tool to create a key database, of CMS type. The GSKCapiCmd tool is a non-Java-based command-line tool (Java does not need to be installed on your system to use this tool).

You invoke GSKCapiCmd using the **gskcapiCmd** command, as described in the *GSKCapiCmd User's Guide*. The path for the command is `sqlib/gskit/bin` on Linux and UNIX operating systems, and `C:\Program Files\IBM\GSK8\bin` on both 32-bit and 64-bit Windows operating systems. (On 64-bit operating systems, the 32-bit GSKit executable files and libraries are also present; in this case, the path for the command is `C:\Program Files (x86)\IBM\GSK8\bin`.)

For example, the following command creates a key database called `mydbclient.kdb` and a stash file called `mydbclient.sth`:

```
gsk8capiCmd_64 -keydb -create -db "mydbclient.kdb" -pw "myClientPassw0rdpw0"
 -stash
```

The **-stash** option creates a stash file at the same path as the key database, with a file extension of `.sth`. At connect time, GSKit uses the stash file to obtain the password to the key database.

3. Add the signer certificate into the client key database .

For example, the following **gsk8capiCmd** command imports the certificate from the file `mydbserver.arm` into the key database called `mydbclient.kdb`:

```
gsk8capiCmd_64 -cert -add -db "mydbclient.kdb" -pw "myClientPassw0rdpw0"
 -label "dbselfsigned" -file "mydbserver.arm" -format ascii -fips
```

4. To connect to the data source by using `mydbclient.kdb` and `mydbclient.sth`, perform the following steps

- a. Configure and start the federation server.
- b. Run the CREATE SERVER command.

```
create server SERVERNAME type TYPE version Version_Number wrapper drda authorization "uid"
```

- c. Run the CREATE USER MAPPING command

```
create user mapping for user server SERVERNAME options(remote_authid 'remote_userid',remote
If everything goes fine, now you are connect to data source using SSL. You can check it on
netstat -anp | grep server_ssl_listen_port
```

Connection is established to the data source using SSL. Run the following command to check the connection on the server side.

```
netstat -anp | grep server_ssl_listen_port
```

If the status of server SSL listen port is 'ESTABLISHED' then it is connected.

5. To connect to the data source by using the server signer certificate only, run the following command:

```
create server SERVERNAME type TYPE version Version_Number wrapper drda authorization "uid" pas
```

## Configuring Secure Sockets Layer (SSL) support in non-Java Db2 clients

You can configure Db2 database clients, such as CLI, CLP, and .Net Data Provider clients, to support Secure Sockets Layer (SSL) for communication with the Db2 server.

## Before you begin

**Note:** If your Version 9.7 Db2 client or Db2 Connect server establishes an SSL connection to a Db2 for z/OS server on a z/OS V1.8, V1.9, or V1.10 system, the appropriate PTF for APAR PK72201 must be applied to the Communication Server for z/OS IP Services.

**Note:** Due to an incompatibility between GSKit version 8 and GSKit 7d versions before 7.0.4.20, CLI applications connecting to an IDS data server using GSKit 7d versions before 7.0.4.20 will fail. To correct the problem, upgrade the GSKit library on the IDS data server to GSKit 7.0.4.20 or later

Before configuring SSL support for a client, perform the following steps:

- If both the client and the server are on the same physical computer, you do not need to install GSKit, because GSKit is automatically installed with the Db2 server.

Starting with Version 9.7 Fix Pack 1, when you install the 64-bit version of the Db2 server, the 32-bit GSKit libraries are automatically included in the installation. To use these libraries, on Linux and UNIX operating systems you must ensure that the **LD\_LIBRARY\_PATH**, **LIBPATH**, or **SHLIB\_PATH** environment variable is correctly set. On Windows operating systems, ensure that the **PATH** environment variable is correctly set, as shown in the following table.

| Application | Operating system      | Location of GSKit libraries           | Environment variable setting                                                                                                             |
|-------------|-----------------------|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| 32-bit      | Linux and UNIX 64-bit | <i>\$INSTHOME</i> /sqllib/lib32/gskit | Include <i>\$INSTHOME</i> /sqllib/lib32/gskit in the <b>LD_LIBRARY_PATH</b> , <b>LIBPATH</b> , or <b>SHLIB_PATH</b> environment variable |
| 64-bit      | Linux and UNIX 64-bit | <i>\$INSTHOME</i> /sqllib/lib64/gskit | Include <i>\$INSTHOME</i> /sqllib/lib64/gskit in the <b>LD_LIBRARY_PATH</b> , <b>LIBPATH</b> , or <b>SHLIB_PATH</b> environment variable |
| 32-bit      | Windows 64-bit        | C:\Program Files (x86)\IBM\GSK8\lib   | Include C:\Program Files (x86)\IBM\GSK8\lib in <b>PATH</b> environment variable                                                          |
| 64-bit      | Windows 64-bit        | C:\Program Files\IBM\GSK8\lib64       | Include C:\Program Files\IBM\GSK8\lib64 in <b>PATH</b> environment variable                                                              |

SSL communication will always be in FIPS mode.

On non-Windows platforms, the Db2 database manager installs GSKit locally, and for a given instance, the GSKit libraries would be located in sqllib/lib/gskit or sqllib/lib64/gskit. It is unnecessary to have another copy of GSKit installed in a global location. If a global copy of GSKit does exist, keep the version of the global GSKit at the same version of the local GSKit.

- If the client is being installed in a separate computer, for "C" based clients, you must install GSKit if the clients use SSL to communicate with the servers. You can install the GSKit libraries from the IBM Db2 Support Files for SSL Functionality DVD. Alternatively, you can install from an image that you downloaded from Passport Advantage®.
  - Ensure that the path to the IBM Global Security Kit (GSKit) libraries appears in the **PATH** environment variable on Windows and in the **LIBPATH**, **SHLIB\_PATH**

or **LD\_LIBRARY\_PATH** environment variables on Linux and UNIX. For example, on Windows, add the GSKit bin and lib directories to the **PATH** environment variable:

```
set PATH="C:\Program Files\ibm\gsk8\bin";%PATH%
set PATH="C:\Program Files\ibm\gsk8\lib";%PATH%
```

- From Db2 V10.5 FP5 onwards, if the client is being installed on a separate computer and uses SSL to communicate with servers, ignore numbers two and three of the Procedure section. GSK also does not need to be installed for CLI, .NET, and open source drivers. You can pass the SSL Certificate to these client drivers using the **SSLServerCertificate** keyword in the CLI connection string, **db2cli.ini** file, or **db2dsdriver.cfg** file. The client driver will create an internal key database and add the certificate to it. You do not need to modify any environment variable.

**Note:** GSK is still required for Certificate-based authentication, such as two-way authentication supported by Db2 for z/OS.

### Documentation for the GSKit tool: GSKCapiCmd

For information about the GSKit tool **GSKCapiCmd**, see the *GSKCapiCmd User's Guide*, available at [ftp://ftp.software.ibm.com/software/webserver/appserv/library/v80/GSK\\_CapiCmd\\_UserGuide.pdf](ftp://ftp.software.ibm.com/software/webserver/appserv/library/v80/GSK_CapiCmd_UserGuide.pdf).

### About this task

The SSL communication will always be in FIPS mode.

### Procedure

To configure SSL support in a Db2 client:

1. Obtain the signer certificate of the server digital certificate on the client. The server certificate can either be a self-signed certificate or a certificate signed by a certificate authority (CA).
  - If your server certificate is a self-signed certificate, you must extract its signer certificate to a file on the server computer and then distribute it to computers running clients that will be establishing SSL connections to that server. See “Configuring Secure Sockets Layer (SSL) support in a Db2 instance” on page 64 for information about how to extract the certificate to a file.
  - If your server certificate is signed by a well known CA, your client key database might already contain the CA certificate that signed your server certificate. If it does not, you must obtain the CA certificate, which is usually done by visiting the website of the CA.
2. On the Db2 client computer, use the **GSKCapiCmd** tool to create a key database, of CMS type. The **GSKCapiCmd** tool is a non-Java-based command-line tool (Java does not need to be installed on your system to use this tool).

You invoke **GSKCapiCmd** using the **gskcapiCmd** command, as described in the *GSKCapiCmd User's Guide*. The path for the command is **sqllib/gskit/bin** on Linux and UNIX operating systems, and **C:\Program Files\IBM\GSK8\bin** on both 32-bit and 64-bit Windows operating systems. (On 64-bit operating systems, the 32-bit GSKit executable files and libraries are also present; in this case, the path for the command is **C:\Program Files (x86)\IBM\GSK8\bin**.)

For example, the following command creates a key database called **mydbclient.kdb** and a stash file called **mydbclient.sth**:

```
gsk8capicmd_64 -keydb -create -db "mydbclient.kdb" -pw "myClientPassw0rdpw0"
-stash
```

The **-stash** option creates a stash file at the same path as the key database, with a file extension of **.sth**. At connect time, GSKit uses the stash file to obtain the password to the key database.

3. Add the signer certificate into the client key database

For example, the following **gsk8capicmd** command imports the certificate from the file **mydbserver.arm** into the key database called **mydbclient.kdb**:

```
gsk8capicmd_64 -cert -add -db "mydbclient.kdb" -pw "myClientPassw0rdpw0"
-label "dbselfsigned" -file "mydbserver.arm" -format ascii -fips
```

4. For your client application, set the appropriate connection string or configuration parameters, as shown in the applicable example for your client.

## Example

### CLP and embedded SQL clients

CLP clients and embedded SQL clients can connect to a database on a remote host that has been added to the node catalog using the **CATALOG TCPIP NODE** command. Issue the **CATALOG TCPIP NODE** command with the **SECURITY** keyword set to **SSL** to specify SSL for that connection.

The following example demonstrates how to catalog a node and database so that a CLP client can connect to them using an SSL connection.

First, catalog the node and database so that client applications can establish SSL connections to them:

```
catalog TCPIP NODE mynode REMOTE 127.0.0.1 SERVER 50001 SECURITY SSL
```

```
catalog DATABASE sample AS myssldb AT NODE mynode AUTHENTICATION SERVER
```

Next, use the **ssl\_clnt\_keydb** and **ssl\_clnt\_stash** configuration parameters to specify the client key-database and the stash file. You set the **ssl\_clnt\_keydb** configuration parameter to the fully qualified path of the key database file (**.kdb**) and the **ssl\_clnt\_stash** configuration parameter to the fully qualified path of the stash file:

```
db2 update dbm cfg using
 SSL_CLNT_KEYDB /home/test1/sql1lib/security/keystore/clientkey.kdb
 SSL_CLNT_STASH /home/test1/sql1lib/security/keystore/clientstore.sth
```

If either the **ssl\_clnt\_keydb** or **ssl\_clnt\_stash** configuration parameter is null (unset), the connection fails and returns error **SQL10013N** with token **GSKit Error: GSKit\_return\_code**.

Then, connect to the server from the CLP client:

```
db2 connect to myssldb user user1 using password
```

Alternatively, an embedded SQL application could use the following statement to connect:

```
Strcpy(dbAlias,"myssldb");
EXEC SQL CONNECT TO :dbAlias USER :user USING :pswd;
```

### CLI/ODBC client applications

Depending in which environment you are running your CLI application, you use either connection string parameters (**SSLClientKeystoredb** and **SSLClientKeystash**) or Db2 configuration parameters (**ssl\_clnt\_keydb** and **ssl\_clnt\_stash**) to specify the path for the client key database and for the stash file.

- If you are using the IBM Data Server Driver for ODBC and CLI, you use connection string parameters, as shown in this example:

Call SQLDriverConnect with a connection string that contains the SECURITY=SSL keyword. For example:

```
"Database=sampledb; Protocol=tcip; Hostname= myhost; Servicename=50001;
Security=ssl; SSLClientKeystoredb=/home/test1/keystore/clientstore.kdb;
SSLClientKeystash=/home/test1/keystore/clientstore.sth;"
```

In this case, because Security=ssl is specified, the SSLClientKeystoredb and SSLClientKeystash connection string parameters must be set, otherwise the connection will fail.

- If you are using the IBM data server client or IBM Data Server Runtime Client, you can use either connection string parameters or Db2 configuration parameters to set the path for the client key database and for the stash file. If the **SSLClientKeystoredb** and **SSLClientKeystash** connection string parameters are set, they override any values set by the **ssl\_clnt\_keydb** or the **ssl\_clnt\_stash** configuration parameters.

This example uses the db2cli.ini file to set connection string parameters:

```
[sampledb]
Database=sampledb
Protocol=tcip
Hostname=myhost
Servicename=50001
Security=ssl
SSLClientKeystoredb=/home/test1/keystore/clientstore.kdb
SSLClientKeystash=/home/test1/keystore/clientstore.sth
```

This example uses the **FileDSN** CLI/ODBC keyword to identify a DSN file that contains the database connectivity information, which sets the connection string parameters. For example, the DSN file may look like this:

```
[ODBC]
DRIVER=IBM DB2 ODBC DRIVER – DB2COPY1
UID=user1
AUTHENTICATION=SERVER
PORT=50001
HOSTNAME=myhost
PROTOCOL=TCPIP
DATABASE=SAMPLEDB
SECURITY=SSL
SSLClientKeystoredb=/home/test1/keystore/clientstore.kdb
SSLClientKeystash=/home/test1/keystore/clientstore.sth
```

In these cases, because Security=ssl is specified, if the **SSLClientKeystoredb** and **SSLClientKeystash** connection string parameters are not set, and also the **ssl\_clnt\_keydb** and **ssl\_clnt\_stash** configuration parameters are not set, the connection will fail.

From Db2 V10.5 FP5 onwards, the SSLClientKeystoredb and SSLClientKeystash keywords are not needed in the connection string, db2cli.ini file, FileDSN, or db2dsdriver.cfg file. If you have not set or passed values for the SSLClientKeystoredb and SSLClientKeystash keywords, the CLI/ODBC client driver will create a default key database internally during the first SSL connection. The Client driver will call GSKit API's to create a key database populated with the default root certificates. If the application has passed the signer certificate of the server (\*.arm file) using the SSLServerCertificate keyword, the client driver will add this certificate to this default key database and proceed for the SSL connection. In this case, the application needs to use the Security=SSL and SSLServerCertificate=<certificate file name> as in below connection string.

```
"Database=samp1edb;Protocol=tcpi;Hostname=myhost;Servicename=50001;Security=ssl;SSLServerC
```

If both `SSLClientKeystoredb` and `SSLServerCertificate` keywords are used, the client driver will add the certificate file to the key database which the `SSLClientKeystoredb` points to and then proceed for SSL connection.

### Certificate-based authentication

The certificate-based authentication allows you to use SSL client authentication without the need of providing database passwords on the database client. When certificate-based authentication is configured to supply authentication information, a password cannot be specified in any other way (as in the `db2dsdriver.cfg` configuration file, in the `db2cli.ini` configuration file, or in the connection string). Since the authentication parameter needs a label to be specified, a new data server driver configuration parameter **SSLClientLabel** is also introduced. If the `CERTIFICATE` authentication is specified, then the new label parameter **SSLClientLabel** must also be specified in the CLI configuration file, `db2cli.ini`, or in the data server driver configuration file, `db2dsdriver.cfg`.

The **SSLClientKeyStoreDBPassword** keyword sets the keystore database password. The configuration parameters **SSLClientKeystash** and **SSLClientKeyStoreDBPassword** are mutually exclusive. When the **SSLClientKeystash** configuration parameter and the **SSLClientKeyStoreDBPassword** configuration parameter are both specified in either the CLI configuration file or the data server driver configuration file, error `CLI0220E` is returned. Hence, for a successful certificate-based authentication, it is recommended to specify only one of the keywords but not both.

An example of the IBM data server driver configuration file (`db2dsdriver.cfg`) entry follows:

```
<parameter name="Authentication" value="CERTIFICATE"/>
```

### Db2 .Net Data Provider applications

A Db2 .Net Data Provider application can establish an SSL connection to a database by specifying the path for the client key database and for the stash file by defining the connection string parameters, `SSLClientKeystoredb` and `SSLClientKeystash`. The connection string must also contain `Security=SSL`. For example:

```
String connectionString = "Server=myhost:50001;Database=samp1edb;Security=ssl;
SSLClientKeystoredb=/home/test1/keystore/clientstore.kdb;
SSLClientKeystash=/home/test1/keystore/clientstore.sth";
```

Then, as shown in the following C# code fragment, to connect to a database, pass this **connectString** to the **DB2Connection** constructor and use the **Open** method of the **DB2Connection** object to connect to the database identified in **connectString**:

```
DB2Connection conn = new DB2Connection(connectionString);
Conn.Open();
Return conn;
```

If either the **SSLClientKeystoredb** or **SSLClientKeystash** connection string parameter is null (unset), the connection fails and returns error `SQL10013N` with token `GSKit Error: GSKit_return_code`.

From Db2 V10.5FP5 onwards, you do not need to use the `SSLClientKeystoredb` and `SSLClientKeystash` keywords in the SSL



connection. However, the connection string must also contain Security=SSL. The connection string can be used as below:

```
String connectString = "Server=myhost:50001;Database=sampledb;Security=ssl;SSLServerCert="
```

#### Notes:

- When a self-signed client certificate created in a keystore database is extracted, the extract command extracts the public key data from the keystore database and places it into an identified file. However, no information that is related to the private key is extracted. If such a certificate is imported into another keystore database, only the public key data is imported and the private key data is not present in the new keystore database. So, an authentication of a client using the new keystore database fails. As a solution, either the entire keystore database needs to be shared by various applications or different client certificates need to be generated for each client.

### Secure Sockets Layer (SSL)

The Db2 database system supports the use of Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS), to enable a client to authenticate a server, and to provide private communication between the client and server by use of encryption. Authentication is performed by the exchange of digital certificates.

**Note:** When this topic mentions SSL, the same information applies to TLS, unless otherwise noted.

Without encryption, packets of information travel through networks in full view of anyone who has access. You can use SSL to protect data in transit on all networks that use TCP/IP (you can think of an SSL connection as a secured TCP/IP connection).

A client and server establish a secure SSL connection by performing an "SSL handshake".

### Overview of the SSL handshake

During an *SSL handshake*, a public-key algorithm, usually RSA, is used to securely exchange digital signatures and encryption keys between a client and a server. This identity and key information is used to establish a secure connection for the session between the client and the server. After the secure session is established, data transmission between the client and server is encrypted using a symmetric algorithm, such as AES.

The client and server perform the following steps during the SSL handshake:

1. The client requests an SSL connection and lists its supported cipher suites.
2. The server responds with a selected cipher suite.
3. The server sends its digital certificate to the client.
4. The client verifies the validity of the server certificate, for authentication purposes. It can do this by checking with the trusted certificate authority that issued the server certificate or by checking in its own key database.
5. The client and server securely negotiate a session key and a message authentication code (MAC).
6. The client and server securely exchange information using the key and MAC selected.

**Note:** The Db2 database system does not support the (optional) authentication of the client during the SSL handshake.

## Using SSL encryption with Db2 authentication

You can use SSL encryption in conjunction with all existing Db2 authentication methods, such as KERBEROS or SERVER. You do this as usual by setting the authentication type for the instance in the DBM configuration parameters to the authentication method of your choice.

## Digital certificates and certificate authorities

Digital certificates are issued by trusted parties, called certificate authorities, to verify the identity of an entity, such as a client or server.

The digital certificate serves two purposes: it verifies the owner's identity and it makes the owner's public key available. It is issued with an expiration date, after which it is no longer guaranteed by the certificate authority (CA).

To obtain a digital certificate, you send a request to the CA of your choice, such as Verisign, or RSA. The request includes your distinguished name, your public key, and your signature. A distinguished name (DN) is a unique identifier for each user or host for which you are applying for a certificate. The CA checks your signature using your public key and performs some level of verification of your identity (this varies with different CAs). After verification, the CA sends you a signed digital certificate that contains your distinguished name, your public key, the CA's distinguished name, and the signature of the certificate authority. You store this signed certificate in your key database.

When you send this certificate to a receiver, the receiver performs two steps to verify your identity:

1. Uses your public key that comes with the certificate to check your digital signature.
2. Verifies that the CA that issued your certificate is legitimate and trustworthy. To do this, the receiver needs the public key of the CA. The receiver might already hold an assured copy of the public key of the CA in their key database, but if not, the receiver must acquire an additional digital certificate to obtain the public key of the CA. This certificate might in turn depend on the digital certificate of another CA; there might be a hierarchy of certificates issued by multiple CAs, each depending on the validity of the next. Eventually, however, the receiver needs the public key of the *root* CA. The root CA is the CA at the top of the hierarchy. To trust the validity of the digital certificate of the root CA, the public-key user must receive that digital certificate in a secure manner, such as through a download from an authenticated server, or with preloaded software received from a reliable source, or on a securely delivered diskette.

Many applications that send a digital certificate to a receiver send not just their own certificate, but also all of the CA digital certificates necessary to verify the hierarchy of certificates up to the root CA certificate.

For a digital certificate to be entirely trustworthy, the owner of the digital certificate must have carefully protected their private key, for example, by encrypting it on their computer's hard drive. If their private key has been compromised, an imposter could misuse their digital certificate.

You can use self-signed digital certificates for testing purposes. A self-signed digital certificate contains your distinguished name, your public key, and your signature.

## Public-key cryptography

SSL uses public-key algorithms to exchange encryption key information and digital certificate information for authentication. Public-key cryptography (also known as asymmetric cryptography) uses two different encryption keys: a public key to encrypt data and an associated private key to decrypt it.

Conversely, symmetric key cryptography uses just one key, which is shared by all parties involved in the secure communication. This secret key is used both to encrypt and decrypt information. The key must be safely distributed to, and stored by, all parties, which is difficult to guarantee. With public-key cryptography, the public key is not secret, but the messages it encrypts can only be decrypted by using its associated private key. The private key must be securely stored, for example, in your key database, or encrypted on your computer's hard drive.

Public-key algorithms alone do not guarantee secure communication, you also need to verify the identity of whoever is communicating with you. To perform this authentication, SSL uses digital certificates. When you send your digital certificate to someone, the certificate provides them with your public key. You have used your private key to digitally sign your certificate and so the receiver of the communication can use your public key to verify your signature. The validity of the digital certificate itself is guaranteed by the certificate authority (CA) that issued it.

## NIST SP 800-131A compliance in a Db2 instance

The Db2 Cancun Release adds NIST SP 800-131A compliance. A Db2 instance is not configured by default to comply with NIST SP 800-131A. If you are required to comply with NIST SP 800-131A, you must configure your database instance.

A Db2 instance is strictly compliant with NIST SP 800-131A and encrypts data in-transit when:

- The database manager configuration parameter `SSL_VERSIONS` is set to `TLSV12`, which is recommended, `TLS11`, or `TLS10`.

**Note:** If the `SSL_VERSIONS` parameter is set to `TLSV12` and `TLSV1`, you can take advantage of TVL 1.2 support and fall back on TLS 1.1 or TLS 1.0 support. In this scenario, the database instance is not strictly compliant with NIST SP 800-131A.

- The database manager configuration parameter `SSL_CIPHERSPECS` is set to a symmetric algorithm key length that is greater than or equal to 112.

**Note:** The following list of cipher suites meet this key length requirement.

- `TLS_RSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
- `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`
- `TLS_RSA_WITH_AES_256_CBC_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
- `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`
- `TLS_RSA_WITH_AES_256_CBC_SHA`
- `TLS_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_RSA_WITH_AES_128_CBC_SHA256`

- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_ECDHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_3DES\_EDE\_CBC\_SHA
- TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- The database manager configuration parameter `SSL_SVC_LABEL` specifies a certificate with RSA key length that is greater than or equal to 2048, and that has a digital signature with minimum SHA2.

**Note:** If `SSL_VERSIONS` is set to `TLS12`, certificates that are signed with SHA1 are automatically excluded. SHA1 is not NIST SP800A-131 compliant.

**Note:** For data at rest encryption, you must use InfoSphere Guardium Data Encryption.

## Examples

1. Setting instance configuration parameters so that the instance is strictly compliant with NIST SP 800-131A.

- Set the Db2 registry variable `DB2COMM` to include SSL.  
`DB2SET DB2COMM=TCPIP,SSL`
- Set the Db2 database manager configuration parameter `SSL_VERSIONS` to `TLSV12`.  
`DB2 UPDATE DBM CFG SSL_VERSIONS=TLSV12`
- Set the database manager configuration parameter `SSL_CIPHERSPECS` to a symmetric algorithm key length that is greater than or equal to 112.  
`DB2 UPDATE DBM CFG SSL_CIPHERSPECS=TLS_RSA_WITH_AES_256_GCM_SHA384`
- Set the database manager configuration parameter `SSL_SVC_LABEL` to a certificate with RSA key length that is greater than or equal to 2048. That certificate must also have a digital signature with minimum SHA2.  
`gsk8capicmd_64 -cert ... -size 2048 -sigalg SHA256WithRSA -label "myselfsigned_SHA2_2K" ...`  
`DB2 UPDATE DBM CFG SSL_SVR_LABEL=myselfsigned_SHA_2K`

These settings ensure that all connections over SSL in any CLP or Java application strictly adhere to NIST SP 800-131A.

2. Setting instance configuration parameters to take advantage of TLS 1.2 support, and be ready to fall back to TLS 1.1 or 1.0.

- Set the Db2 registry variable `DB2COMM` to include SSL.  
`DB2SET DB2COMM=TCPIP,SSL`
- Set the Db2 database manager configuration parameter `SSL_VERSIONS` to `TLSV12, TLSV1`.  
`DB2 UPDATE DBM CFG SSL_VERSIONS=TLSV12, TLSV1`

## NIST SP 800-131A compliance and LDAP

The Db2 Cancun Release adds NIST SP 800-131A compliance. If you are required to comply with NIST SP 800-131A, you must configure your LDAP environment.

An LDAP plug-in, and an LDAP server is strictly compliant with NIST SP 800-131A when:

- The TLSV12 is enabled in an LDAP security plug-in.

The following database manager configuration parameters are set to the following values:

```
SRVCON_PW_PLUGIN = IBMLDAPauthserver
CLNT_PW_PLUGIN = IBMLDAPauthclient
GROUP_PLUGIN = IBMLDAPgroups
```

The IBMLDAPSecurity.ini file specifies only TLSV12:

```
LDAP_HOST = myhost
SSL_KEYFILE = /home/xxx/sql/lib/cfg/IBMLDAPSecurity.kdb
SSL_PW = mypassword
ENABLE_SSL = true
FIPS_MODE = true
SECURITY_PROTOCOL = TLSV12
```

- The LDAP server is NIST SP 800-131A compliant when the IBMSLDAPD\_SECURITY\_PROTOCOL is set to TLS12. That ensures other protocols such as SSL 3.0, TLS 1.0 and TLS 1.1 are disabled. The LDAP server must also set IBMSLDAPD\_SSL\_EXTN\_SIGALG to an appropriate value to ensure certificates with valid signature and hash algorithms are used.

With valid configuration in both the LDAP client and server, communication between Db2 LDAP security plug-ins and the LDAP server are NIST SP 800-131A compliant.

## Supported cipher suites

During an SSL handshake, the client and server negotiate which cipher suite to use to exchange data. A cipher suite is a set of algorithms that are used to provide authentication, encryption, and data integrity.

The Db2 database system uses GSKit running in FIPS mode to provide SSL support. GSKit supports the following cipher suites:

- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384
- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA

- TLS\_ECDHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_3DES\_EDE\_CBC\_SHA

The name of each cipher suite specifies the algorithms that it uses for authentication, encryption, and data integrity checking. For example, the cipher suite TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA uses RSA for authentication; AES 256-bit and CBC for encryption algorithms; and SHA-1 for the hash function for data integrity.

During the SSL handshake, the Db2 database system automatically picks the strongest cipher suite supported by both the client and the server. If you want the server to accept only one or more specific cipher suites, you can set the **ssl\_cipherspecs** configuration parameter to any of the following values:

- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384
- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_ECDHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_3DES\_EDE\_CBC\_SHA
- Any combination of these three values. To set multiple values, separate each value by a comma but do not put a space between the values.
- Null. In this case, the strongest available algorithm is automatically picked.

You cannot prioritize which cipher suite is selected. If you set the **ssl\_cipherspecs** configuration parameter, the Db2 database system picks the strongest available cipher suite; this selection does not depend on the order you specify the cipher suites when you set **ssl\_cipherspecs**.

## Bundled library and process rules

When Db2 bundles vendor software that requires GSKit, or when vendor software that requires GSKit bundles Db2, certain rules must be followed.

## Library rule

When Db2 bundles vendor software that requires GSKit, the vendor software provides libraries that Db2 links with. These libraries must follow a certain rule. This rule is called a library rule.

*Library rule: Use short name*

When dynamically loading a GSKit library, the caller must pass the loader function only the base file name of the GSKit library and not the path.

For example, `dlopen("libgsk8ssl_64.so", RTLD_NOW | RTLD_GLOBAL)` is correct, while `dlopen("/usr/opt/ibm/gsk8_64/lib/libgsk8ssl_64.so", RTLD_NOW | RTLD_GLOBAL)` is incorrect.

## Process rule

When vendor software that requires GSKit, bundles Db2, the vendor software links with the IBM data server client. The vendor software must follow a certain rule. This rule is called a process rule.

*Process rule: Set up the environment search path*

A process must set up the environment search path under which it finds the GSKit libraries. The process must do this setup so that the included libraries can load the GSKit libraries from the same location.

On AIX, a process can set `LIBPATH` or `RPATH` of the program to the path of the GSKit libraries. In **setuid** and **setgid** cases, a process can use `db2chglbpath` to include the search path of GSKit in the `RPATH` of the program. Only then can GSKit libraries from that location can be used. On Linux, Sun, and HP-UX, a process can set `LD_LIBRARY_PATH` to the path of the GSKit libraries. In **setuid** and **setgid** cases, a process can use `db2chglbpath` to include the search path of GSKit in the `RPATH` of the IBM data server client library. Only then can GSKit libraries from that location can be used. For example, when a process must use global GSKit in server instances, or it must use its own local GSKit in client or server instances, it might use `db2chglbpath` to change the `RPATH`.

## Symbolic link approach and restriction

When you install Db2 on UNIX and Linux, local GSKit libraries are installed as well. Those libraries are in `<db2_install_path>/lib64/gskit_db2` or `<db2_install_path>/lib32/gskit_db2`.

During the installation of other IBM products another copy of the GSKit libraries might be installed. Depending on the product, these libraries might be either local GSKit libraries or global GSKit libraries. When Db2 and another IBM product that includes GSKit libraries are both installed on the same system, some interoperability issues might arise. These interoperability issues might occur because GSKit allows only libraries from a single GSKit source to exist in any single process. The interoperability issues might lead to unpredictable behavior and runtime errors.

To ensure that a single source of GSKit libraries is used, the symbolic link approach can be used. During an initial Db2 installation, the installer creates a symbolic link `<db2_install_path>/lib64/gskit` or `<db2_install_path>/lib32/`

gskit to <db2\_install\_path>/lib64/gskit\_db2 or <db2\_install\_path>/lib32/gskit\_db2. This location is the default location from which GSKit libraries are loaded.

Products that bundle Db2 and change the symbolic link from the default directory to the library directory of another copy of GSKit must ensure that the newly installed GSKit is at the same or newer level. This restriction applies whether the libraries are global or local. During an upgrade or update of Db2, the symbolic link is preserved. If the newly installed copy has a symbolic link to the default location, the symbolic link that is associated with the older installation copy is preserved. If the newly installed copy has a symbolic link that does not point to the default, the symbolic link that is associated with the newer installation copy is used in the newer installation copy. Some limitations exist since the symbolic link <db2\_install\_path>/lib64/gskit or <db2\_install\_path>/lib32/gskit is in the path of the Db2 installation copy. For example, if two or more instances that are created for any Db2 copy, the symbolic link changes affect all the instances.

The GSKit version included with Db2 is 8.0.50.31.

## Examples

Db2 bundles the LDAP client. The Db2 processes follow the process rule. To follow the process rule, the environment search path in RPATH is set to its local copy of GSKit. LDAP client libraries load GSKit libraries from the same location. LDAP client libraries, which follow library rules, loads GSKit libraries by their base file names when GSKIT\_LOCAL\_INSTALL\_MODE is set.

LDAP server bundles Db2. LDAP processes follow the process rule. The environment search path is set to the global copy of GSKit and IBM data server client libraries loads GSKit libraries from the same location. IBM data server client libraries, which follow library rules, loads GSKit libraries by their base file names.

## IBM InfoSphere Guardium Data Encryption for encryption of data at rest

IBM InfoSphere Guardium Data Encryption is a comprehensive software data security solution that when used in conjunction with native Db2 security provides effective protection of the data and the database application against a broad array of threats.

InfoSphere Guardium Data Encryption helps organizations ensure that private and confidential data is strongly protected and in compliance with regulations and legislative acts. The key benefits of InfoSphere Guardium Data Encryption are:

- Proven, strong data security for the Db2 database system
- Protection of live files, configuration files, log files and back-up data
- Transparent to application, database and storage environments
- Unified policy and key management for protecting data in both online and offline environments
- Meets performance requirements

InfoSphere Guardium Data Encryption enables you to encrypt offline database backups and to encrypt online ("live") database files. This is encryption of data on the disk, sometimes called "data at rest" as opposed to "data in flight", which is travelling over the network.



- For backups, data is encrypted as it is being backed up, so the data on the backup device is encrypted. Should the data need to be recovered, the recovery server recognizes that the data is encrypted and will un-encrypt the data.
- For database files, the operating system data files containing the data from the Db2 database are encrypted. This protects the data files from unauthorized users trying to read the "raw" database file.

InfoSphere Guardium Data Encryption is transparent to users, databases, applications, and storage. No code changes or changes to existing infrastructure are required. InfoSphere Guardium Data Encryption can protect data in any storage environment, while users continue to access data the in the same way as before.

InfoSphere Guardium Data Encryption can protect database applications, because it can prevent changes to executable files, configuration files, libraries, and so on, thereby preventing attacks on the application.

**Note:** For Db2 pureScale environments, InfoSphere Guardium Data Encryption is supported only on AIX. InfoSphere Guardium Data Encryption is not supported on other platforms that are running Db2 pureScale environments.

## Architecture of InfoSphere Guardium Data Encryption

InfoSphere Guardium Data Encryption is a set of agent and server software packages that you administer by using a Web-based user-interface and command-line utilities. The InfoSphere Guardium Data Encryption administrator configures security policies that govern how security and encryption are implemented.

According to how these security policies are defined, the InfoSphere Guardium Data Encryption backup agent encrypts Db2 backups, and the InfoSphere Guardium Data Encryption file system agent encrypts Db2 data files.

The security server stores the security policies, encryption keys and event log files. Security policies contain sets of security rules that must be satisfied in order to allow or deny access. Each security rule evaluates who, what, when, and how protected data is accessed and, if these criteria match, the security server either permits or denies access.

Figure 4 on page 92 illustrates the architecture of InfoSphere Guardium Data Encryption.

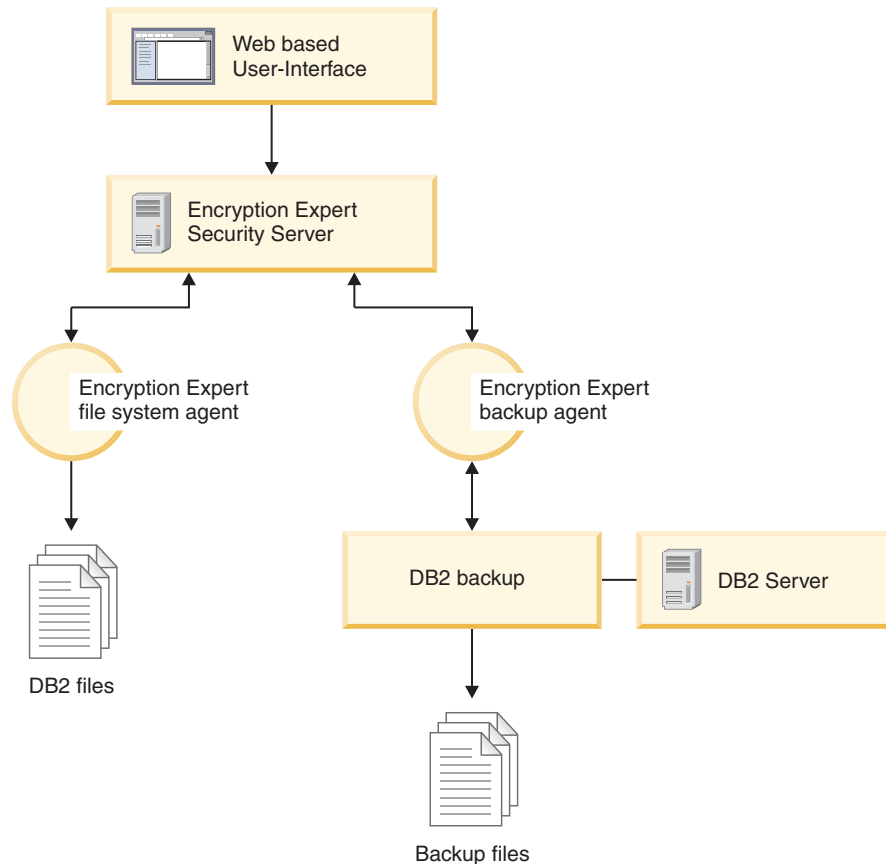


Figure 4. Architecture of InfoSphere Guardium Data Encryption

## File system agent

The InfoSphere Guardium Data Encryption file system agent process is always running in the background. The agent intercepts any attempt to process is always running in the background. The agent intercepts any attempt to access data files, directories, or executables that you are protecting. The InfoSphere Guardium Data Encryption file system agent forwards the access attempt to the security server and, based upon the applied policy, the security server grants or denies the attempted access.

InfoSphere Guardium Data Encryption protection extends beyond simply allowing or denying access to a file, you can also encrypt files. Just the file contents is encrypted, but the file metadata is left intact. Therefore, you do not have to decrypt an encrypted file just to see its name, timestamps, file type, and so on. This allows data management applications to perform their functions without exposing the file contents. For example, backup managers can backup specific data, without being able to see the contents.

If an encrypted file is accessed by an unauthorized user, its contents are worthless without the appropriate security server approval and encryption keys. However, users with the correct policies and permissions are unaware that encryption and decryption are taking place.

## Backup agent

All database backup functions that are normally performed by the Db2 backup API system are supported by the InfoSphere Guardium Data Encryption server,

including native database compression. Other than an additional command-line argument, Db2 backup operators are unaware of InfoSphere Guardium Data Encryption intervention. InfoSphere Guardium Data Encryption backs up and restores static data-at-rest and active online data.

Basic backup and restore configuration is supported. In the basic configuration, data is encrypted and backed up with one server and multiple agents; data is decrypted and restored on an agent that is configured with the same server that was originally used to make the backup.

Single-site and multi-site configurations are also supported for backup and restore. In a single-site scenario, configuration data is mirrored across multiple security servers in a single data center. In a multi-site scenario, backups are restored on different security servers in different data centers.

## Audit logging

InfoSphere Guardium Data Encryption agent activity is closely monitored and logged through a centralized audit logging facility. All auditable events, including backups, restores, and security administration operations can be logged. This includes InfoSphere Guardium Data Encryption system events, such as initialization, shut down and restart; and network connects and disconnects between different InfoSphere Guardium Data Encryption components.

## Database encryption using AIX encrypted file system (EFS)

If you are running a Db2 system on the AIX operating system, you have the option to set up an encrypted database by using AIX encrypted file system (EFS). For detailed information about EFS, see your AIX documentation.

**Note:** If you are working in a partitioned database environment, to use EFS, your database should be in a single database partition.

You can encrypt the operating system files that contain the data in database tables by using the underlying EFS with JFS2 file system.

To set up encryption, the steps are as follows:

1. Enable EFS on the system.
2. Load the keystores for the user account under which the Db2 database daemons run.
3. Enable EFS on the database file system.
4. Determine the operating system file to encrypt.
5. Encrypt the file that contains the database table that requires EFS protection.

## Enabling EFS on the system

Before you enable EFS, the `clic.rte` fileset must be installed. The `clic.rte` install image can be found on the Expansion Pack CD.

Run the following command as root to enable EFS on the system:

```
% efsenable -a
```

You need to run the **efsenable** command only once.

## Loading the keystores

In the following configuration examples, the Db2 user account under which the database daemons run is called abst. The user abst must have a keystore and any group that abst is a member of must also have a keystore.

1. All keystores must be associated with the abst process before starting the Db2 daemons.

You can verify that they are associated by using the **efskeymgr -V** command, as shown in the following example:

```
lsuser abst
abst id=203 pgrp=abstgp groups=abstgp,staff ...

efskeymgr -V
List of keys loaded in the current process:
 Key #0:
 Kind User key
 Id (uid / gid) 203
 Type Private
key
 Algorithm RSA_1024
 Validity Key is
valid
 Fingerprint
24c88df2:d91cb6a2:c3e11b6a:4c13f8b4:666fabd8

 Key #1:
 Kind Group
key
 Id (uid / gid) 1
 Type Private
key
 Algorithm RSA_1024
 Validity Key is
valid
 Fingerprint
03fead42:57e7646e:a1715626:cfa56c8e:8abed1c1

 Key #2:
 Kind Group
key
 Id (uid / gid) 212
 Type Private
key
 Algorithm RSA_1024
 Validity Key is
valid
 Fingerprint
339dfb19:bc850f4c:5551c975:7fe4961b:2dddf3bc
```

2. If there are no keystores shown as associated with the abst process, try loading the keystores using the command: % efskeymgr -o ksh

This command prompts for the keystore password, which is initially set to the login password.

3. Confirm that the user and group keys are loaded by rerunning the command: % efskeymgr -V

Both the user and group keys should be listed. If the group keystores are still not listed, continue with Step 4.

4. Depending on how a group was created, the group keystore may not exist. If the **efskeymgr -V** command does not list the user's group keystores, you must create the group keystores.

As root or the RBAC role aix.efs\_admin, create the group keystore:

```
% efskeymgr -C group_name
```

5. Assign group keystore access to each applicable user:

```
% efskeymgr -k group /group_name -s user/user_name
```

If a user is already logged in, they will not immediately have access to the group keystore, and they should reload their keystore using the **efskeymgr -o ksh** command, or re-login.

## Enabling EFS on the database file system

EFS only runs on JFS2 file systems and must be specifically enabled.

If your database resides on an existing file system, run the `% chfs -a efs=yes filesystem` command to enable EFS, for example:

```
% chfs -a efs=yes /test01
```

If you are creating a new file system, you can enable EFS using the `-a efs=yes` option with the **smitty** command or the **crfs** command. For example:

```
% crfs -v jfs2 -a efs=yes -m mount_point -d devide -A yes
```

EFS is now enabled on the file system but is not turned on. Turn on EFS only for the particular database tables requiring encrypted data (for more information, see your AIX EFS documentation about the **efsmgr** command and inheritance).

## Determining the file to encrypt

To determine which file contains a particular database table that you want to protect with EFS encryption, follow these steps that use the EMPLOYEE table as an example.

1. Use a query similar to the following example to find the TBSPACEID for the table:

```
SELECT TABNAME, TBSPACEID FROM syscat.tables WHERE tabname='EMPLOYEE'
```

Assume the results of this query are as follows:

TABNAME	TBSPACEID
EMPLOYEE	2

2. Look up the table spaces for that TBSPACEID with a query similar to the following example:

```
LIST TABLESPACE CONTAINERS FOR 2
```

Assume the results of this query are as follows:

Container ID	Name	Type
0	/test01/abst/NODE0000/BAR/T0000002/C0000000.LRG	File

You now know that this table space is contained in the operating system file called `/test01/abst/NODE0000/BAR/T0000002/C0000000.LRG`. This is the file you need to encrypt.

## Encrypting the file

First, as you would do before making any major change to data or databases, back up your database.

Follow these steps to encrypt the file:

1. List the file, for example:

```
ls -U /test01/abst/NODE0000/BAR/T0000002/C0000000.LRG
```

```
-rw----- 1 abst abstgp 33554432 Jul 30 18:01
/test01/abst/NODE0000/BAR/T0000002/C0000000.LRG
```

2. Encrypt the file using the **efsmgr** command, for example:

```
efsmgr -e /test01/abst/NODE0000/BAR/T0000002/C0000000.LRG
```

If you list the file again you will see an “e” at the end of the permissions string that indicates the file is encrypted. For example:

```
ls -U /test01/abst/NODE0000/BAR/T0000002/C0000000.LRG
```

```
-rw-----e 1 abst abstgp 33554432 Jul 30 18:03
/test01/abst/NODE0000/BAR/T0000002/C0000000.LRG
```

3. Start the Db2 database manager and use it as normal. All data added to the EMPLOYEE table and this encrypted table space will be encrypted by EFS in the underlying file system. Whenever the data is retrieved, it will be decrypted and presented as normal through the Db2 database manager.

## Db2 native encryption

You can encrypt your databases and backup images using Db2 native encryption. Native encryption provides transparent and secure key management and requires no changes to your hardware, software, applications, or schemas.

### Components

#### Data encryption key

The database system encrypts data with a *data encryption key* before the data is written to disk. The data encryption key is stored, encrypted, in the database (or backup image.)

#### Master key

A *master key* is an encryption key that is used to encrypt a data encryption key. Master keys are specified by a *label*. Each encrypted database has one master key.

#### Keystore

Master keys are stored in a file or database called a *keystore*.

- You can use one of the following keystores:
  - A local keystore that is located on the same system as the Db2 server
  - A centralized keystore that is located on a system other than the Db2 server
  - A PKCS #11 keystore that is located on a system other than the Db2 server.

The primary benefit of a PKCS #11 keystore is the protection it provides to encryption keys. This protection is accomplished by imposing a restriction that keys never leave the secure environment of the keystore. Data on disk is encrypted with a data encryption key (DEK) that is stored with the database. The DEK, in turn, is encrypted by a master key (MK), which is stored externally to the database. The DEK is sent to the PKCS #11 keystore, where it is decrypted by the MK. The only exception to this principle of keys not leaving the keystore is when migrating keys from a local keystore file to a PKCS #11 keystore. In such cases, these keys are marked as

external. However, an immediate key rotation following migration will start to make use of internally defined keys.

Using a centralized keystore or a PKCS #11 keystore is useful when you have multiple databases and you do not want to maintain individual keystores.

- A password is required to access the keystore. You can optionally store (or *stash*) the keystore password, in obfuscated form, in a *stash file*. Stashing the password makes it possible for the database manager to access the keystore (when the database manager starts, for example) without someone having to manually enter the keystore password.
- For use with native encryption, a local keystore must be compliant with the public-key cryptography standard 12 (PKCS#12).
- A Db2 instance can be configured to use only one keystore for native encryption at a time.

### Keystore password

*Keystore password* is a broad term whose definition depends of the keystore that you use:

#### Local keystore

The local keystore password is the password to the PKCS#12 keystore file. The location of this file is defined by the **keystore\_location** database manager configuration parameter. This keystore file holds the master keys that are used for Db2 native encryption.

#### Centralized keystore

The centralized keystore password is the password to the centralized keystore file. The location of this file is defined by the **SSL\_KEYDB** parameter in the centralized keystore configuration file. This keystore file holds the SSL certificates for communication between the Db2 server and the centralized key manager.

#### PKCS #11 keystore

The PKCS #11 keystore password is the string that is used to authenticate the user to the PKCS #11 keystore. This password can be specified in one of two ways:

- By providing the keystore password interactively when you run the **db2start** command with the **OPEN KEYSTORE USING** *password* option.
- By using a stash file, whose location is defined by the **KEYSTORE\_STASH** parameter in the PKCS #11 keystore configuration file.

### Key manager

A *key manager* is software that you can use to create, update, and secure a keystore:

- You can use IBM Global Security Kit (GSKit) to manage a local keystore.
- For a centralized keystore, you can use any key manager that supports the Key Management Interoperability Protocol (KMIP) version 1.1 or higher. For example, you can use IBM Security Key Lifecycle Manager to manage a centralized keystore.
- For a PKCS #11 keystore, you can use one of the following supported key managers:
  - Gemalto Safenet HSM (formerly Luna) version 6.1 (firmware version 6.23.0) and above

- Thales nShield HSM, security world software version 11.50

### Database configuration

#### **encrlib**

Specifies the absolute path of the encryption library to be used for automatically encrypting backup images.

#### **encropts**

Specifies options that are specific to the encryption library to be used for automatically encrypting backup images.

### Database manager configuration

#### **keystore\_type**

Specifies the type of keystore. For more information, see `keystore_type` - Keystore type configuration parameter.

#### **keystore\_location**

Specifies where to find the keystore. For more information, see `keystore_location` - Keystore location configuration parameter.

## Prerequisites for Db2 native encryption

The Db2 native encryption feature is included in all Db2 offerings.

## Licensing

The following editions of Db2 database include the Db2 native encryption license:

- Developer Edition
- Db2 Enterprise Server Edition
- Db2 Workgroup Server Edition
- Db2 Advanced Enterprise Server Edition
- Db2 Advanced Workgroup Server Edition

## GSKit

To use Db2 native encryption, you must verify that GSKit is installed and configured.

### Verifying GSKit installation and configuration:

For the Db2 native encryption feature to work properly, IBM Global Security Kit (GSKit) must be installed, and the environment must be configured correctly.

### Procedure

1. Verify that GSKit is installed:

#### **Linux and UNIX operating systems**

- On Linux and UNIX operating systems, the Db2 installer installs GSKit locally. For each instance, the GSKit libraries will be located in `sql1lib/lib32/gskit` or `sql1lib/lib64/gskit`.
- If a global copy of GSKit exists (for example, in `/usr/lib` on Linux or UNIX) keep the global copy of GSKit and the copies of GSKit installed by the Db2 installer at the same version level.

#### **Windows operating systems**

You must install GSKit manually. See: GSKit installation instructions for Windows



2. Verify that the path to the GSKit libraries is set in the appropriate environment variable for your operating system:

*Table 5. Environment variable settings for GSKit libraries on Linux, UNIX, and Windows operating systems*

Operating system	Location of GSKit libraries	Environment variable setting
Linux and UNIX 32-bit	<code>\$INSTHOME/sqllib/lib32/gskit</code>	Include <code>\$INSTHOME/sqllib/lib32/gskit</code> in the <b>LD_LIBRARY_PATH</b> , <b>LIBPATH</b> , or <b>SHLIB_PATH</b> environment variable.
Linux and UNIX 64-bit	<code>\$INSTHOME/sqllib/lib64/gskit</code>	Include <code>\$INSTHOME/sqllib/lib64/gskit</code> in the <b>LD_LIBRARY_PATH</b> , <b>LIBPATH</b> , or <b>SHLIB_PATH</b> environment variable.
Windows 32-bit	<code>C:\Program Files (x86)\IBM\gsk8\lib</code>	Include <code>C:\Program Files (x86)\IBM\gsk8\lib</code> in the <b>PATH</b> environment variable.
Windows 64-bit	64-bit GSKit libraries: <code>C:\Program Files\IBM\gsk8\lib64</code>  32-bit GSKit libraries: <code>C:\Program Files (x86)\IBM\gsk8\lib</code>	Include <code>C:\Program Files\IBM\gsk8\lib64</code> or <code>C:\Program Files (x86)\IBM\gsk8\lib</code> in the <b>PATH</b> environment variable.

## Setting up Db2 native encryption

To use Db2 native encryption, you need to set up a keystore for storing master keys and then you need to configure the Db2 instance to use the keystore.

### Before you begin

Verify GSKit installation and configuration

### Procedure

1. Set up the keystore:
  - If you will be using a local keystore or a centralized keystore, create a local keystore.
  - If you will be using a centralized keystore to store master keys, set up a centralized keystore.
  - If you will be using a PKCS #11 keystore, set up the PKCS #11 keystore
2. Configure the DB2 instance for the keystore where you will store master keys.

### What to do next

- Create an encrypted database
- Create an encrypted backup image

### Creating a local keystore:

You can create a keystore on the local system using the GSKit library command **gsk8capicmd\_64**.

## Procedure

Log in as the Db2 instance owner, and then create the local keystore by executing the **gsk8capicmd\_64** command.

### Example

```
gsk8capicmd_64 -keydb -create -db "/home/thomas/keystores/ne-keystore.p12"
-pw "g00d.pWd" -type pkcs12 -stash
```

### Basic command syntax

```
gsk8capicmd_64 -keydb -create -db "<file-name>" -pw "<password>" -type pkcs12 -stash
```

- <file-name> is the full path and file name you want to give the keystore file
- Keystore format:
  - For use with native encryption, the format of the keystore must be PKCS#12, so it is mandatory to specify -type pkcs12
  - PKCS#12 keystore file names must have the extension ".p12"
- Stashing the password:
  - If you specify the -stash parameter, the keystore password will be stored (or *stashed*) in a stash file with the same base name as the keystore file but with the file extension ".sth"
  - If the password is not stashed, you will be prompted for a password whenever the database manager accesses the keystore, including during db2start
  - You can stash the password in a stash file later by issuing the **gsk8capicmd\_64** command with the -stashpw parameter

**Note:** Stashing the password with the **gsk8capicmd\_64** command is intended to be used in a local keystore only. Do not attempt to stash a password in a local keystore with the **db2credman** command. The **db2credman** command is intended to be used with a PKCS #11 keystore.

For information about the full syntax of the **gsk8capicmd\_64** command, see: GSKCapiCmd Users Guide .

*Adding a master key to a local keystore:*

With Db2 native encryption, when you create a database with the **ENCRYPT** parameter, by default the database manager creates a new master key for the database and adds that master key to the keystore. Alternatively, you can generate a master key in a local keystore yourself, and then specify that your generated master key should be used for a new database instead of the default.

## Procedure

Generate a master key in an existing, local keystore by issuing the **gsk8capicmd\_64** command.

### Example

```
gsk8capicmd_64 -secretkey -create -db "/home/thomas/keystores/ne-keystore.p12"
-stashed -label "my_manual_master_key" -size "16"
```

### Basic syntax

```
gsk8capicmd_64 -secretkey -create -db "<keystore-file-name>"
[-pw "<password>" | -stashed]
-label "<label>" -size "<key-length-in-bytes>"
```

- <keystore-file-name> is the full path and name of the keystore file

- If the keystore password is stashed, you can specify the `-stashed` parameter to cause the password to be retrieved from the stash file
- If the password is not stashed, you may specify the password with the `-pw` parameter
- If neither `-stashed` nor `-pw` is specified, you will be prompted for the keystore password

For information about the full syntax of the **gsk8capicmd\_64** command, see: GSKCapiCmd Users Guide.

### Setting up a centralized keystore:

To set up a centralized keystore for use with Db2 native encryption, you need to create a centralized keystore configuration file and configure SSL communication between the Db2 instance and the key manager.

#### Before you begin

Set up the centralized key manager.

- If you are using IBM Security Key Lifecycle Manager, see: Quick Start Guide

#### Procedure

1. Create a centralized keystore configuration file
2. Configure SSL between the Db2 instance and the key manager using one of the following methods:
  - The KMIP server must support TLS 1.2.
  - All certificates must be signed with a signature algorithm that uses SHA2 (SHA256, SHA384, SHA512). The use of SHA1 is not supported.
  - All certificates must have a key size of at least 2048 bits.
  - "All certificates" above refers to the Db2 client certificate, the KMIP server certificate, and any Certificate Authority (CA) and intermediate CA root certificates.
  - Configure SSL with ISKLM
  - Configure SSL with KeySecure

**Note:** Other key manager products can be configured in a similar manner.

#### What to do next

Configure the DB2 instance to use this centralized keystore to store database master keys for Db2 native encryption.

*Creating a centralized keystore configuration file:*

To store master keys in a centralized keystore with Db2 native encryption, you need to create a configuration file that lists details about the centralized keystore.

#### Procedure

On the Db2 server, create the centralized keystore configuration file in a text editor.

## Example

```
VERSION=1
PRODUCT_NAME=ISKLM
ALLOW_KEY_INSERT_WITHOUT_KEYSTORE_BACKUP=true
SSL_KEYDB=/home/userName/sql1ib/security/keydb.p12
SSL_KEYDB_STASH=/home/userName/sql1ib/security/keydb.sth
SSL_KMIP_CLIENT_CERTIFICATE_LABEL=db2_client_label
DEVICE_GROUP=DB2
MASTER_SERVER_HOST=serverName.domainName
MASTER_SERVER_KMIP_PORT=kmipPortNumber
CLONE_SERVER_HOST=clone1.domainName
CLONE_SERVER_KMIP_PORT=kmipPortNumber
CLONE_SERVER_HOST=clone2.domainName
CLONE_SERVER_KMIP_PORT=kmipPortNumber
```

## Keywords

### VERSION

Required. Version of the configuration file. Currently, 1 is the only supported value.

### PRODUCT\_NAME

Required. Key manager product. Supported values:

- ISKLM for IBM Security Key Lifecycle Manager
- KEYSECURE for SafeNet KeySecure
- OTHER for any other key manager that supports the Key Management Interoperability Protocol (KMIP) version 1.1 or higher

### ALLOW\_KEY\_INSERT\_WITHOUT\_KEYSTORE\_BACKUP

Optional. Allow the database manager to insert new keys into the centralized key manager. New keys are inserted when the **CREATE DATABASE ENCRYPT** or **ADMIN\_ROTATE\_MASTER\_KEY** commands are run without a specified existing master key label, or when the migration tool **db2p12tokmip** is run. When this parameter is set to TRUE, new keys are allowed to be inserted, if set to FALSE an error is returned if the database manager attempts to insert a new key. You should only set this to TRUE if you are not creating your master keys within the centralized key manager, and you have an automated backup solution of your centralized key manager for newly inserted keys. This parameter must be set to TRUE if you are migrating keys by using the **db2p12tokmip** command. It can be changed to FALSE after the tool has completed. Default value: FALSE.

### ALLOW\_NONCRITICAL\_BASIC\_CONSTRAINT

Optional. If you set the parameter to TRUE, this allows Db2 to use local Certificate Authority within KMIP server that does not have a "critical" keyword set and avoids "414" error that is returned by GSKit. This parameter was introduced in Db2 V11.1.2.2. Default value: FALSE.<sup>1</sup>

---

1. Error SQL1782N is returned by the GSKit layer (manifested as error DIA3604E: The SSL function "gsk\_secure\_soc\_init" failed with the return code "414" in "sqlccSSLSocketSetup" in the db2diag.log) in case the basic constraints extension of the certificate that is issued by the Certificate Authority (CA) does not have the 'critical' keyword asserted. Using the command "gsk8capicmd\_64 -cert -details -db <filename> -stashed -label <localCALabel>" you can check the basic constraints of the CA to see whether the keyword 'critical' is asserted. For a local CA the keyword 'critical' might not be set.

Example:

#### **SSL\_KEYDB**

Required. Absolute path and name of the local keystore file that holds the SSL certificates for communication between the Db2 server and the centralized key manager.

#### **SSL\_KEYDB\_STASH**

Optional. Absolute path and name of the stash file for the local keystore that holds the SSL certificates for communication between the Db2 server and the centralized key manager. Default value: None.

#### **SSL\_KMIP\_CLIENT\_CERTIFICATE\_LABEL**

Required. The label of the SSL certificate for authenticating the client during communication with the centralized key manager.

#### **DEVICE\_GROUP**

Name of the centralized key manager device group containing the keys used by the Db2 server. Not all centralized key manager products support this parameter. This parameter is required for IBM Security Key Lifecycle Manager (ISKLM).

#### **MASTER\_SERVER\_HOST**

Required. Host name or IP address of the centralized key manager. (For ISKLM, this information is available on the "Welcome" tab of the web console.)

#### **MASTER\_SERVER\_KMIP\_PORT**

Required. The "KMIP SSL port" of the centralized key manager. (For ISKLM, this information is available on the "Welcome" tab of the web console.)

#### **CLONE\_SERVER\_HOST**

Optional. Host name or IP address of secondary centralized keystore. Default value: None. You can specify up to five clone servers by repeating the **CLONE\_SERVER\_HOST** and **CLONE\_SERVER\_KMIP\_PORT** parameter pairs in the configuration file, each host with a different value. Clone servers are considered read only and are only used for retrieving existing master keys from the centralized keystore. Clone servers are not used when inserting a new key, which occurs when an existing master key label has not been specified for the **CREATE DATABASE ENCRYPT** or **ADMIN\_ROTATE\_MASTER\_KEY** commands, or for the **db2p12tokmip** executable.

#### **CLONE\_SERVER\_KMIP\_PORT**

Optional. The "KMIP SSL port" of secondary centralized keystore. Default value: None. You can specify up to five clone servers by repeating the **CLONE\_SERVER\_HOST** and **CLONE\_SERVER\_KMIP\_PORT** parameter pairs in the configuration file, each host with a different value.

#### **COMMUNICATION\_ERROR\_RETRY\_TIME**

Optional. The number of times the Db2 database manager cycles

---

Extensions  
basicConstraints  
ca = true  
pathLen = 140730370034921  
critical

through the list of configured master and clone centralized key managers if the connection fails or an error is returned from all of the centralized key managers. A wait of a length specified in the **ALL\_SERVER\_UNAVAILABLE\_SLEEP** parameter is inserted before each cycle. Default value: 50.

#### **UNAVAILABLE\_SERVER\_BLACKOUT\_PERIOD**

Optional. The amount of time, in seconds, to skip sending key requests to a particular master or clone centralized key manager after a failed connection attempt or it has returned errors. This parameter was introduced in Db2 V11.1.2.2. Default value: 300 seconds.

#### **ALL\_SERVER\_UNAVAILABLE\_SLEEP**

Optional. When all master and clone centralized key managers are unavailable and in a blackout period, this parameter is the amount of time to wait, in seconds, before removing the blackout period and reattempting connections to all centralized key managers. This parameter was introduced in Db2 V11.1.2.2. Default value: 0 seconds.

*Configuring SSL between a Db2 instance and a centralized key manager (ISKLM):*

To store master keys in a centralized keystore with Db2 native encryption, you need to set up SSL communication between the Db2 instance and the centralized key manager.

#### **Before you begin**

On the Db2 server, create a local keystore to store SSL certificates.

#### **About this task**

- On the Db2 server, the **gsk8capicmd\_64** command is used to create, extract, and add SSL certificates to the local keystore. For detailed information about the command, see: GSKCapiCmd Users Guide .
- Some examples below show self-signed certificates. Self-signed certificates are suitable for test environments, but for production environments certificates that are signed by third party certificate authorities are more appropriate.
- Some information about using the IBM Security Key Lifecycle Manager web interface and command line interface is included below. For more complete information, see: Setup for SSL handshake between IBM Security Key Lifecycle Manager server and client device.

#### **Procedure**

1. On the Db2 server: create an SSL signer certificate.
  - a. Create the certificate by issuing the **gsk8capicmd\_64** command.

##### **Example**

```
gsk8capicmd_64 -cert -create -db "clientkeydb.p12"
-label "DB2_signer_certificate"
-dn "CN=weblinux.Raleigh.ibm.com,O=ibm,OU=IBM HTTP Server,L=RTP,ST=NC,C=US"
-sig_alg SHA256_WITH_RSA -size 2048
```

- b. Extract the certificate to a file by issuing the **gsk8capicmd\_64** command.

### Example

```
gsk8capicmd_64 -cert -extract -db "clientkeydb.p12"
-label "DB2_signer_certificate"
-target "/path/to/DB2_certificate_file.pem"
-format ascii -fips
```

- c. Securely transmit the Db2 server certificate file to the centralized key manager.
2. On the centralized key manager: add the Db2 server certificate to the keystore.  
The following substeps describe how to add a certificate to IBM Security Key Lifecycle Manager using the web console.
  - a. Create a device group :
    - 1) Select "Create" in the "Device Group" list of the "Advanced Configuration" tab.
    - 2) Select the device family "General Parallel File System (GPFS)" and then enter "DB2" as the new device group name.
    - 3) Leave the "Enable machine affinity" check box unselected.
  - b. Import the DB2 server certificate file :
    - 1) On the "Welcome" tab select your new group, "DB2".
    - 2) From the "Go to" list, select "Manage Keys and Devices". This will bring you to the Advanced Configuration tab.
    - 3) Select "Certificates" from the "Add" list.
    - 4) Specify the certificate name and the file path when prompted.
    - 5) In the "Advanced Configuration" window, select "Import" from the "Client Device Communication Certificates" menu.
3. On the centralized key manager: create an SSL signer certificate.  
The following substeps describe how to create a certificate and then extract it to a file using the IBM Security Key Lifecycle Manager web console and command-line interface.
  - a. Create a self-signed certificate or obtain a certificate from a certificate authority.
  - b. Extract the certificate to a file using the command-line interface:
    - 1) Enable the Jython scripting language.

### Example

```
./wsadmin.sh -username "<admin-user>"
-password "<password>" -lang jython
```

- 2) Export the certificate using the tklmCertExport command.

### Example

```
print AdminTask.tklmCertExport
(['-uuid CERTIFICATE-61f8e7ca-62aa-47d5-a915-8adbfbdca9de'
-format DER
-fileName d:\\ISKLM_certificate_file.pem'])
```

- c. Securely transmit the centralized key manager certificate file to the Db2 server.
4. On the Db2 server: add the centralized key manager certificate to the local keystore.
  - a. Add the certificate by issuing the **gsk8capicmd\_64** command.

### Example

```
gsk8capicmd_64 -cert -add -db "clientkeydb.p12"
-label "ISKLM_signer_certificate"
-file "/path/to/ISKLM_certificate_file.pem"
```

## Results

When the Db2 database manager connects to the centralized key manager, SSL communication will be used.

## What to do next

Configure the DB2 instance to use the centralized keystore

*Configuring SSL between a Db2 instance and a centralized key manager (KeySecure):*

To store master keys in a centralized keystore with Db2 native encryption, you need to set up SSL communication between the Db2 instance and the centralized key manager.

## Before you begin

On the Db2 server, create a local keystore to store SSL certificates.

## About this task

- On the Db2 server, the **gsk8capicmd\_64** command is used to create, extract, and add SSL certificates to the local keystore. For detailed information about the command, see: GSKCapiCmd Users Guide.

## Procedure

*On KeySecure, create a CA and add it to the Trusted CA list:*

1. Verify that a CA certificate is created or installed. Make sure that the CA is added to the trusted CA list.
2. Make sure that a server certificate request is created and signed with the CA certificate.
3. Check that a Cryptographic Key Server is created. Also, verify that the appropriate authentication settings are configured.
  - a. Ensure the appropriate Cryptographic Key Server Properties:
    - **Protocol:** Select KMIP.
    - **IP:** Select ALL or a specific IP address.
    - **Port:** Select a port number. The standard KMIP port number is 5696. In the centralized keystore configuration file, the value for the `MASTER_SERVER_KMIP_PORT` or `CLONE_SERVER_KMIP_PORT` parameter must be configured according to the value specified for the port number.
    - **Use SSL:** Select True
    - **Server Certificate:** Select the label of the server certificate.
  - b. Ensure the appropriate Authentication Settings:
    - **Password Authentication:** Select *Not Used*.
    - **Client Certification Authentication:** Select *Used for SSL session and username*.
    - **Trusted CA list Profile:** Select the profile that contains the Trusted CA list to which the CA was added.
    - **User name Field in Client Certificate:** Select either the CN or OU value from the dropdown list.
    - **Require Client Certificate to Contain Source IP:** Leave unticked.



- c. Create a Local User whose user name matches the *User name field in Client Certificate* field in the client certificate.
4. Download the CA certificate to the client keystore.

*On the Db2 server, add the CA certificate and create a client certificate request:*

5. Add the CA certificate that was previously downloaded to the local keystore.

```
gsk8capicmd_64 -cert -add -db "clientkeydb.p12" -stashed -label "trustedCA" -file "trustedCA.c
```

6. Create a client certificate request.

```
gsk8capicmd_64 -certreq -create -db "clientkeydb.p12" -stashed -label "clientCert"
-dn "CN=db2KeySecureUser,O=IBM,OU=DB2,L=Toronto,ST=Ontario,C=CA" -target "client_cert_req
```

*At your CA, sign the client certificate request:*

7. Sign the client certificate request with the CA certificate, and then download the signed certificate.

*On the Db2 server, add the signed client certificate:*

8. Add the signed client certificate to the local keystore.

```
gsk8capicmd_64 -cert -receive -db "clientkeydb.p12" -stashed
-file "client_cert_signed.arm"
```

## Results

When the Db2 database manager connects to the centralized key manager, SSL communication is used.

## What to do next

Configure the Db2 instance to use the centralized keystore

*Migrating from a local keystore to a centralized keystore:*

If you are currently using Db2 native encryption with master keys stored in a local keystore and you want to start using a centralized keystore instead, you can copy the master keys from your local keystore to the centralized keystore by issuing the **db2p12tokmip** command.

## Before you begin

- Create a centralized keystore configuration file
- Configure SSL between the DB2 instance and the centralized key manager

## Procedure

1. Back up the centralized keystore. See: Backing up IBM Security Key Lifecycle Manager
2. Set the **allow\_key\_insert\_without\_keystore\_backup** parameter to TRUE in the centralized keystore configuration file.
3. Copy all master keys from the local keystore to the centralized keystore by issuing the **db2p12tokmip** command.

### Example

```
db2p12tokmip -from /home/thomas/keystores/ne-keystore.p12
-to /home/thomas/keystores/isklm.cfg
```

To see full syntax information, type: **db2p12tokmip -h** or refer to **db2p12tokmip** command.

4. Set the **allow\_key\_insert\_without\_keystore\_backup** parameter to FALSE in the centralized keystore configuration file.

#### What to do next

1. Configure the DB2 instance to use the centralized keystore
2. Change the master key by running the ADMIN\_ROTATE\_MASTER\_KEY procedure.

#### Setting up a PKCS #11 keystore:

To set up a PKCS #11 keystore for use with Db2 native encryption, begin by creating a PKCS #11 keystore configuration file.

#### Before you begin

1. Install and configure the vendor software that lets you access the PKCS #11 keystore. For a PKCS #11 keystore, you can use one of the following supported key managers:
  - Gemalto Safenet HSM (formerly Luna) version 6.1 (firmware version 6.23.0) and above
  - Thales nShield HSM, security world software version 11.50
2. Check the ability to connect to the PKCS #11 keystore by using vendor utilities. For example:
  - For SafeNet (formerly Luna) hardware security module (HSM), use **vt1 verify**
  - For Thales nShield HSM, use **enquiry**

#### Procedure

1. Create a PKCS #11 keystore configuration file
2. Optional: Create a stash file

#### What to do next

Configure the DB2 instance to use this PKCS #11 keystore to store database master keys for Db2 native encryption.

*Creating a PKCS #11 keystore configuration file:*

To store master keys in a PKCS #11 keystore with Db2 native encryption, you need to create a configuration file that contains details about the PKCS #11 keystore.

#### Procedure

On the Db2 server, create the PKCS #11 keystore configuration file in a text editor.

#### Example

```
VERSION=1
PRODUCT_NAME=Luna
ALLOW_KEY_INSERT_WITHOUT_KEYSTORE_BACKUP=true
LIBRARY=/usr/safenet/lunaclient/luna6.1/lib/libCryptoki2_64.so
SLOT_LABEL=DB2Partition
NEW_OBJECT_TYPE=PRIVATE
KEYSTORE_STASH=/home/userName/sqllib/security/pkcs11_pw.sth
```

#### Keywords

## VERSION

Required. Version of the configuration file. Currently, 1 is the only supported value.

## PRODUCT\_NAME

Optional. Use this value to override the PKCS #11 keystore type that is determined from product information returned by PKCS #11 API calls.. Supported values are:

- Luna for SafeNet (formerly Luna) hardware security module (HSM)
- Thales for Thales nShield HSM
- Other for any other key manager that supports PKCS #11

## ALLOW\_KEY\_INSERT\_WITHOUT\_KEYSTORE\_BACKUP

Optional. Allow the database manager to insert new keys into the centralized key manager. New keys are inserted when the **CREATE DATABASE ENCRYPT** or **ADMIN\_ROTATE\_MASTER\_KEY** commands are run without a specified existing master key label, or when the migration tool **db2p12tokmip** is run. When this parameter is set to TRUE, new keys are allowed to be inserted, if set to FALSE an error is returned if the database manager attempts to insert a new key. You should only set this to TRUE if you are not creating your master keys within the centralized key manager, and you have an automated backup solution of your centralized key manager for newly inserted keys. This parameter must be set to TRUE if you are migrating keys by using the **db2p12tokmip** command. It can be changed to FALSE after the tool has completed. Default value: FALSE.

## LIBRARY

Required. The absolute path and name (including extension) of the PKCS #11 keystore vendor-supplied shared library. The format is platform-dependent:

### Examples for AIX or Linux:

```
/usr/safenet/lunaclient/luna6.1/lib/libCryptoki2_64.so
/opt/nfast/toolkits/pkcs11/libcknfast.so
```

### Examples for Windows:

```
C:\safenet\lunaclient\luna6.1\lib\libCryptoki2_64.dll
C:\nfast\toolkits\pkcs11\libcknfast.dll
```

## SLOT\_LABEL

Optional. Identifies the slot in the HSM by a label. The label is a name that is defined by the application, and is assigned during token initialization. If specified, the value must be 1 - 32 characters long. This parameter cannot be specified if SLOT\_ID is specified.

## SLOT\_ID

Optional. Identifies the slot in the HSM by an ID. Must be an integer value. This parameter cannot be specified if SLOT\_LABEL is specified.

## NEW\_OBJECT\_TYPE

Optional. Defines whether new master keys generated at the PKCS #11 keystore are created as private or public objects. The default value is PRIVATE. The supported values are:

- PRIVATE for private objects

- PUBLIC for public objects

### KEYSTORE\_STASH

Optional. Absolute path and name of the stash file that holds the PKCS #11 keystore password. The instance uses the stash file to authenticate to the PKCS #11 keystore.

### What to do next

Create a stash file, if you choose to store the HSM credentials in a stash file.

*Creating a stash file:*

Create a stash file to address operational concerns that involve access to PKCS #11 keystore credentials.

### Before you begin

- Create a PKCS #11 keystore configuration file

### About this task

A stash file stores the password of a keystore in obfuscated form. The stash file contributes to enhanced operations. If you create a stash file, the database manager can access credentials that it requires to log in to the PKCS #11 keystore. Without a stash file, the only realistic solutions to restart an instance immediately in the event of an unplanned outage are less than ideal:

- Store the credentials in plain form so that an automated script can restart the instance. However, storing the password in plain form is not desirable since it violates security policies and best practices.
- Have a DBA always available to provide the access credentials for the PKCS #11 keystore when the instance restarts. However, having to rely on human intervention, with the expectancy of instant response time, is rarely feasible from an operational perspective.

### Restrictions

The following procedure is intended to be used in a PKCS #11 keystore. Do not attempt to stash a password by using the **gsk8capicmd\_64** command, since that command is intended to be used exclusively with a local keystore. Conversely, do not attempt to stash a password for a local keystore by using the following procedure.

### Procedure

To create a stash file in a PKCS #11 keystore:

1. Run the **db2credman** command to stash the provided password to a file.  

```
db2credman -stash -password Str0ngPassw0rd -to /home/db2inst1/keystore/pkcs11_pw.sth
```
2. Update the PKCS #11 keystore configuration file by adding the **KEYSTORE\_STASH** parameter.  

```
...
KEYSTORE_STASH=/home/db2inst1/keystore/pkcs11_pw.sth
```
3. Run the **db2stop** command to remove the in-memory copy of the password.
4. Run the **db2start** command without the OPEN KEYSTORE USING *password* option.

## Results

The PKCS #11 keystore password is now stored in the stash file, in obfuscated form. The next operation that requires the PKCS #11 keystore password will read it from the stash file.

## What to do next

If you are currently using Db2 native encryption with master keys that are stored in a local keystore and you want to start to use a PKCS #11 keystore instead, Migrate the local keystore to a PKCS #11 keystore.

If you decide to stop using the stash file, in favor of providing PKCS #11 keystore credentials on instance start, follow these steps:

1. Run the **db2start** command with the OPEN KEYSTORE USING *password* option.
2. Update the PKCS #11 keystore configuration file by removing the **KEYSTORE\_STASH** parameter.
3. Delete the stash file to eliminate any potential security risks that this unused file poses.

The next operation that requires the PKCS #11 keystore password will read it from memory.

*Migrating from a local keystore to a PKCS #11 keystore:*

If you are currently using Db2 native encryption with master keys stored in a local keystore and you want to start using a PKCS #11 keystore instead, you can copy the master keys from your local keystore to the PKCS #11 keystore by issuing the **db2p12top11** command.

## Before you begin

- Create a PKCS #11 keystore configuration file

## Procedure

1. Back up the PKCS #11 keystore by using the vendor's key manager software.
2. Set the **ALLOW\_KEY\_INSERT\_WITHOUT\_KEYSTORE\_BACKUP** parameter to **TRUE** in the PKCS #11 keystore keystore configuration file.
3. If you are migrating to a hardware security module (HSM) of the Thales nCipher family, you must assign the **unwrap\_kek** parameter to the **CKNFAST\_OVERRIDE\_SECURITY\_ASSURANCES** environment variable.
4. Copy all master keys from the local keystore to the PKCS #11 keystore by issuing the **db2p12top11** command.

## Example

```
db2p12top11 -to /pkcs11_keystore.cfg -pin Str0ngPassw0rd
```

To see full syntax information, type: **db2p12top11 -h** or refer to **db2p12top11** command.

5. Set the **ALLOW\_KEY\_INSERT\_WITHOUT\_KEYSTORE\_BACKUP** parameter to **FALSE** in the PKCS #11 keystore keystore configuration file.

## What to do next

1. Configure the DB2 instance to use the PKCS #11 keystore.

2. Change the master key by running the `ADMIN_ROTATE_MASTER_KEY` procedure.

### Configuring a Db2 instance to use a keystore:

To configure a Db2 instance to use a keystore for native encryption, you just need to set two database manager configuration parameters: **keystore\_type** and **keystore\_location**.

#### Procedure

- For a local keystore, set **keystore\_type** to "PKCS12", and set **keystore\_location** to the absolute path and file name of the local keystore file.

##### Example

```
update dbm cfg using keystore_location /home/thomas/keystores/ne-keystore.p12
update dbm cfg using keystore_type pkcs12
```

- For a centralized keystore, set **keystore\_type** to "KMIP", and set **keystore\_location** to the absolute path and file name of the centralized keystore configuration file.

##### Example

```
update dbm cfg using keystore_location /home/thomas/keystores/isklm.cfg
update dbm cfg using keystore_type kmip
```

- For a PKCS #11 keystore, set **keystore\_type** to "PKCS11", and set **keystore\_location** to the absolute path and file name of the PKCS #11 keystore configuration file.

##### Example

```
update dbm cfg using keystore_location /home/thomas/keystores/pkcs11.cfg
update dbm cfg using keystore_type pkcs11
```

### What to do next

Restart the database manager instance to cause the configuration changes to take effect.

### Keystore best practices:

Employ security best practices to keep your keystores and master keys secure.

#### Procedure

- Use standard operating system access security protocols to keep your local keystore files and keystore configuration files safe.
- For local keystores and centralized keystores, use strong keystore passwords. The **gsk8capicmd\_64** command has a parameter, `-strong`, that enforces stronger passwords. For more information, see: GSKCapiCmd Users Guide
- Rotate database master keys regularly using the **ADMIN\_ROTATE\_MASTER\_KEY** procedure.
- For local keystores and centralized keystores, change keystore passwords regularly using the **gsk8capicmd\_64** command with the `-changepw` parameter.
- Back up files at regular intervals, and every time the keystore changes (for example, when a key or certificate is added, a master key is rotated, or the password is changed.)
  - Back up local keystore files and keystore configuration files, and store the backed up files in a secure location.

- If you are using a centralized keystore, use built-in features of the key manager to back up the keystore. See: Backing up IBM Security Key Lifecycle Manager
- Do not delete keys from the keystore. Deleting keys from the keystore might result in the inability to access backups or transaction log files. Backups use the master key that is specified during the backup operation and are not affected by the ADMIN\_ROTATE\_MASTER\_KEY procedure. Although the master key is rotated for primary transaction log files following a call to the ADMIN\_ROTATE\_MASTER\_KEY routine, the change is not immediate, and access to the old master key is required for a period.

## Creating an encrypted database

Create an encrypted database by specifying the ENCRYPT option when using the **CREATE DATABASE** command.

### Before you begin

- If you are using a local key manager, configure GSKit, then create the local keystore file and master key.
- If you are using a centralized key manager, configure the centralized key manager and master key. You must also have SSL set up correctly between the centralized key manager and your database server

### Procedure

- To create an encrypted database with the default settings, specify the ENCRYPT option on the **CREATE DATABASE** command:  
db2 create db <encrypted\_database\_name> encrypt
- To create an encrypted database with custom settings, specify other options on the **CREATE DATABASE** command:  
db2 create db <encrypted\_database\_name> encrypt  
cipher aes key length <length\_of\_data\_encryption\_key>  
master key label <master\_key\_label>

Where:

- CIPHER *cipher-name* specifies the encryption algorithm that is to be used for encrypting the database.
- KEY LENGTH *key-length* specifies the length in bits of the data encryption key that is to be used for encrypting the database.
- MASTER KEY LABEL *label-name* specifies a label for the master key that is used to encrypt the database. If you specify this option, the master key must already exist. If you exclude this option, a master key for the database is automatically generated and added to the keystore.

### Results

The information in your database can only be accessed using the appropriate stash file or password.

### Encrypting an existing database:

If you would like to encrypt the data in an existing, unencrypted database, you must create a backup image of the database, drop it, and then restore it into an encrypted database.

## Procedure

To encrypt an existing database:

1. Create a keystore. if you are using a local key manager. If you are using a centralized key manager, ensure you have set up a centralized keystore.
2. Configure the database instance with the new keystore.
3. Generate a backup image of the database you would like to encrypt:  
`db2 backup database <database_name>`
4. Drop the original copy of the database you wanted to encrypt:  
`db2 drop database <database_name>`
5. Restore the backup image into a new encrypted database :  
`db2 restore database <database_name> into <new_database_name> encrypt`

## Results

The new database will contain the same information as the original, except with encrypted data.

### Verifying a database is protected by native encryption:

Verify whether or not your database is encrypted by native encryption by verifying the value of the **Encrypted database db** configuration parameter. You can also use the `ADMIN_GET_ENCRYPTION_INFO` table function if you would like more information on your encryption settings.

## Procedure

To verify that your database has been successfully encrypted by Db2 native encryption, ensure that the value of the **Encrypted database db** configuration parameter value is YES:

```
db2 get db cfg for <example_encrypted_database>
 Encrypted database = YES
```

## Encrypted database backup images

You can create an encrypted backup image of your database, then retrieve it using the **RESTORE DATABASE** or **RECOVER DATABASE** command. **RECOVER DATABASE** runs both the **RESTORE DATABASE** and **ROLLFORWARD DATABASE** command.

### Creating encrypted backup images:

Create an encrypted backup image of your database using the **BACKUP DATABASE** command and specifying which library you would like to use in backup operations.

## Procedure

- If you have the **enclib** value set in the database configuration file, simply run the **BACKUP DATABASE** command:  
`db2 backup database <database_name>`
- If you do not have the **enclib** value set in the database configuration file, specify the encrypt keyword:  
`db2 backup database <database_name> encrypt`



### Verifying the database backup image is encrypted:

By default backups of encrypted databases are also encrypted. However, if you would still like to ensure that your backup image is encrypted, you can run the **db2ckbkp** command and verify that it returns valid values for **compression**.

#### Procedure

To verify that the database backup image is encrypted, run the **db2ckbkp** command:  
`db2ckbkp -h <backup_image> | grep Compression`

If the image has been successfully encrypted by Db2 native encryption, the **compression** parameter should return 2 if the backup image is encrypted, or 3 if it is both encrypted and compressed.

### Restoring an encrypted backup image to a different system:

Retrieve encrypted backup images of your databases on the same system or on a different system.

Restoring encrypted backup images to the same system is the same, whether or not you are using a local or centralized key manager. Simply run the **RESTORE DATABASE** or **RECOVER DATABASE** command.

If you restore an encrypted backup image to a different system with a centralized key manager, simply configure the new system with the centralized key manager. If you are using local key management, you must take into account the security settings of the original and new system.

*Restoring encrypted backup images to the same system:*

You can restore an encrypted database backup image to the same system with the **RESTORE DATABASE** command.

#### Procedure

To restore an encrypted database backup image to the same system:

Run the **RESTORE DATABASE** command:

```
db2 restore database <database_name> ENCLIB 'db2encr.d11'
```

*Restoring an encrypted backup image to a different system with a local key manager:*

If the systems have identical security requirements, you can restore an encrypted backup image by copying the local keystore from one system to another. Otherwise, you must use a new master key for the backup image and new system.

#### Procedure

The procedure depends on the security protocol:

- When the systems implement identical security requirements:
  1. Use a secure copy protocol such as SCP to copy the keystore and its associated stash file from System A to System B. An SCP is available with most secure shell (SSH) implementations.

2. Update the value of the **keystore\_location** database manager configuration parameter to point to the copied keystore on System B.
3. Restore the backup image on System B:  

```
db2 restore database <database_name> encrypt;
```
- When the systems implement different security requirements:
  1. Add a new master key to the keystore.
  2. Generate an encrypted backup on System A:  

```
db2 backup database <database_name>
 encrypt encrlib 'db2encr.d11'
 encropts 'Master Key Label=<label_systemB_admin>'
```
  3. Send the secret key file securely to the System B administrator.
  4. Have the System B administrator add the key to the keystore on System B.
  5. Have the System B administrator restore the backup image on System B. Ensure that the target database is also encrypted by restoring the image into a new database and specifying the ENCRYPT option on the **RESTORE DATABASE** command:  

```
db2 restore database <database_name> encropts 'Master Key Label=<systemB_admin_label>'
 encrypt cipher aes key length <key_length_in_bits>
```

*Restoring an encrypted backup image to a different system with a centralized key manager:*

If you are using a centralized key manager, restore an encrypted backup image on a different system by configuring that system with the centralized key manager, then running the **RESTORE DATABASE** command.

### Procedure

To restore an encrypted backup image from System A to System B:

1. Copy the centralized keystore configuration file securely to System B.
2. Copy the keystore file which stores the SSL certificates securely to System B.
3. Configure System B with the centralized key manager by updating the **keystore\_location** configuration parameter. Also update the SSL\_KEYDB keyword in the centralized keystore configuration file to point to where you copied the keystore file with the SSL certificates. Update SSL\_KEYDB\_STASH as well if you have a stash file.
4. Restore the backup image on System B:  

```
db2 restore database <database_name> encrypt;
```

---

## Auditing DB2 activities

### Introduction to the Db2 audit facility

To manage access to your sensitive data, you can use a variety of authentication and access control mechanisms to establish rules and controls for acceptable data access. But to protect against and discover unknown or unacceptable behaviors you can monitor data access by using the Db2 audit facility.

Successful monitoring of unwanted data access and subsequent analysis can lead to improvements in the control of data access and the ultimate prevention of malicious or careless unauthorized access to data. The monitoring of application and individual user access, including system administration actions, can provide a historical record of activity on your database systems.

The Db2 audit facility generates, and allows you to maintain, an audit trail for a series of predefined database events. The records generated from this facility are kept in an audit log file. The analysis of these records can reveal usage patterns that would identify system misuse. Once identified, actions can be taken to reduce or eliminate such system misuse.

The audit facility provides the ability to audit at both the instance and the individual database level, independently recording all instance and database level activities with separate logs for each. The system administrator (who holds SYSADM authority) can use the **db2audit** tool to configure audit at the instance level as well as to control when such audit information is collected. The system administrator can use the **db2audit** tool to archive both instance and database audit logs as well as to extract audit data from archived logs of either type.

The security administrator (who holds SECADM authority within a database) can use audit policies in conjunction with the SQL statement, **AUDIT**, to configure and control the audit requirements for an individual database. The security administrator can use the following audit routines to perform the specified tasks:

- The **SYSPROC.AUDIT\_ARCHIVE** stored procedure archives audit logs.
- The **SYSPROC.AUDIT\_LIST\_LOGS** table function allows you to locate logs of interest.
- The **SYSPROC.AUDIT\_DELIM\_EXTRACT** stored procedure extracts data into delimited files for analysis.

The security administrator can grant **EXECUTE** privilege on these routines to another user, therefore enabling the security administrator to delegate these tasks, if required.

When working in a partitioned database environment, many of the auditable events occur at the database partition at which the user is connected (the coordinator partition) or at the catalog partition (if they are not the same database partition). The implication of this is that audit records can be generated by more than one database partition. Part of each audit record contains information identifying the coordinator partition and originating partition (the partition where audit record originated).

At the instance level, the audit facility must be stopped and started explicitly by use of the **db2audit start** and **db2audit stop** commands. When you start instance-level auditing, the audit facility uses existing audit configuration information. Since the audit facility is independent of the Db2 database server, it will remain active even if the instance is stopped. In fact, when the instance is stopped, an audit record may be generated in the audit log. To start auditing at the database level, first you need to create an audit policy, then you associate this audit policy with the objects you want to monitor, such as, authorization IDs, database authorities, trusted contexts or particular tables.

## Categories of audit records

There are different categories of audit records that may be generated. In the following description of the categories of events available for auditing, you should notice that following the name of each category is a one-word keyword used to identify the category type. The categories of events available for auditing are:

- **Audit (AUDIT)**. Generates records when audit settings are changed or when the audit log is accessed.

- Authorization Checking (CHECKING). Generates records during authorization checking of attempts to access or manipulate Db2 database objects or functions.
- Object Maintenance (OBJMAINT). Generates records when creating or dropping data objects, and when altering certain objects.
- Security Maintenance (SECMAINT). Generates records when:
  - Granting or revoking object privileges or database authorities
  - Granting or revoking security labels or exemptions
  - Altering the group authorization, role authorization, or override or restrict attributes of an LBAC security policy
  - Granting or revoking the SETSESSIONUSER privilege
  - Modifying any of the SYSADM\_GROUP, SYSCTRL\_GROUP, SYSMANT\_GROUP, or SYSMON\_GROUP configuration parameters.
- System Administration (SYSADMIN). Generates records when operations requiring SYSADM, SYSMANT, or SYSCTRL authority are performed.
- User Validation (VALIDATE). Generates records when authenticating users or retrieving system security information.
- Operation Context (CONTEXT). Generates records to show the operation context when a database operation is performed. This category allows for better interpretation of the audit log file. When used with the log's event correlator field, a group of events can be associated back to a single database operation. For example, a query statement for dynamic queries, a package identifier for static queries, or an indicator of the type of operation being performed, such as CONNECT, can provide needed context when analyzing audit results.

**Note:** The SQL or XQuery statement providing the operation context might be very long and is completely shown within the CONTEXT record. This can make the CONTEXT record very large.

- Execute (EXECUTE). Generates records during the execution of SQL statements.

For any of the categories listed previously, you can audit failures, successes, or both.

Any operations on the database server may generate several records. The actual number of records generated in the audit log depends on the number of categories of events to be recorded as specified by the audit facility configuration. It also depends on whether successes, failures, or both, are audited. For this reason, it is important to be selective of the events to audit.

## Audit policies

The security administrator can use audit policies to configure the audit facility to gather information only about the data and objects that are needed.

The security administrator can create audit policies to control what is audited within an individual database. The following objects can have an audit policy associated with them:

- The whole database  
All auditable events that occur within the database are audited according to the audit policy.
- Tables  
All data manipulation language (DML) and XQUERY access to the table (untyped), MQT (materialized query table), or nickname is audited. Only

EXECUTE category audit events with or without data are generated when the table is accessed even if the policy indicates that other categories should be audited.

- Trusted contexts

All auditable events that happen within a trusted connection defined by the particular trusted context are audited according to the audit policy.

- Authorization IDs representing users, groups, or roles

All auditable events that are initiated by the specified user are audited according to the audit policy.

All auditable events that are initiated by users that are a member of the group or role are audited according to the audit policy. Indirect role membership, such as through other roles or groups, is also included.

You can capture similar data by using the Work Load Management event monitors by defining a work load for a group and capturing the activity details. You should be aware that the mapping to workloads can involve attributes in addition to just the authorization ID, which can cause you to not achieve the wanted granularity in auditing, or if those other attributes are modified, connections may map to different (possibly unmonitored) workloads. The auditing solution provides a guarantee that a user, group or role will be audited.

- Authorities (SYSADM, SECADM, DBADM, SQLADM, WLMADM, ACCESSCTRL, DATAACCESS, SYSCTRL, SYSMANT, SYSMON)

All auditable events that are initiated by a user that holds the specified authority, even if that authority is unnecessary for the event, are audited according to the audit policy.

The security administrator can create multiple audit policies. For example, your company might want a policy for auditing sensitive data and a policy for auditing the activity of users holding DBADM authority. If multiple audit policies are in effect for a statement, all events required to be audited by each of the audit policies are audited (but audited only once). For example, if the database's audit policy requires auditing successful EXECUTE events for a particular table and the user's audit policy requires auditing failures of EXECUTE events for that same table, both successful and failed attempts at accessing that table are audited.

For a specific object, there can only be one audit policy in effect. For example, you cannot have multiple audit policies associated with the same table at the same time.

An audit policy cannot be associated with a view or a typed table. Views that access a table that has an associated audit policy are audited according to the underlying table's policy.

The audit policy that applies to a table does not automatically apply to a MQT based on that table. If you associate an audit policy with a table, associate the same policy with any MQT based on that table.

Auditing performed during a transaction is done based on the audit policies and their associations at the start of the transaction. For example, if the security administrator associates an audit policy with a user and that user is in a transaction at the time, the audit policy does not affect any remaining statements performed within that transaction. Also, changes to an audit policy do not take effect until they are committed. If the security administrator issues an ALTER AUDIT POLICY statement, it does not take effect until the statement is committed.

The security administrator uses the CREATE AUDIT POLICY statement to create an audit policy, and the ALTER AUDIT POLICY statement to modify an audit policy. These statements can specify:

- The status values for events to be audited: None, Success, Failure, or Both.  
Only auditable events that match the specified status value are audited.
- The server behavior when errors occur during auditing.

The security administrator uses the AUDIT statement to associate an audit policy with the current database or with a database object, at the current server. Any time the object is in use, it is audited according to this audit policy.

To delete an audit policy, the security administrator uses the DROP statement. You cannot drop an audit policy if it is associated with any object. Use the AUDIT REMOVE statement to remove any remaining association with an object. To add metadata to an audit policy, the security administrator uses the COMMENT statement.

### Events generated before a full connection has been established

For some events generated during connect and a switch user operation, the only audit policy information available is the policy that is associated with the database. These events are shown in the following table:

*Table 6. Connection events*

Event	Audit category	Comment
CONNECT	CONTEXT	
CONNECT_RESET	CONTEXT	
AUTHENTICATION	VALIDATE	This includes authentication during both connect and switch user within a trusted connection.
CHECKING_FUNC	CHECKING	The access attempted is SWITCH_USER.

These events are audited based only on the audit policy associated with the database and not with audit policies associated with any other object such as a user, their groups, or authorities. For the CONNECT and AUTHENTICATION events that occur during connect, the instance-level audit settings are used until the database is activated. The database is activated either during the first connection or when the ACTIVATE DATABASE command is issued.

### Effect of switching user

If a user is switched within a trusted connection, no remnants of the original user are left behind. In this case, the audit policies associated with the original user are no longer considered, and the applicable audit policies are re-evaluated according to the new user. Any audit policy associated with the trusted connection is still in effect.

If a SET SESSION USER statement is used, only the session authorization ID is switched. The audit policy of the authorization ID of the original user (the system authorization ID) remains in effect and the audit policy of the new user is used as well. If multiple SET SESSION USER statements are issued within a session, only the audit policies associated with the original user (the system authorization ID)

and the current user (the session authorization ID) are considered.

### **Data definition language restrictions**

The following data definition language (DDL) statements are called AUDIT exclusive SQL statements:

- AUDIT
- CREATE AUDIT POLICY, ALTER AUDIT POLICY, and DROP AUDIT POLICY
- DROP ROLE and DROP TRUSTED CONTEXT, if the role or trusted context being dropped is associated with an audit policy

AUDIT exclusive SQL statements have some restrictions in their use:

- Each statement must be followed by a COMMIT or ROLLBACK.
- These statements cannot be issued within a global transaction, for example an XA transaction.

Only one uncommitted AUDIT exclusive DDL statement is allowed at a time across all partitions. If an uncommitted AUDIT exclusive DDL statement is executing, subsequent AUDIT exclusive DDL statements wait until the current AUDIT exclusive DDL statement commits or rolls back.

**Note:** Changes are written to the catalog, but do not take effect until COMMIT, even for the connection that issues the statement.

### **Example of auditing any access to a specific table**

Consider a company where the EMPLOYEE table contains extremely sensitive information and the company wants to audit any and all SQL access to the data in that table. The EXECUTE category can be used to track all access to a table; it audits the SQL statement, and optionally the input data value provided at execution time for that statement.

There are two steps to track activity on the table. First, the security administrator creates an audit policy that specifies the EXECUTE category, and then the security administrator associates that policy with the table:

```
CREATE AUDIT POLICY SENSITIVEDATAPOLICY
 CATEGORIES EXECUTE STATUS BOTH ERROR TYPE AUDIT
COMMIT

AUDIT TABLE EMPLOYEE USING POLICY SENSITIVEDATAPOLICY
COMMIT
```

### **Example of auditing any actions by SYSADM or DBADM**

In order to complete their security compliance certification, a company must show that any and all activities within the database by those people holding system administration (SYSADM) or database administrative (DBADM) authority can be monitored.

To capture all actions within the database, both the EXECUTE and SYSADMIN categories should be audited. The security administrator creates an audit policy that audits these two categories. The security administrator can use the AUDIT statement to associate this audit policy with the SYSADM and DBADM authorities. Any user that holds either SYSADM or DBADM authority will then have any auditable events logged. The following example shows how to create such an audit policy and associate it with the SYSADM and DBADM authorities:

```
CREATE AUDIT POLICY ADMINSPOLICY CATEGORIES EXECUTE STATUS BOTH,
 SYSADMIN STATUS BOTH ERROR TYPE AUDIT
COMMIT
AUDIT SYSADM, DBADM USING POLICY ADMINSPOLICY
COMMIT
```

### Example of auditing any access by a specific role

A company has allowed its web applications access to their corporate database. The exact individuals using the web applications are unknown. Only the role that is used is known and that role is used to manage the database authorizations. The company wants to monitor the actions of anyone who is a member of that role in order to examine the requests they are submitting to the database and to ensure that they only access the database through the web applications.

The EXECUTE category contains the necessary level of auditing to track the activity of the users for this situation. The first step is to create the appropriate audit policy and associate it with the roles that are used by the web applications (in this example, the roles are TELLER and CLERK):

```
CREATE AUDIT POLICY WEBAPPPOLICY CATEGORIES EXECUTE WITH DATA
 STATUS BOTH ERROR TYPE AUDIT
COMMIT
AUDIT ROLE TELLER, ROLE CLERK USING POLICY WEBAPPPOLICY
COMMIT
```

### Example of enabling auditing for a database

A company wants to determine who is making DDL changes (example: ALTER TABLE) on the database named SAMPLE.

```
CONNECT TO SAMPLE
```

```
CREATE AUDIT POLICY ALTPOLICY CATEGORIES AUDIT STATUS BOTH,
 OBJMAINT STATUS BOTH, CHECKING STATUS BOTH,
 EXECUTE STATUS BOTH, ERROR TYPE NORMAL
```

```
AUDIT DATABASE USING POLICY ALTPOLICY
```

### Storage and analysis of audit logs

Archiving the audit log moves the active audit log to an archive directory while the server begins writing to a new, active audit log. Later, you can extract data from the archived log into delimited files and then load data from these files into Db2 database tables for analysis.

Configuring the location of the audit logs allows you to place the audit logs on a large, high-speed disk, with the option of having separate disks for each member in a multiple member database environment, such as a Db2 pureScale environment or a partitioned database environment. In a multiple member database environment, the path for the active audit log can be a directory that is unique to each member. Having a unique directory for each member helps to avoid file contention, because each member is writing to a different disk.

The default path for the audit logs on Windows operating systems is *instance\security\auditdata* and on Linux and UNIX operating systems is *instance/security/auditdata*. If you do not want to use the default location, you can choose different directories (you can create new directories on your system to use as alternative locations, if they do not already exist). To set the path for the



active audit log location and the archived audit log location, use the **db2audit configure** command with the **datapath** and **archivepath** parameters, as shown in this example:

```
db2audit configure datapath /auditlog archivepath /auditarchive
```

The audit log storage locations you set using **db2audit** apply to all databases in the instance.

**Note:** If there are multiple instances on the server, then each instance should have separate data and archive paths.

### The path for active audit logs (datapath) in a multiple member database environment

In a multiple member database environment, the same active audit log location (set by the **datapath** parameter) must be used on each member. There are two ways to accomplish this:

1. Use database member expressions when you specify the **datapath** parameter. Using database member expressions allows the member number to be included in the path of the audit log files and results in a different path on each database member.
2. Use a shared drive that is the same on all members.

You can use database member expressions anywhere within the value you specify for the **datapath** parameter. For example, on a three member system, where the database member number is 10, the following command:

```
db2audit configure datapath '/pathForNode $N'
```

uses the following paths:

- /pathForMember10
- /pathForMember20
- /pathForMember30

**Note:** You cannot use database member expressions to specify the archive log file path (**archivepath** parameter).

### Archiving active audit logs

The system administrator can use the **db2audit** tool to archive both instance and database audit logs as well as to extract audit data from archived logs of either type.

The security administrator, or a user to whom the security administrator has granted EXECUTE privilege on the audit routines, can archive the active audit log by running the SYSPROC.AUDIT\_ARCHIVE stored procedure. To extract data from the log and load it into delimited files, they can use the SYSPROC.AUDIT\_DELIM\_EXTRACT stored procedure.

These are the steps to archive and extract the audit logs using the audit routines:

1. Schedule an application to perform regular archives of the active audit log using the stored procedure SYSPROC.AUDIT\_ARCHIVE.
2. Determine which archived log files are of interest. Use the SYSPROC.AUDIT\_LIST\_LOGS table function to list all of the archived audit logs.

3. Pass the file name as a parameter to the SYSPROC.AUDIT\_DELIM\_EXTRACT stored procedure to extract data from the log and load it into delimited files.
4. Load the audit data into Db2 database tables for analysis.

The archived log files do not need to be immediately loaded into tables for analysis; they can be saved for future analysis. For example, they may only need to be looked at when a corporate audit is taking place.

If a problem occurs during archive, such as running out of disk space in the archive path, or the archive path does not exist, the archive process fails and an interim log file with the file extension `.bk` is generated in the audit log data path, for example, `db2audit.instance.log.0.20070508172043640941.bk`. After the problem is resolved (by allocating sufficient disk space in the archive path, or by creating the archive path) you must move this interim log to the archive path. Then, you can treat it in the same way as a successfully archived log.

### Archiving active audit logs in a multiple member database environment

In a multiple member database environment, if the archive command is issued while the instance is running, the archive process automatically runs on every member. The same timestamp is used in the archived log file name on all members. For example, on a three member system, where the database member number is 10, the following command:

```
db2audit archive to /auditarchive
```

creates the following files:

- `/auditarchive/db2audit.log.10.timestamp`
- `/auditarchive/db2audit.log.20.timestamp`
- `/auditarchive/db2audit.log.30.timestamp`

If the archive command is issued while the instance is not running, you can control on which member the archive is run by one of the following methods:

- Use the **node** option with the **db2audit** command to perform the archive for the current member only.
- Use the **db2\_all** command to run the archive on all members.

For example:

```
db2_all db2audit archive node to /auditarchive
```

This sets the **DB2NODE** environment variable to indicate on which members the command is invoked.

Alternatively, you can issue an individual archive command on each member separately. For example:

- On member 10:  

```
db2audit archive node 10 to /auditarchive
```
- On member 20:  

```
db2audit archive node 20 to /auditarchive
```
- On member 30:  

```
db2audit archive node 30 to /auditarchive
```

**Note:** When the instance is not running, the timestamps in the archived audit log file names are not the same on each member.

**Note:** It is recommended that the archive path is shared across all members, but it is not required.

**Note:** The AUDIT\_DELIM\_EXTRACT stored procedure and AUDIT\_LIST\_LOGS table function can only access the archived log files that are visible from the current (coordinator) member.

### **Example of archiving a log and extracting data to a table**

To ensure their audit data is captured and stored for future use, a company needs to create a new audit log every six hours and archive the current audit log to a WORM drive. The company schedules the following call to the SYSPROC.AUDIT\_ARCHIVE stored procedure to be issued every six hours by the security administrator, or by a user to whom the security administrator has granted EXECUTE privilege on the AUDIT\_ARCHIVE stored procedure. The path to the archived log is the default archive path, /auditarchive, and the archive runs on all members:

```
CALL SYSPROC.AUDIT_ARCHIVE('/auditarchive', -2)
```

As part of their security procedures, the company has identified and defined a number of suspicious behaviors or disallowed activities that it needs to watch for in the audit data. They want to extract all the data from the one or more audit logs, place it in a relational table, and then use SQL queries to look for these activities. The company has decided on appropriate categories to audit and has associated the necessary audit policies with the database or other database objects.

For example, they can call the SYSPROC.AUDIT\_DELIM\_EXTRACT stored procedure to extract the archived audit logs for all categories from all members that were created with a timestamp in April 2006, using the default delimiter:

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT(
 '', '', '/auditarchive', 'db2audit.%.200604%', '')
```

In another example, they can call the SYSPROC.AUDIT\_DELIM\_EXTRACT stored procedure to extract the archived audit records with success events from the EXECUTE category and failure events from the CHECKING category, from a file with the timestamp they are interested in:

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT('', '', '/auditarchive',
 'db2audit.%.20060419034937', 'category
 execute status success, checking status failure);
```

### **Audit log file names:**

The audit log files have names that distinguish whether they are instance-level or database-level logs and which member they originate from in a multiple member database environment, such as a Db2 pureScale environment or a partitioned database environment. Archived audit logs have the timestamp of when the archive command was run appended to their file name.

### **Active audit log file names**

In a multiple member database environment, the path for the active audit log can be a directory that is unique to each member so that each member writes to an individual file. In order to accurately track the origin of audit records, the member number is included as part of the audit log file name. For example, on member 20, the instance level audit log file name is db2audit.instance.log.20. For a database called testdb in this instance, the audit log file is db2audit.db.testdb.log.20.

In a single member database environment the member number is considered to be 0 (zero). In this case, the instance level audit log file name is `db2audit.instance.log.0`. For a database called `testdb` in this instance, the audit log file is `db2audit.db.testdb.log.0`.

### Archived audit log file names

When the active audit log is archived, the current timestamp in the following format is appended to the filename: `YYYYMMDDHHMMSS` (where *YYYY* is the year, *MM* is the month, *DD* is the day, *HH* is the hour, *MM* is the minutes, and *SS* is the seconds).

The file name format for an archive audit log depends on the level of the audit log:

#### instance-level archived audit log

The file name of the instance-level archived audit log is:  
`db2audit.instance.log.member.YYYYMMDDHHMMSS`.

#### database-level archived audit log

The file name of the database-level archived audit log is:  
`db2audit.dbdatabase.log.member.YYYYMMDDHHMMSS`.

In a single member database environment, the value for *member* is 0 (zero).

The timestamp represents the time that the archive command was run, therefore it does not always precisely reflect the time of the last record in the log. The archived audit log file may contain records with timestamps a few seconds later than the timestamp in the log file name because:

- When the archive command is issued, the audit facility waits for the writing of any in-process records to complete before creating the archived log file.
- In a multi-machine environment, the system time on a remote machine may not be synchronized with the machine where the archive command is issued.

In a multiple member database environment, if the server is running when archive is run, the timestamp is consistent across members and reflects the timestamp generated at the member at which the archive was performed.

### Creating tables to hold the Db2 audit data:

Before you can work with audit data in database tables, you need to create the tables to hold the data. You should consider creating these tables in a separate schema to isolate the data in the tables from unauthorized users.

#### Before you begin

- See the `CREATE SCHEMA` statement for the authorities and privileges that you require to create a schema.
- See the `CREATE TABLE` statement for the authorities and privileges that you require to create a table.
- Decide which table space you want to use to hold the tables. (This topic does not describe how to create table spaces.)

**Note:** The format of the tables you need to create to hold the audit data might change from release to release. New columns might be added or the size of an existing column might change. The script, `db2audit.ddl`, creates tables of the correct format to contain the audit records.

## About this task

The examples that follow show how to create the tables to hold the records from the delimited files. If you want, you can create a separate schema to contain these tables.

If you do not want to use all of the data that is contained in the files, you can omit columns from the table definitions, or bypass creating certain tables, as required. If you omit columns from the table definitions, you must modify the commands that you use to load data into these tables.

### Procedure

1. Issue the **db2** command to open a Db2 command window.
2. Optional: Create a schema to hold the tables. For this example, the schema is called AUDIT:

```
CREATE SCHEMA AUDIT
```

3. Optional: If you created the AUDIT schema, switch to the schema before creating any tables:

```
SET CURRENT SCHEMA = 'AUDIT'
```

4. Run the script, `db2audit.ddl`, to create the tables that will contain the audit records.

The script `db2audit.ddl` is located in the `sql11ib/misc` directory (`sql11ib\misc` on Windows). The script assumes that a connection to the database exists and that an 8K table space is available. The command to run the script is: **db2 +o -tf sql11ib/misc/db2audit.ddl** The tables that the script creates are: AUDIT, CHECKING, OBJMAINT, SECMAINT, SYSADMIN, VALIDATE, CONTEXT, and EXECUTE.

5. After you have created the tables, the security administrator can use the `SYSPROC.AUDIT_DELIM_EXTRACT` stored procedure, or the system administrator can use the **db2audit extract** command, to extract the audit records from the archived audit log files into delimited files. You can load the audit data from the delimited files into the database tables you just created.

### Loading Db2 audit data into tables:

After you have archived and extracted the audit log file into delimited files, and you have created the database tables to hold the audit data, you can load the audit data from the delimited files into the database tables for analysis.

## About this task

You use the load utility to load the audit data into the tables. Issue a separate load command for each table. If you omitted one or more columns from the table definitions, you must modify the version of the **LOAD** command that you use to successfully load the data. Also, if you specified a delimiter character other than the default when you extracted the audit data, you must also modify the version of the **LOAD** command that you use.

### Procedure

1. Issue the **db2** command to open a Db2 command window.
2. To load the AUDIT table, issue the following command:

```
LOAD FROM audit.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
INSERT INTO schema.AUDIT
```

**Note:** Specify the **DELPRIORITYCHAR** modifier to ensure proper parsing of binary data.

**Note:** Specify the **LOBSINFILE** option of the **LOAD** command (due to the restriction that any inline data for large objects must be limited to 32K). In some situations, you might also need to use the **LOBS FROM** option.

**Note:** When specifying the file name, use the fully qualified path name. For example, if you have the Db2 database system installed on the C: drive of a Windows operating system, you would specify C:\Program Files\IBM\SQLLIB\instance\security\audit.del as the fully qualified file name for the audit.del file.

3. To load the CHECKING table, issue the following command:  

```
LOAD FROM checking.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
INSERT INTO schema.CHECKING
```
4. To load the OBJMAINT table, issue the following command:  

```
LOAD FROM objmaint.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
INSERT INTO schema.OBJMAINT
```
5. To load the SECMAINT table, issue the following command:  

```
LOAD FROM secmaint.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
INSERT INTO schema.SECMAINT
```
6. To load the SYSADMIN table, issue the following command:  

```
LOAD FROM sysadmin.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
INSERT INTO schema.SYSADMIN
```
7. To load the VALIDATE table, issue the following command:  

```
LOAD FROM validate.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
INSERT INTO schema.VALIDATE
```
8. To load the CONTEXT table, issue the following command:  

```
LOAD FROM context.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
INSERT INTO schema.CONTEXT
```
9. To load the EXECUTE table, issue the following command:  

```
LOAD FROM execute.del OF DEL MODIFIED BY DELPRIORITYCHAR LOBSINFILE
INSERT INTO schema.EXECUTE
```
10. After you finish loading the data into the tables, delete the .del files from the security/auditdata subdirectory of the sqllib directory.
11. When you have loaded the audit data into the tables, you are ready to select data from these tables for analysis.

### What to do next

If you have already populated the tables a first time, and want to do so again, use the **INSERT** option to have the new table data added to the existing table data. If you want to have the records from the previous **db2audit extract** operation removed from the tables, load the tables again using the **REPLACE** option.

### Audit archive and extract stored procedures:

The security administrator can use the SYSPROC.AUDIT\_ARCHIVE stored procedure and table function, the SYSPROC.AUDIT\_DELIM\_EXTRACT stored procedure, and the SYSPROC.AUDIT\_LIST\_LOGS table function to archive audit logs and extract data to delimited files.

The security administrator can delegate use of these routines to another user by granting the user EXECUTE privilege on these routines. Only the security

administrator can grant EXECUTE privilege on these routines. EXECUTE privilege WITH GRANT OPTION cannot be granted for these routines (SQLSTATE 42501).

You must be connected to a database in order to use these stored procedures and table functions to archive or list that database's audit logs.

If you copy the archived files to another database system, and you want to use the stored procedures and table functions to access them, ensure that the database name is the same, or rename the files to include the same database name.

These stored procedures and table functions do not archive or list the instance level audit log. The system administrator must use the **db2audit** command to archive and extract the instance level audit log.

You can use these stored procedures and table functions to perform the following operations:

*Table 7. Audit system stored procedures and table functions*

Stored procedure and table function	Operation	Comments
AUDIT_ARCHIVE	Archives the current audit log.	Takes the archive path as input. If the archive path is not supplied, this stored procedure takes the archive path from the audit configuration file.  The archive is run on each member, and a synchronized timestamp is appended to the name of the audit log file.
AUDIT_LIST_LOGS	Returns a list of the archived audit logs at the specified path, for the current database.	

Table 7. Audit system stored procedures and table functions (continued)

Stored procedure and table function	Operation	Comments
AUDIT_ DELIM_EXTRACT	Extracts data from the binary archived logs and loads it into delimited files.	<p>The extracted audit records are placed in a delimited format suitable for loading into Db2 database tables. The output is placed in separate files, one for each category. In addition, the file <b>auditlobs</b> is created to hold any large objects that are included in the audit data. The file names are:</p> <ul style="list-style-type: none"> <li>• <b>audit.del</b></li> <li>• <b>checking.del</b></li> <li>• <b>objmaint.del</b></li> <li>• <b>secmaint.del</b></li> <li>• <b>sysadmin.del</b></li> <li>• <b>validate.del</b></li> <li>• <b>context.del</b></li> <li>• <b>execute.del</b></li> <li>• <b>auditlobs</b></li> </ul> <p>If the files already exist, the output is appended to them. The <b>auditlobs</b> file is created if the CONTEXT or EXECUTE categories are extracted. Only archived audit logs for the current database can be extracted. Only files that are visible to the coordinator member are extracted.</p> <p>Only the instance owner can delete archived audit logs.</p>

### The EXECUTE category for auditing SQL statements

Use the EXECUTE category to accurately track the SQL statements that are issued by a user. In Version 9.5 and earlier releases, you had to use the CONTEXT category to find this information.

As part of a comprehensive security policy, a company can require the ability to retroactively go back a set number of years and analyze the effects of any particular request against certain tables in their database. To do this, a company must institute a policy of archiving their weekly backups and associated log files such that they can reconstitute the database for any chosen moment in time. Also required, is sufficient database audit information captured about every request made against the database to allow, at any future time, the replay and analysis of any request against the relevant, restored database. This requirement can cover both static and dynamic SQL statements.

This EXECUTE category captures the SQL statement text as well as the compilation environment and other values that are needed to replay the statement at a later date. For example, replaying the statement can show you exactly which rows a



SELECT statement returned. In order to re-run a statement, the database tables must first be restored to their state when the statement was issued.

When you audit using the EXECUTE category, the statement text for both static and dynamic SQL is recorded, as are input parameter markers and host variables. You can configure the EXECUTE category to be audited with or without input values.

**Note:** Global variables are not audited.

The auditing of EXECUTE events takes place at the completion of the event (for SELECT statements this is on cursor close). The status that the event completed with is also stored. Because EXECUTE events are audited at completion, long-running queries do not immediately appear in the audit log.

**Note:** The preparation of a statement is not considered part of the execution. Most authorization checks are performed at prepare time (for example, SELECT privilege). This means that statements that fail during prepare due to authorization errors do not generate EXECUTE events.

Statement Value Index, Statement Value Type and Statement Value Data fields may be repeated for a given execute record. For the report format generated by the extraction, each record lists multiple values. For the delimited file format, multiple rows are used. The first row has an event type of STATEMENT and no values. Following rows have an event type of DATA, with one row for each data value associated with the SQL statement. You can use the event correlator and application ID fields to link STATEMENT and DATA rows together. The columns Statement Text, Statement Isolation Level, and Compilation Environment Description are not present in the DATA events.

The statement text and input data values that are audited are converted into the database code page when they are stored on disk (all audited fields are stored in the database code page). No error is returned if the code page of the input data is not compatible with the database code page; the unconverted data will be logged instead. Because each database has its own audit log, databases having different code pages does not cause a problem.

ROLLBACK and COMMIT are audited when executed by the application, and also when issued implicitly as part of another command, such as BIND.

After an EXECUTE event has been audited due to access to an audited table, all statements that affect which other statements are executed within a unit of work, are audited. These statements are COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT and SAVEPOINT.

### **Savepoint ID field**

You can use the Savepoint ID field to track which statements were affected by a ROLLBACK TO SAVEPOINT statement. An ordinary DML statement (such as SELECT, INSERT, and so on) has the current savepoint ID audited. However, for the ROLLBACK TO SAVEPOINT statement, the savepoint ID that is rolled back to will be audited instead. Therefore, every statement with a savepoint ID greater than or equal to that ID will be rolled back, as demonstrated by the following example. The table shows the sequence of statements run; all events with a Savepoint ID greater than or equal to 2 will be rolled back. Only the value of 3 (from the first INSERT statement) is inserted into the table T1.

*Table 8. Sequence of statements to demonstrate effect of ROLLBACK TO SAVEPOINT statement*

Statement	Savepoint ID
INSERT INTO T1 VALUES (3)	1
SAVEPOINT A	2
INSERT INTO T1 VALUES (5)	2
SAVEPOINT B	3
INSERT INTO T1 VALUES (6)	3
ROLLBACK TO SAVEPOINT A	2
COMMIT	

### WITH DATA option

Not all input values are audited when you specify the WITH DATA option. LOB, LONG, XML and structured type parameters appear as NULL.

Date, time, and timestamp fields are recorded in ISO format.

If WITH DATA is specified in one policy, but WITHOUT DATA is specified in another policy associated with objects involved in the execution of the SQL statement, then WITH DATA takes precedence and data is audited for that particular statement. For example, if the audit policy associated with a user specifies WITHOUT DATA, but the policy associated with a table specifies WITH DATA, when that user accesses that table, the input data used for the statement is audited.

You are not able to determine which rows were modified on a positioned-update or positioned-delete statement. Only the execution of the underlying SELECT statement is logged, not the individual FETCH. It is not possible from the EXECUTE record to determine which row the cursor is on when the statement is issued. When replaying the statement at a later time, it is only possible to issue the SELECT statement to see what range of rows may have been affected.

### Example of replaying past activities

Consider in this example that as part of their comprehensive security policy, a company requires that they retain the ability to retroactively go back up to seven years to analyze the effects of any particular request against certain tables in their database. To do this, they institute a policy of archiving their weekly backups and associated log files such that they can reconstitute the database for any chosen moment in time. They require that the database audit capture sufficient information about every request made against the database to allow the replay and analysis of any request against the relevant, restored database. This requirement covers both static and dynamic SQL statements.

This example shows the audit policy that must be in place at the time the SQL statement is issued, and the steps to archive the audit logs and later to extract and analyze them.

1. Create an audit policy that audits the EXECUTE category and apply this policy to the database:

```
CREATE AUDIT POLICY STATEMENTS CATEGORIES EXECUTE WITH DATA
STATUS BOTH ERROR TYPE AUDIT
COMMIT
```

```
AUDIT DATABASE USING POLICY STATEMENTS
COMMIT
```

2. Regularly archive the audit log to create an archive copy.

The following statement should be run by the security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT\_ARCHIVE stored procedure, on a regular basis, for example, once a week or once a day, depending on the amount of data logged. These archived files can be kept for whatever period is required. The AUDIT\_ARCHIVE procedure is called with two input parameters: the path to the archive directory and -2, to indicate that the archive should be run on all members:

```
CALL SYSPROC.AUDIT_ARCHIVE('/auditarchive', -2)
```

3. The security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT\_LIST\_LOGS table function, uses AUDIT\_LIST\_LOGS to examine all of the available audit logs from April 2006, to determine which logs may contain the necessary data:

```
SELECT FILE FROM TABLE(SYSPROC.AUDIT_LIST_LOGS('/auditarchive'))
AS T WHERE FILE LIKE 'db2audit.dbname.log.0.200604%'
FILE

...
db2audit.dbname.log.0.20060418235612
db2audit.dbname.log.0.20060419234937
db2audit.dbname.log.0.20060420235128
```

4. From this output, the security administrator observes that the necessary logs should be in one file: db2audit.dbname.log.20060419234937. The timestamp shows this file was archived at the end of the day for the day the auditors want to see.

The security administrator, or a user to whom they grant EXECUTE privilege for the SYSPROC.AUDIT\_DELIM\_EXTRACT stored procedure, uses this filename as input to AUDIT\_DELIM\_EXTRACT to extract the audit data into delimited files. The audit data in these files can be loaded into Db2 database tables, where it can be analyzed to find the particular statement the auditors are interested in. Even though the auditors are only interested in a single SQL statement, multiple statements from the unit of work may need to be examined in case they have any impact on the statement of interest.

5. In order to replay the statement, the security administrator must take the following actions:
  - Determine the exact statement to be issued from the audit record.
  - Determine the user who issued the statement from the audit record.
  - Re-create the exact permissions of the user at the time they issued the statement, including any LBAC protection.
  - Reproduce the compilation environment, by using the compilation environment column in the audit record in combination with the SET COMPILATION ENVIRONMENT statement.
  - Restore the database to its exact state at the time the statement was issued.

To avoid disturbing the production system, any restore of the database and replay of the statement should be done on a second database system. The security administrator, running as the user who issued the statement, can reissue the statement as found in the statement text with any input variables that are provided in the statement value data elements.

## Enabling replay of past activities:

As part of a comprehensive security policy, a company can require the ability to retroactively go back a set number of years and analyze the effects of any particular request against certain tables in their database.

### Before you begin

A company must institute a policy of archiving their weekly backups and associated log files such that they can reconstitute the database for any chosen moment in time.

### About this task

To allow, at any future time, the replay and analysis of any request against the relevant, restored database, sufficient database audit information must be captured about every request made against the database. This requirement can cover both static and dynamic SQL statements. The EXECUTE category, when logged WITH DATA contains the necessary information to replay past SQL statements, assuming that the data in the database is restored to the state it was when the statement was issued.

### Restrictions

The following authority and privileges are required:

- SECADM authority is required to create the audit policies,
- EXECUTE privilege is required for the audit routines and procedures.

### Procedure

To enable replay of past activities, as the SECADM:

1. Create an audit policy that audits the EXECUTE category and apply this policy to the database.

```
CREATE AUDIT POLICY STATEMENTS CATEGORIES EXECUTE WITH DATA
STATUS BOTH ERROR TYPE AUDIT
COMMIT
AUDIT DATABASE USING POLICY STATEMENTS
COMMIT
```

2. Regularly archive the audit log to create an archive copy. To archive the audit log, run the following command on a regular basis, specifying the path to the archive directory and -2 to indicate the archive should be run on all members:

```
CALL SYSPROC.AUDIT_ARCHIVE('/auditarchive', -2)
```

3. Check that the audit log files were created. These archived files will then be kept for the number of years specified by the company's business policy. To check the audit log files run:

```
SELECT FILE FROM SESSION.AUDIT_ARCHIVE_RESULTS
```

### Results

Your environment is now set up so data and information is archived to allow future replay of logged database activity.

## Replaying past database activities:

Replaying past database activity is possible if all required data, logs and information is available. This reference topic shows how a SECADM might replay past database activity via example.

### Description

At some point, company auditors might want to analyze the activities of a particular user that occurred in the past. The SECADM can use the backup database images, coupled with the backup logs, and audit logs to reconstitute the database in question and replay the activity the auditors want to analyze. Suppose the activities of a particular user that occurred on April 19, 2006 are in question, the following example shows the flow of how a SECADM would help the auditors carry out their analysis.

### Example

1. The SECADM would issue the AUDIT\_LIST\_LOGS to find all available audit logs from April 2006.  

```
SELECT FILE FROM TABLE(SYSPROC.AUDIT_LIST_LOGS('/auditarchive'))
AS T WHERE FILE LIKE 'db2audit.db.sample.log.0.200604%'
FILENAME

...
db2audit.db.sample.log.0.20060418235612
db2audit.db.sample.log.0.20060419234937
db2audit.db.sample.log.0.20060420235128
```
2. From this output, the SECADM observes that the necessary logs should be in the db2audit.db.sample.log.20060419234937 file. The log was taken at the end of the business day on April 19, 2006.
3. This is used as input to the SYSPROC.AUDIT\_DELIM\_EXTRACT stored procedure. The arguments passed into the procedure are:
  - character delimiter (default),
  - output path,
  - path to the archived audit logs,
  - the filename filter to determine what files are extracted from,
  - the status for each category to be extracted, in this case the only category is EXECUTE.

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT('', '', '/auditarchive',
'db2audit.db.sample.log.0.20060419234937',
'category execute')
```
4. The audit data is now in delimited files. The SECADM will load the audit data from the EXECUTE category into the AUDITDATA.EXECUTE table. The table can be created by executing the following:  

```
db2 CONNECT TO sample
db2 SET CURRENT SCHEMA AUDITDATA
db2 -tvf sqllib/misc/db2audit.ddl
```
5. Next, load the data from execute.del to the AUDITDATA.EXECUTE table. The do this run the following command:  

```
db2 LOAD FROM FILE execute.del OF DEL MODIFIED BY LOBSINFILE
INSERT INTO AUDITDATA.EXECUTE
```
6. The SECADM now has all the audit data in the audit tables located within the AUDITDATA schema. This data can now be analyzed to find the particular statement the auditors are interested in.

**Note:** Even though the auditors are only interested in a single SQL statement, multiple statements from the unit of work may need to be examined in case they have any impact on the statement of interest.

7. In order to replay the statement, the following actions must be taken:
  - The exact statement issued must be determined from the audit record.
  - The user who issued the statement must be determined from the audit record.
  - The exact permissions of the user at the time they issued the statement must be re-created, including any LBAC protection.
  - The compilation environment must be reproduced, by using the compilation environment column in the audit record in combination with the SET COMPILATION ENVIRONMENT statement.
  - The exact state of the database at the time the statement was issued must be re-created.

**Note:** So as not to disturb the production system, any restore of the database and replay of the statement should be done on a secondary database system.

8. The SECADM would need to roll forward to the time the statement will start executing. The statement local start time (local\_start\_time) is part of the EXECUTE audit record. Using the following EXECUTE audit record as an example:

```
timestamp=2006-04-10-13.20.51.029203;
category=EXECUTE;
audit event=STATEMENT;
event correlator=1;
event status=0;
database=SAMPLE;
userid=smith;
authid=SMITH;
session authid=SMITH;
application id=*LOCAL.prodrieg.060410172044;
application name=myapp;
package schema=NULLID;
package name=SQLC2F0A;
package section=201;
uow id=2;
activity id=3;
statement invocation id=0;
statement nesting level=0;
statement text=SELECT * FROM DEPARTMENT WHERE DEPTNO = ? AND DEPTNAME = ?;
statement isolation level=CS;
compilation environment=
 isolation level=CS
 query optimization=5
 degree=1
 sqlrules=DB2
 refresh age=+000000000000000.000000
 schema=SMITH
 maintained table type=SYSTEM
 resolution timestamp=2006-04-10-13.20.51.000000
 federated asynchrony=0;
value index=0;
value type=CHAR;
value data=C01;
value index=1;
value type=VARCHAR;
value index=INFORMATION CENTER;
local_start_time=2006-04-10-13.20.51.021507;
```

The rollforward statement would look like this:

```
ROLLFORWARD DATABASE sample
TO 2006-04-10-13.20.51.021507
USING LOCAL TIME AND COMPLETE
```

9. The compilation environment needs to be set as well. The compilation environment variable can be set by the SET COMPILATION ENVIRONMENT statement. The SECADM, running as the user who issued the statement, can now replay the statement as found in statement text with any input variables that are provided in the statement value data elements. Here is a sample program in C embedded SQL that will set the COMPILATION ENVIRONMENT and replay the SELECT statement the auditors want to analyze:

```
EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
 SQL TYPE IS BLOB(1M) hv_blob;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE c1 CURSOR FOR SELECT COMPENVDESC
 FROM AUDITDATA.EXECUTE TIMESAMP= '2006-04-10-13.20.51.029203';
EXEC SQL DECLARE c2 CURSOR FOR SELECT *
 FROM DEPARTMENT
 WHERE DEPTNO = 'C01'
 AND DEPTNAME = 'INFORMATION CENTER';
EXEC SQL OPEN c1;

EXEC SQL FETCH c1 INTO :hv_blob;

EXEC SQL SET COMPILATION ENVIRONMENT :hv_blob;

EXEC SQL OPEN c2;

....

EXEC SQL CLOSE c1;
EXEC SQL CLOSE c2;
```

## Audit facility management

### Audit facility behavior

This topic provides background information to help you understand how the timing of writing audit records to the log can affect database performance; how to manage errors that occur within the audit facility; and how audit records are generated in different situations.

### Controlling the timing of writing audit records to the active log

The writing of the audit records to the active log can take place synchronously or asynchronously with the occurrence of the events causing the generation of those records. The value of the **audit\_buf\_sz** database manager configuration parameter determines when the writing of audit records is done.

If the value of **audit\_buf\_sz** is zero (0), the writing is done synchronously. The event generating the audit record waits until the record is written to disk. The wait associated with each record causes the performance of the Db2 database to decrease.

If the value of **audit\_buf\_sz** is greater than zero, the record writing is done asynchronously. The value of the **audit\_buf\_sz** when it is greater than zero is the number of 4 KB pages used to create an internal buffer. The internal buffer is used to keep a number of audit records before writing a group of them out to disk. The

statement generating the audit record as a result of an audit event will not wait until the record is written to disk, and can continue its operation.

In the asynchronous case, it could be possible for audit records to remain in an unfilled buffer for some time. To prevent this from happening for an extended period, the database manager forces the writing of the audit records regularly. An authorized user of the audit facility can also flush the audit buffer with an explicit request. Also, the buffers are automatically flushed during an archive operation.

There are differences when an error occurs dependent on whether there is synchronous or asynchronous record writing. In asynchronous mode, there might be some records lost because the audit records are buffered before being written to disk. In synchronous mode, there might be one record lost because the error could only prevent at most one audit record from being written.

### Managing audit facility errors

The setting of the `ERRORTYPE` audit facility parameter controls how errors are managed between the Db2 database system and the audit facility. When the audit facility is active, and the setting of the `ERRORTYPE` audit facility parameter is `AUDIT`, then the audit facility is treated in the same way as any other part of Db2 database. An audit record must be written (to disk in synchronous mode; or to the audit buffer in asynchronous mode) for an audit event associated with a statement to be considered successful. Whenever an error is encountered when running in this mode, a negative `SQLCODE` is returned to the application for the statement generating an audit record.

If the error type is set to `NORMAL`, then any error from `db2audit` is ignored and the operation's `SQLCODE` is returned.

### Audit records generated in different situations

Depending on the API or query statement and the audit settings, none, one, or several audit records might be generated for a particular event. For example, an SQL `UPDATE` statement with a `SELECT` subquery might result in one audit record containing the results of the authorization check for `UPDATE` privilege on a table and another record containing the results of the authorization check for `SELECT` privilege on a table.

For dynamic data manipulation language (DML) statements, audit records are generated for all authorization checking at the time that the statement is prepared. Reuse of those statements by the same user will not be audited again since no authorization checking takes place at that time. However, if a change was made to one of the catalog tables containing privilege information, then in the next unit of work, the statement privileges for the cached dynamic SQL or XQuery statements are checked again and one or more new audit records created.

For a package containing only static DML statements, the only auditable event that could generate an audit record is the authorization check to see if a user has the privilege to execute that package. The authorization checking and possible audit record creation required for the static SQL or XQuery statements in the package is carried out at the time the package is precompiled or bound. The execution of the static SQL or XQuery statements within the package is auditable using the `EXECUTE` category. When a package is bound again either explicitly by the user, or implicitly by the system, audit records are generated for the authorization checks required by the static SQL or XQuery statements.



For statements where authorization checking is performed at statement execution time (for example, data definition language (DDL), GRANT, and REVOKE statements), audit records are generated whenever these statements are used.

**Note:** When executing DDL, the section number recorded for all events (except the context events) in the audit record will be zero (0) no matter what the actual section number of the statement might have been.

## **Audit facility tips and techniques**

Best practices for managing your audit include regularly archiving the audit log, using the error type AUDIT when you create an audit policy, and other tips as described here.

### **Archiving the audit log**

You should archive the audit log on a regular basis. Archiving the audit log moves the current audit log to an archive directory while the server begins writing to a new, active audit log. The name of each archived log file includes a timestamp that helps you identify log files of interest for later analysis.

For long-term storage, you might want to compress groups of archived files.

For archived audit logs that you are no longer interested in, the instance owner can simply delete the files from the operating system.

### **Error handling**

When you create an audit policy, you should use the error type AUDIT, unless you are just creating a test audit policy. For example, if the error type is set to AUDIT, and an error occurs, such as running out of disk space, then an error is returned. The error condition must be corrected before any more auditable actions can continue. However, if the error type was set to NORMAL, the logging would simply fail and no error is returned to the user. Operation continues as if the error did not happen.

If a problem occurs during archive, such as running out of disk space in the archive path, or the archive path does not exist, the archive process fails and an interim log file with the file extension .bk is generated in the audit log data path, for example, db2audit.instance.log.0.20070508172043640941.bk. After the problem is resolved (by allocating sufficient disk space in the archive path, or by creating the archive path) you must move this interim log to the archive path. Then, you can treat it in the same way as a successfully archived log.

### **DDL statement restrictions**

Some data definition language (DDL) statements, called AUDIT exclusive SQL statements, do not take effect until the next unit of work. Therefore, you are advised to use a COMMIT statement immediately after each of these statements.

The AUDIT exclusive SQL statements are:

- AUDIT
- CREATE AUDIT POLICY, ALTER AUDIT POLICY, and DROP AUDIT POLICY
- DROP ROLE and DROP TRUSTED CONTEXT, if the role or trusted context being dropped is associated with an audit policy

## Table format for holding archived data might change

The security administrator can use the SYSPROC.AUDIT\_DEL\_EXTRACT stored procedure, or the system administrator can use the **db2audit extract** command, to extract audit records from the archived audit log files into delimited files. You can load the audit data from the delimited files into Db2 database tables for analysis. The format of the tables you need to create to hold the audit data might change from release to release.

**Important:** The script, `db2audit.ddl`, creates tables of the correct format to contain the audit records. You should expect to run `db2audit.ddl` for each release, as columns might be added or the size of an existing column might change.

## Using CHECKING events

In most cases, when working with CHECKING events, the object type field in the audit record is the object being checked to see if the required privilege or authority is held by the user ID attempting to access the object. For example, if a user attempts to ALTER a table by adding a column, then the CHECKING event audit record indicates the access attempted was "ALTER" and the object type being checked was "TABLE" (not the column, because it is table privileges that are checked).

However, when the checking involves verifying if a database authority exists to allow a user ID to CREATE or BIND an object, or to DROP an object, then although there is a check against the database, the object type field will specify the object being created, bound, or dropped (rather than the database itself).

When creating an index on a table, the privilege to create an index is required, therefore the CHECKING event audit record has an access attempt type of "index" rather than "create".

## Audit records created for binding a package

When binding a package that already exists, then an OBJMAINT event audit record is created for the DROP of the package and then another OBJMAINT event audit record is created for the CREATE of the new copy of the package.

## Using CONTEXT event information after ROLLBACK

Data Definition Language (DDL) might generate OBJMAINT or SECMAINT events that are logged as successful. It is possible however that following the logging of the event, a subsequent error might cause a ROLLBACK to occur. This would leave the object as not created; or the GRANT or REVOKE actions as incomplete. The use of CONTEXT events becomes important in this case. Such CONTEXT event audit records, especially the statement that ends the event, indicates the nature of the completion of the attempted operation.

## The load delimiter

When extracting audit records in a delimited format suitable for loading into a Db2 database table, you should be clear regarding the delimiter used within the statement text field. This can be done when extracting the delimited file, using:

```
db2audit extract delasc delimiter load_delimiter
```

The *load \_delimiter* can be a single character (such as ") or a four-byte string representing a hexadecimal value (such as "0x3b"). Examples of valid commands are:

```
db2audit extract delasc
db2audit extract delasc delimiter !
db2audit extract delasc delimiter 0x3b
```

If you have used anything other than the default load delimiter as the delimiter when extracting, you should use the **MODIFIED BY** option on the **LOAD** command. A partial example of the **LOAD** command with "0x3b" used as the delimiter follows:

```
db2 load from context.del of del modified by chardel0x3b replace into ...
```

This overrides the default load character string delimiter which is " (double quote).

---

## Security model for the db2cluster command

The **db2cluster** command is the main interface into Db2 cluster services, and as such acts on both the cluster manager and shared file system cluster provided for the IBM Db2 pureScale Feature. The **db2cluster** command options that are available to a user depend on the user's authority.

In terms of the security model for the **db2cluster** command, there are three user groups, broken down by the type of tasks each user group is likely to perform:

- Anyone with a userid on the system

Users in this group are able to use the **db2cluster** command to report information about the Db2 pureScale instance, but not to make any changes.

- The SYSADM, SYSCTL or SYSMAINT group

Users in this group are able to use the **db2cluster** command to keep the instance up and running, and to perform some administrative tasks on the cluster manager. By definition, a user in this group is either the userid of the instance, a member of the primary group of the instance owner, or a member of a non-primary group of the instance owner. Db2 recommends that normal day to day activities are performed using a userid with membership in a non-primary group of the instance owner

- The Db2 cluster services administrator

Users in this group have no requirements to access data in the database; this is an administrative role used for:

- installation and configuration of the Db2 cluster services portion of Db2
- maintaining clustered instances in the cluster domain and maintaining the shared file system cluster

The Db2 cluster services administrator role is an end user with access to a root-owned userid for the operating system; for example, an operating system administrator. Db2 cluster services can affect all clustered environments, whether you are using the Db2 pureScale Feature or a partitioned database environment with integrated HA. Therefore, roles such as DBADM, SECADM, SQLADM, WLMADM, EXPLAIN, ACCESSCTRL, and DATAACCESS that act on databases, do not provide the appropriate level of authority for cluster management. The Db2 cluster services administrator can be the same person as someone with a userid in the SYSADM, SYSCTL or SYSMAINT groups.

**Note:** Just because a user has SYSADM privileges, it does not necessarily mean the user has operating system administration privileges.

## Cluster manager tasks for db2cluster

- Anyone with a userid on the system can retrieve information about the current state of the cluster domain using the **-list** and **-verify** options.
- Users in the SYSADM, SYSMAINT or SYSCTL group can query and change the preferred primary cluster caching facility using the **-list** and **-set** options. As well, these users can use the **-clear -alert** option to clear alerts for any of the hosts, members, and cluster caching facilities in the current instance (as defined by the DB2INSTANCE registry variable). Users in this group can also create and delete cluster resources, and repair the cluster manager resource model; however, it is strongly recommended that these tasks be performed only under the advisement of Db2 service personnel.
- The Db2 cluster services administrator can perform administrative tasks that affect Db2 cluster services as a whole across all clustered instances on all hosts in the cluster domain. This user can perform configuration tasks such as setting the tiebreaker device and the host failure detection time, using the **-set** option. As well, the Db2 cluster services administrator can perform maintenance-related tasks, such as putting hosts into maintenance mode, using the **-enter** option, or committing changes or updates to the cluster manager, using the **-commit** option. This user can also perform advanced maintenance operations on the cluster manager peer domain, such as creating, deleting, starting, or stopping the domain, and adding or removing hosts; however, it is strongly recommended that these tasks be performed only under the advisement of Db2 service personnel. Certain DB2 cluster administrative commands require *DB2INSTANCE* environment variable to be set.

## Shared file system tasks for db2cluster

- Anyone with a userid on the system can retrieve information about the current state of the cluster domain using the **-list** and **-verify** options. These users can also perform a wide variety of file system operations with the **db2cluster** command options, but what they can do is constrained by regular file system permissions. As long as the userid running the command has read and write ownership of the device being used, that user can create file systems and add disks. Once a file system has been created or mounted, access to that file system is limited to the userid that created it and to the Db2 cluster services administrator, so only those users can remove, delete, or rebalance a file system. Either the userid that created it, or the Db2 cluster services administrator can create directories that are accessible to other users, much as with a normal file system.
- The Db2 cluster services administrator can perform administrative tasks that affect Db2 cluster services as a whole across all clustered instances on all hosts in the cluster domain. This user can perform change options for the tiebreaker device, using the **-set** option. As well, the Db2 cluster services administrator can perform maintenance-related tasks, such as putting hosts into maintenance mode, using the **-enter** option, or committing changes or updates to the shared file system, using the **-commit** option. This user can also perform advanced maintenance operations on the shared file system cluster, such as creating, deleting, starting, or stopping the domain, and adding or removing hosts; however, it is strongly recommended that these tasks be performed only under the advisement of Db2 service personnel.

---

## Chapter 2. Roles

Roles simplify the administration and management of privileges by offering an equivalent capability as groups but without the same restrictions.

A role is a database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC, or other roles by using a GRANT statement, or can be assigned to a trusted context by using a CREATE TRUSTED CONTEXT or ALTER TRUSTED CONTEXT statement. A role can be specified for the SESSION\_USER ROLE connection attribute in a workload definition.

Roles provide several advantages that make it easier to manage privileges in a database system:

- Security administrators can control access to their databases in a way that mirrors the structure of their organizations (they can create roles in the database that map directly to the job functions in their organizations).
- Users are granted membership in the roles that reflect their job responsibilities. As their job responsibilities change, their membership in roles can be easily granted and revoked.
- The assignment of privileges is simplified. Instead of granting the same set of privileges to each individual user in a particular job function, the administrator can grant this set of privileges to a role representing that job function and then grant that role to each user in that job function.
- A role's privileges can be updated and all users who have been granted that role receive the update; the administrator does not need to update the privileges for every user on an individual basis.
- The privileges and authorities granted to roles are always used when you create views, triggers, materialized query tables (MQTs), static SQL and SQL routines, whereas privileges and authorities granted to groups (directly or indirectly) are not used.

This is because the Db2 database system cannot determine when membership in a group changes, as the group is managed by third-party software (for example, the operating system or an LDAP directory). Because roles are managed inside the database, the Db2 database system can determine when authorization changes and act accordingly. Roles granted to groups are not considered, due to the same reason groups are not considered.

- All the roles assigned to a user are enabled when that user establishes a connection, so all privileges and authorities granted to roles are taken into account when a user connects. Roles cannot be explicitly enabled or disabled.
- The security administrator can delegate management of a role to others.

All Db2 privileges and authorities that can be granted within a database can be granted to a role. For example, a role can be granted any of the following authorities and privileges:

- DBADM, SECADM, DATAACCESS, ACCESSCTRL, SQLADM, WLMADM, LOAD, and IMPLICIT\_SCHEMA database authorities
- CONNECT, CREATETAB, CREATE\_NOT\_FENCED, BINDADD, CREATE\_EXTERNAL\_ROUTINE, or QUIESCE\_CONNECT database authorities
- Any database object privilege (including CONTROL)

A user's roles are automatically enabled and considered for authorization when a user connects to a database; you do not need to activate a role by using the SET ROLE statement. For example, when you create a view, a materialized query table (MQT), a trigger, a package, or an SQL routine, the privileges that you gain through roles apply. However, privileges that you gain through roles granted to groups of which you are a member do not apply.

A role does not have an owner. The security administrator can use the WITH ADMIN OPTION clause of the GRANT statement to delegate management of the role to another user, so that the other user can control the role membership.

## Restrictions

There are a few restrictions in the use of roles:

- A role cannot own database objects.
- Permissions and roles granted to groups are not considered when you create the following database objects:
  - Packages containing static SQL
  - Views
  - Materialized query tables (MQT)
  - Triggers
  - SQL Routines

Only roles granted to the user creating the object or to PUBLIC, directly or indirectly (such as through a role hierarchy), are considered when creating these objects.

---

## Creating and granting membership in roles

The security administrator holds the authority to create, drop, grant, revoke, and comment on a role. The security administrator uses the GRANT (Role) statement to grant membership in a role to an authorization ID and uses the REVOKE (Role) statement to revoke membership in a role from an authorization ID.

The security administrator can delegate the management of membership in a role to an authorization ID by granting the authorization ID membership in the role with the WITH ADMIN OPTION. The WITH ADMIN OPTION clause of the GRANT (Role) statement gives another user the ability to:

- Grant roles to others.
- Revoke roles from others.
- Comment on the role.

The WITH ADMIN OPTION clause does not give the ability to:

- Drop the role.
- Revoke the WITH ADMIN OPTION for a role from an authorization ID.
- Grant WITH ADMIN OPTION to someone else (if you do not hold SECADM authority).

After the security administrator has created a role, the database administrator can use the GRANT statement to assign authorities and privileges to the role. All Db2 privileges and authorities that can be granted within a database can be granted to a role. Instance level authorities, such as SYSADM authority, cannot be assigned to a role.

The security administrator, or any user who the security administrator has granted membership in a role with WITH ADMIN OPTION can use the GRANT (Role) statement to grant membership in that role to other users, groups, PUBLIC or roles. A user may have been granted membership in a role with WITH ADMIN OPTION either directly, or indirectly through PUBLIC, a group or a role.

All the roles assigned to a user are enabled when that user establishes a session. All the privileges and authorities associated with a user's roles are taken into account when the Db2 database system checks for authorization. Some database systems use the SET ROLE statement to activate a particular role. The Db2 database system supports SET ROLE to provide compatibility with other products using the SET ROLE statement. In a Db2 database system, the SET ROLE statement checks whether the session user is a member of the role and returns an error if they are not.

To revoke a user's membership in a role, the security administrator, or a user who holds WITH ADMIN OPTION privilege on the role, uses the REVOKE (Role) statement.

## Example

A role has a certain set of privileges and a user who is granted membership in this role inherits those privileges. This inheritance of privileges eliminates managing individual privileges when reassigning the privileges of one user to another user. The only operations required when using roles is to revoke membership in the role from one user and grant membership in the role to the other user.

For example, the employees BOB and ALICE, working in department DEV, have the privilege to SELECT on the tables SERVER, CLIENT and TOOLS. One day, management decides to move them to a new department, QA, and the database administrator has to revoke their privilege to select on tables SERVER, CLIENT and TOOLS. Department DEV later hires a new employee, TOM, and the database administrator has to grant SELECT privilege on tables SERVER, CLIENT and TOOLS to TOM.

When using roles, the following steps occur:

1. The security administrator creates a role, DEVELOPER:  
`CREATE ROLE DEVELOPER`
2. The database administrator (who holds DBADM authority) grants SELECT on tables SERVER, CLIENT, and TOOLS to role DEVELOPER:  
`GRANT SELECT ON TABLE SERVER TO ROLE DEVELOPER`  
`GRANT SELECT ON TABLE CLIENT TO ROLE DEVELOPER`  
`GRANT SELECT ON TABLE TOOLS TO ROLE DEVELOPER`
3. The security administrator grants the role DEVELOPER to the users in department DEV, BOB and ALICE:  
`GRANT ROLE DEVELOPER TO USER BOB, USER ALICE`
4. When BOB and ALICE leave department DEV, the security administrator revokes the role DEVELOPER from users BOB and ALICE:  
`REVOKE ROLE DEVELOPER FROM USER BOB, USER ALICE`
5. When TOM is hired in department DEV, the security administrator grants the role DEVELOPER to user TOM:  
`GRANT ROLE DEVELOPER TO USER TOM`

---

## Role hierarchies

A role hierarchy is formed when one role is granted membership in another role.

A role *contains* another role when the other role is granted to the first role. The other role inherits all of the privileges of the first role. For example, if the role DOCTOR is granted to the role SURGEON, then SURGEON is said to contain DOCTOR. The role SURGEON inherits all the privileges of role DOCTOR.

Cycles in role hierarchies are not allowed. A *cycle* occurs if a role is granted in circular way such that one role is granted to another role and that other role is granted to the original role. For example, the role DOCTOR is granted to role SURGEON, and then the role SURGEON is granted back to the role DOCTOR. If you create a cycle in a role hierarchy, an error is returned (SQLSTATE 428GF).

### Example of building a role hierarchy

The following example shows how to build a role hierarchy to represent the medical levels in a hospital.

Consider the following roles: DOCTOR, SPECIALIST, and SURGEON. A role hierarchy is built by granting a role to another role, but without creating cycles. The role DOCTOR is granted to role SPECIALIST, and role SPECIALIST is granted to role SURGEON.

Granting role SURGEON to role DOCTOR would create a cycle and is not allowed.

The security administrator runs the following SQL statements to build the role hierarchy:

```
CREATE ROLE DOCTOR
CREATE ROLE SPECIALIST
CREATE ROLE SURGEON

GRANT ROLE DOCTOR TO ROLE SPECIALIST

GRANT ROLE SPECIALIST TO ROLE SURGEON
```

---

## Effect of revoking privileges from roles

When privileges are revoked, this can sometimes cause dependent database objects, such as views, packages or triggers, to become invalid or inoperative.

The following examples show what happens to a database object when some privileges are revoked from an authorization identifier and privileges are held through a role or through different means.

### Example of revoking privileges from roles

1. The security administrator creates the role DEVELOPER and grants the user BOB membership in this role:  

```
CREATE ROLE DEVELOPER
GRANT ROLE DEVELOPER TO USER BOB
```
2. User ALICE creates a table, WORKITEM:  

```
CREATE TABLE WORKITEM (x int)
```
3. The database administrator grants SELECT and INSERT privileges on table WORKITEM to PUBLIC and also to the role DEVELOPER:



```
GRANT SELECT ON TABLE ALICE.WORKITEM TO PUBLIC
GRANT INSERT ON TABLE ALICE.WORKITEM TO PUBLIC
GRANT SELECT ON TABLE ALICE.WORKITEM TO ROLE DEVELOPER
GRANT INSERT ON TABLE ALICE.WORKITEM TO ROLE DEVELOPER
```

4. User BOB creates a view, PROJECT, that uses the table WORKITEM, and a package, PKG1, that depends on the table WORKITEM:  

```
CREATE VIEW PROJECT AS SELECT * FROM ALICE.WORKITEM
PREP emb001.sqc BINDFILE PACKAGE USING PKG1 VERSION 1
```
5. If the database administrator revokes SELECT privilege on table ALICE.WORKITEM from PUBLIC, then the view BOB.PROJECT remains operative and package PKG1 remains valid because the view definer, BOB, still holds the privileges required through his membership in the role DEVELOPER:  

```
REVOKE SELECT ON TABLE ALICE.WORKITEM FROM PUBLIC
```
6. If the database administrator revokes SELECT privilege on table ALICE.WORKITEM from the role DEVELOPER, the view BOB.PROJECT becomes inoperative and package PKG1 becomes invalid because the view and package definer, BOB, does not hold the required privileges through other means:  

```
REVOKE SELECT ON TABLE ALICE.WORKITEM FROM ROLE DEVELOPER
```

## Example of revoking DBADM authority

In this example, the role DEVELOPER holds DBADM authority and is granted to user BOB.

1. The security administrator creates the role DEVELOPER:  

```
CREATE ROLE DEVELOPER
```
2. The system administrator grants DBADM authority to the role DEVELOPER:  

```
GRANT DBADM ON DATABASE TO ROLE DEVELOPER
```
3. The security administrator grants user BOB membership in this role:  

```
GRANT ROLE DEVELOPER TO USER BOB
```
4. User ALICE creates a table, WORKITEM:  

```
CREATE TABLE WORKITEM (x int)
```
5. User BOB creates a view PROJECT that uses table WORKITEM, a package PKG1 that depends on table WORKITEM, and a trigger, TRG1, that also depends on table WORKITEM:  

```
CREATE VIEW PROJECT AS SELECT * FROM ALICE.WORKITEM
PREP emb001.sqc BINDFILE PACKAGE USING PKG1 VERSION 1
CREATE TRIGGER TRG1 AFTER DELETE ON ALICE.WORKITEM
 FOR EACH STATEMENT MODE DB2SQL
 INSERT INTO ALICE.WORKITEM VALUES (1)
```
6. The security administrator revokes the role DEVELOPER from user BOB:  

```
REVOKE ROLE DEVELOPER FROM USER BOB
```

  
 Revoking the role DEVELOPER causes the user BOB to lose DBADM authority because the role that held that authority was revoked. The view, package, and trigger are affected as follows:
  - View BOB.PROJECT is still valid.
  - Package PKG1 becomes invalid.
  - Trigger BOB.TRG1 is still valid.

View BOB.PROJECT and trigger BOB.TRG1 are usable while package PKG1 is not usable. View and trigger objects created by an authorization ID holding DBADM authority are not affected when DBADM authority is lost.

---

## Delegating role maintenance by using the WITH ADMIN OPTION clause

Using the WITH ADMIN OPTION clause of the GRANT (Role) SQL statement, the security administrator can delegate the management and control of membership in a role to someone else.

The WITH ADMIN OPTION clause gives another user the authority to grant membership in the role to other users, to revoke membership in the role from other members of the role, and to comment on a role, but not to drop the role.

The WITH ADMIN OPTION clause does not give another user the authority to grant WITH ADMIN OPTION on a role to another user. It also does not give the authority to revoke WITH ADMIN OPTION for a role from another authorization ID.

### Example demonstrating use of the WITH ADMIN OPTION clause

1. A security administrator creates the role, DEVELOPER, and grants the new role to user BOB using the WITH ADMIN OPTION clause:

```
CREATE ROLE DEVELOPER
GRANT ROLE DEVELOPER TO USER BOB WITH ADMIN OPTION
```

2. User BOB can grant membership in the role to and revoke membership from the role from other users, for example, ALICE:

```
GRANT ROLE DEVELOPER TO USER ALICE
REVOKE ROLE DEVELOPER FROM USER ALICE
```

3. User BOB cannot drop the role or grant WITH ADMIN OPTION to another user (only a security administrator can perform these two operations). These commands issued by BOB will fail:

```
DROP ROLE DEVELOPER - FAILURE!
- only a security administrator is allowed to drop the role
GRANT ROLE DEVELOPER TO USER ALICE WITH ADMIN OPTION - FAILURE!
- only a security administrator can grant WITH ADMIN OPTION
```

4. User BOB cannot revoke role administration privileges (conferred by WITH ADMIN OPTION) from users for role DEVELOPER, because he does not have security administrator (SECADM) authority. When BOB issues the following command, it fails:

```
REVOKE ADMIN OPTION FOR ROLE DEVELOPER FROM USER SANJAY - FAILURE!
```

5. A security administrator is allowed to revoke the role administration privileges for role DEVELOPER (conferred by WITH ADMIN OPTION) from user BOB, and user BOB still has the role DEVELOPER granted:

```
REVOKE ADMIN OPTION FOR ROLE DEVELOPER FROM USER BOB
```

Alternatively, if a security administrator simply revokes the role DEVELOPER from user BOB, then BOB loses all the privileges he received by being a member of the role DEVELOPER and the authority on the role he received through the WITH ADMIN OPTION clause:

```
REVOKE ROLE DEVELOPER FROM USER BOB
```

---

## Roles compared to groups

Privileges and authorities granted to groups are not considered when creating views, materialized query tables (MQTs), SQL routines, triggers, and packages containing static SQL. Avoid this restriction by using roles instead of groups.

Roles allow users to create database objects using their privileges acquired through roles, which are controlled by the Db2 database system. Groups and users are controlled externally from the Db2 database system, for example, by an operating system or an LDAP server.

## Example of replacing the use of groups with roles

This example shows how you can replace groups by using roles.

Assume that there are three groups, DEVELOPER\_G, TESTER\_G and SALES\_G. The users BOB, ALICE, and TOM are members of these groups, as shown in the following table:

Table 9. Example groups and users

Group	Users belonging to this group
DEVELOPER_G	BOB
TESTER_G	ALICE, TOM
SALES_G	ALICE, BOB

1. The security administrator creates the roles DEVELOPER, TESTER, and SALES to be used instead of the groups.

```
CREATE ROLE DEVELOPER
CREATE ROLE TESTER
CREATE ROLE SALES
```

2. The security administrator grants membership in these roles to users (setting the membership of users in groups was the responsibility of the system administrator):

```
GRANT ROLE DEVELOPER TO USER BOB
GRANT ROLE TESTER TO USER ALICE, USER TOM
GRANT ROLE SALES TO USER BOB, USER ALICE
```

3. The database administrator can grant to the roles similar privileges or authorities as were held by the groups, for example:

```
GRANT privilege ON object TO ROLE DEVELOPER
```

The database administrator can then revoke these privileges from the groups, as well as ask the system administrator to remove the groups from the system.

## Example of creating a trigger using privileges acquired through a role

This example shows that user BOB can successfully create a trigger, TRG1, when he holds the necessary privilege through the role DEVELOPER.

1. First, user ALICE creates the table, WORKITEM:

```
CREATE TABLE WORKITEM (x int)
```

2. Then, the privilege to alter ALICE's table is granted to role DEVELOPER by the database administrator.

```
GRANT ALTER ON ALICE.WORKITEM TO ROLE DEVELOPER
```

3. User BOB successfully creates the trigger, TRG1, because he is a member of the role, DEVELOPER.

```
CREATE TRIGGER TRG1 AFTER DELETE ON ALICE.WORKITEM
FOR EACH STATEMENT MODE DB2SQL INSERT INTO ALICE.WORKITEM VALUES (1)
```

## Notes

- Roles that are granted to groups are not considered.

---

## Using roles after migrating from IBM Informix Dynamic Server

If you have migrated from IBM Informix® Dynamic Server to the Db2 database system and are using roles there are a few things you need to be aware of.

The Informix Dynamic Server (IDS) SQL statement, GRANT ROLE, provides the clause WITH GRANT OPTION. The Db2 database system GRANT ROLE statement provides the clause WITH ADMIN OPTION (this conforms to the SQL standard) that provides the same functionality. During an IDS to Db2 database system migration, after the **dbschema** tool generates CREATE ROLE and GRANT ROLE statements, the **dbschema** tool replaces any occurrences of WITH GRANT OPTION with WITH ADMIN OPTION.

In an IDS database system, the SET ROLE statement activates a particular role. The Db2 database system supports the SET ROLE statement, but only to provide compatibility with other products using that SQL statement. The SET ROLE statement checks whether the session user is a member of the role and returns an error if they are not.

### Example dbschema output

Assume that an IDS database contains the roles DEVELOPER, TESTER and SALES. Users BOB, ALICE, and TOM have different roles granted to each of them; the role DEVELOPER is granted to BOB, the role TESTER granted to ALICE, and the roles TESTER and SALES granted to TOM. To migrate to the Db2 database system, use the **dbschema** tool to generate the CREATE ROLE and GRANT ROLE statements for the database:

```
CREATE ROLE DEVELOPER
CREATE ROLE TESTER
CREATE ROLE SALES

GRANT DEVELOPER TO BOB
GRANT TESTER TO ALICE, TOM
GRANT SALES TO TOM
```

You must create the database in the Db2 database system, and then you can run the preceding statements in that database to re-create the roles and assignment of the roles.

---

## Chapter 3. Using trusted contexts and trusted connections

You can establish an explicit trusted connection by making a request within an application when a connection to a Db2 database is established. The security administrator must have previously defined a trusted context, using the CREATE TRUSTED CONTEXT statement, with attributes matching those of the connection you are establishing (see Step 1, later).

### Before you begin

The API you use to request an explicit trusted connection when you establish a connection depends on the type of application you are using (see the table in Step 2).

After you have established an explicit trusted connection, the application can switch the user ID of the connection to a different user ID using the appropriate API for the type of application (see the table in Step 3).

### Procedure

1. The security administrator defines a trusted context in the server by using the CREATE TRUSTED CONTEXT statement. For example:

```
CREATE TRUSTED CONTEXT MYTCX
 BASED UPON CONNECTION USING SYSTEM AUTHID NEWTON
 ATTRIBUTES (ADDRESS '192.0.2.1')
 WITH USE FOR PUBLIC WITHOUT AUTHENTICATION
 ENABLE
```

2. To establish a trusted connection, use one of the following APIs in your application:

Option	Description
Application	API
CLI/ODBC	SQLConnect, SQLSetConnectAttr
XA CLI/ODBC	Xa_open
JAVA	getDB2TrustedPooledConnection, getDB2TrustedXAConnection

3. To switch to a different user, with or without authentication, use one of the following APIs in your application:

Option	Description
Application	API
CLI/ODBC	SQLSetConnectAttr
XA CLI/ODBC	SQLSetConnectAttr
JAVA	getDB2Connection, reuseDB2Connection
.NET	DB2Connection.ConnectionString keywords: TrustedContextSystemUserID and TrustedContextSystemPassword

The switching can be done either with or without authenticating the new user ID, depending on the definition of the trusted context object associated with the

explicit trusted connection. For example, suppose that the security administrator creates the following trusted context object:

```
CREATE TRUSTED CONTEXT CTX1
 BASED UPON CONNECTION USING SYSTEM AUTHID USER1
 ATTRIBUTES (ADDRESS '192.0.2.1')
 WITH USE FOR USER2 WITH AUTHENTICATION,
 USER3 WITHOUT AUTHENTICATION
 ENABLE
```

Further, suppose that an explicit trusted connection is established. A request to switch the user ID on the trusted connection to USER3 without providing authentication information is allowed because USER3 is defined as a user of trusted context CTX1 for whom authentication is not required. However, a request to switch the user ID on the trusted connection to USER2 without providing authentication information will fail because USER2 is defined as a user of trusted context CTX1 for whom authentication information must be provided.

### Example of establishing an explicit trusted connection and switching the user

In the following example, a middle-tier server needs to issue some database requests on behalf of an end-user, but does not have access to the end-user's credentials to establish a database connection on behalf of that end-user.

You can create a trusted context object on the database server that allows the middle-tier server to establish an explicit trusted connection to the database. After establishing an explicit trusted connection, the middle-tier server can switch the current user ID of the connection to a new user ID without the need to authenticate the new user ID at the database server. The following CLI code snippet demonstrates how to establish a trusted connection using the trusted context, MYTCX, defined in Step 1, earlier, and how to switch the user on the trusted connection without authentication.

```
int main(int argc, char *argv[])
{
 SQLHANDLE henv; /* environment handle */
 SQLHANDLE hdbc1; /* connection handle */
 char origUserid[10] = "newton";
 char password[10] = "test";
 char switchUserid[10] = "zurbie";
 char dbName[10] = "testdb";

 // Allocate the handles
 SQLAllocHandle(SQL_HANDLE_ENV, &henv);
 SQLAllocHandle(SQL_HANDLE_DBC, &hdbc1);

 // Set the trusted connection attribute
 SQLSetConnectAttr(hdbc1, SQL_ATTR_USE_TRUSTED_CONTEXT,
 SQL_TRUE, SQL_IS_INTEGER);

 // Establish a trusted connection
 SQLConnect(hdbc1, dbName, SQL_NTS, origUserid, SQL_NTS,
 password, SQL_NTS);

 //Perform some work under user ID "newton"

 // Commit the work
 SQLEndTran(SQL_HANDLE_DBC, hdbc1, SQL_COMMIT);
}
```

```

// Switch the user ID on the trusted connection
SQLSetConnectAttr(hdbc1,
SQL_ATTR_TRUSTED_CONTEXT_USERID, switchUserid,
SQL_IS_POINTER
);

//Perform new work using user ID "zurbie"
.

//Commit the work
SQLEndTranSQL_HANDLE_DBC, hdbc1, SQL_COMMIT);

// Disconnect from database
SQLDisconnect(hdbc1);

return 0;

} /* end of main */

```

## What to do next

### When does the user ID actually get switched?

After the command to switch the user on the trusted connection is issued, the switch user request is not performed until the next statement is sent to the server. This is demonstrated by the following example where the **list applications** command shows the original user ID until the next statement is issued.

1. Establish an explicit trusted connection with USERID1.
2. Issue the switch user command, such as **getDB2Connection** for USERID2.
3. Run db2 list applications. It still shows that USERID1 is connected.
4. Issue a statement on the trusted connection, such as `executeQuery("values current sqlid")`, to perform the switch user request at the server.
5. Run db2 list applications again. It now shows that USERID2 is connected.

---

## Trusted contexts and trusted connections

A trusted context is a database object that defines a trust relationship for a connection between the database and an external entity such as an application server.

The trust relationship is based upon the following set of attributes:

- System authorization ID: Represents the user that establishes a database connection
- IP address (or domain name): Represents the host from which a database connection is established
- Data stream encryption: Represents the encryption setting (if any) for the data communication between the database server and the database client

When a user establishes a database connection, the Db2 database system checks whether the connection matches the definition of a trusted context object in the database. When a match occurs, the database connection is said to be trusted.

A trusted connection cannot be established if the connection is to a local database using inter-process communication (IPC).

A trusted connection allows the initiator of this trusted connection to acquire additional capabilities that may not be available outside the scope of the trusted connection. The additional capabilities vary depending on whether the trusted connection is explicit or implicit.

The initiator of an explicit trusted connection has the ability to:

- Switch the current user ID on the connection to a different user ID with or without authentication
- Acquire additional privileges via the role inheritance feature of trusted contexts

An implicit trusted connection is a trusted connection that is not explicitly requested; the implicit trusted connection results from a normal connection request rather than an explicit trusted connection request. No application code changes are needed to obtain an implicit connection. Also, whether you obtain an implicit trusted connection or not has no effect on the connect return code (when you request an explicit trusted connection, the connect return code indicates whether the request succeeds or not). The initiator of an implicit trusted connection can only acquire additional privileges via the role inheritance feature of trusted contexts; they cannot switch the user ID.

## How using trusted contexts enhances security

The three-tiered application model extends the standard two-tiered client and server model by placing a middle tier between the client application and the database server. It has gained great popularity in recent years particularly with the emergence of web-based technologies and the Java 2 Enterprise Edition (J2EE) platform. An example of a software product that supports the three-tier application model is IBM WebSphere® Application Server (WAS).

In a three-tiered application model, the middle tier is responsible for authenticating the users running the client applications and for managing the interactions with the database server. Traditionally, all the interactions with the database server occur through a database connection established by the middle tier using a combination of a user ID and a credential that identify that middle tier to the database server. This means that the database server uses the database privileges associated with the middle tier's user ID for all authorization checking and auditing that must occur for any database access, including access performed by the middle tier on behalf of a user.

While the three-tiered application model has many benefits, having all interactions with the database server (for example, a user request) occur under the middle tier's authorization ID raises several security concerns, which can be summarized as follows:

- Loss of user identity  
Some enterprises prefer to know the identity of the actual user accessing the database for access control purposes.
- Diminished user accountability  
Accountability through auditing is a basic principle in database security. Not knowing the user's identity makes it difficult to distinguish the transactions performed by the middle tier for its own purpose from those performed by the middle tier on behalf of a user.
- Over granting of privileges to the middle tier's authorization ID



The middle tier's authorization ID must have all the privileges necessary to execute all the requests from all the users. This has the security issue of enabling users who do not need access to certain information to obtain access anyway.

- Weakened security

In addition to the privilege issue raised in the previous point, the current approach requires that the authorization ID used by the middle tier to connect must be granted privileges on all resources that might be accessed by user requests. If that middle-tier authorization ID is ever compromised, then all those resources will be exposed.

- "Spill over" between users of the same connection

Changes by a previous user can affect the current user.

Clearly, there is a need for a mechanism whereby the actual user's identity and database privileges are used for database requests performed by the middle tier on behalf of that user. The most straightforward approach of achieving this goal would be for the middle-tier to establish a new connection using the user's ID and password, and then direct the user's requests through that connection. Although simple, this approach suffers from several drawbacks which include the following:

- Inapplicability for certain middle tiers. Many middle-tier servers do not have the user authentication credentials needed to establish a connection.
- Performance overhead. There is an obvious performance overhead associated with creating a new physical connection and re-authenticating the user at the database server.
- Maintenance overhead. In situations where you are not using a centralized security set up or are not using single sign-on, there is maintenance overhead in having two user definitions (one on the middle tier and one at the server). This requires changing passwords at different places.

The trusted contexts capability addresses this problem. The security administrator can create a trusted context object in the database that defines a trust relationship between the database and the middle-tier. The middle-tier can then establish an explicit trusted connection to the database, which gives the middle tier the ability to switch the current user ID on the connection to a different user ID, with or without authentication. In addition to solving the end-user identity assertion problem, trusted contexts offer another advantage. This is the ability to control when a privilege is made available to a database user. The lack of control on when privileges are available to a user can weaken overall security. For example, privileges may be used for purposes other than they were originally intended. The security administrator can assign one or more privileges to a role and assign that role to a trusted context object. Only trusted database connections (explicit or implicit) that match the definition of that trusted context can take advantage of the privileges associated with that role.

## Enhancing performance

When you use trusted connections, you can maximize performance because of the following advantages:

- No new connection is established when the current user ID of the connection is switched.
- If the trusted context definition does not require authentication of the user ID to switch to, then the overhead associated with authenticating a new user at the database server is not incurred.

## Example of creating a trusted context

Suppose that the security administrator creates the following trusted context object:

```
CREATE TRUSTED CONTEXT CTX1
 BASED UPON CONNECTION USING SYSTEM AUTHID USER2
 ATTRIBUTES (ADDRESS '192.0.2.1')
 DEFAULT ROLE managerRole
 ENABLE
```

If user *user1* requests a trusted connection from IP address 192.0.2.1, the Db2 database system returns a warning (SQLSTATE 01679, SQLCODE +20360) to indicate that a trusted connection could not be established, and that user *user1* simply got a non-trusted connection. However, if user *user2* requests a trusted connection from IP address 192.0.2.1, the request is honored because the connection attributes are satisfied by the trusted context CTX1. Now that user *user2* has established a trusted connection, he or she can now acquire all the privileges and authorities associated with the trusted context role *managerRole*. These privileges and authorities may not be available to user *user2* outside the scope of this trusted connection

---

## Role membership inheritance through a trusted context

The current user of a trusted connection can acquire additional privileges through the automatic inheritance of a role through the trusted context, if this was specified by the security administrator as part of the relevant trusted context definition.

A role can be inherited by all users of the trusted connection by default. The security administrator can also use the trusted context definition to specify a role for specific users to inherit.

The active roles that a session authorization ID can hold while on a trusted connection are:

- The roles of which the session authorization ID is normally considered a member, plus
- Either the trusted context default role or the trusted context user-specific role, if they are defined

### Note:

- If you configure user authentication using a custom security plugin that is built such that the system authorization ID and the session authorization ID produced by this security plug-in upon a successful connection are different from each other, then a trusted context's role cannot be inherited through that connection, even if it is a trusted connection.
- Trusted context privileges acquired through a role are effective only for dynamic DML operations. They are not effective for:
  - DDL operations
  - Non-dynamic SQL (operations involving static SQL statements such as BIND, REBIND, implicit rebind, incremental bind, and so on)

## Acquiring trusted context user-specific privileges

The security administrator can use the trusted context definition to associate roles with a trusted context so that:

- All users of the trusted connection can inherit a specified role by default

- Specific users of the trusted connection can inherit a specified role

When the user on a trusted connection is switched to a new authorization ID and a trusted context user-specific role exists for this new authorization ID, the user-specific role overrides the trusted context default role, if one exists, as demonstrated in the example.

### **Example of creating a trusted context that assigns a default role and a user-specific role**

Suppose that the security administrator creates the following trusted context object:

```
CREATE TRUSTED CONTEXT CTX1
 BASED UPON CONNECTION USING SYSTEM AUTHID USER1
 ATTRIBUTES (ADDRESS '192.0.2.1')
 WITH USE FOR USER2 WITH AUTHENTICATION,
 USER3 WITHOUT AUTHENTICATION
 DEFAULT ROLE AUDITOR
 ENABLE
```

When USER1 establishes a trusted connection, the privileges granted to the role AUDITOR are inherited by this authorization ID. Similarly, these same privileges are also inherited by USER3 when the current authorization ID on the trusted connection is switched to his or her user ID. (If the user ID of the connection is switched to USER2 at some point, then USER2 would also inherit the trusted context default role, AUDITOR.) The security administrator may choose to have USER3 inherit a different role than the trusted context default role. They can do so by assigning a specific role to this user as follows:

```
CREATE TRUSTED CONTEXT CTX1
 BASED UPON CONNECTION USING SYSTEM AUTHID USER1
 ATTRIBUTES (ADDRESS '192.0.2.1')
 WITH USE FOR USER2 WITH AUTHENTICATION,
 USER3 WITHOUT AUTHENTICATION ROLE OTHER_ROLE
 DEFAULT ROLE AUDITOR
 ENABLE
```

When the current user ID on the trusted connection is switched to USER3, this user no longer inherits the trusted context default role. Rather, they inherit the specific role, OTHER\_ROLE, assigned to him or her by the security administrator.

---

## **Rules for switching the user ID on an explicit trusted connection**

On an explicit trusted connection, you can switch the user ID of the connection to a different user ID. Certain rules apply.

1. If the switch request is not made from an explicit trusted connection, and the switch request is sent to the server for processing, the connection is shut down and an error message is returned (SQLSTATE 08001, SQLCODE -30082 with reason code 41).
2. If the switch request is not made on a transaction boundary, the transaction is rolled back, and the switch request is sent to the server for processing, the connection is put into an unconnected state and an error message is returned (SQLSTATE 58009, SQLCODE -30020).
3. If the switch request is made from within a stored procedure, an error message is returned (SQLCODE -30090, reason code 29), indicating this is an illegal operation in this environment. The connection state is maintained and the connection is not placed into an unconnected state. Subsequent requests may be processed.

4. If the switch request is delivered to the server on an instance attach (rather than a database connection), the attachment is shut down and an error message is returned (SQLCODE -30005).
5. If the switch request is made with an authorization ID that is not allowed on the trusted connection, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.
6. If the switch request is made with an authorization ID that is allowed on the trusted connection WITH AUTHENTICATION, but the appropriate authentication token is not provided, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.
7. If the trusted context object associated with the trusted connection is disabled, and a switch request for that trusted connection is made, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.

In this case, the only switch user request that is accepted is one that specifies the user ID that established the trusted connection or the NULL user ID. If a switch to the user ID that established the trusted connection is made, this user ID does not inherit any trusted context role (neither the trusted context default role nor the trusted context user-specific role).

8. If the system authorization ID attribute of the trusted context object associated with the trusted connection is changed, and a switch request for that trusted connection is made, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.

In this case, the only switch user request that is accepted is one that specifies the user ID that established the trusted connection or the NULL user ID. If a switch to the user ID that established the trusted connection is made, this user ID does not inherit any trusted context role (neither the trusted context default role nor the trusted context user-specific role).

9. If the trusted context object associated with the trusted connection is dropped, and a switch request for that trusted connection is made, error (SQLSTATE 42517, SQLCODE -20361) is returned, and the connection is put in an unconnected state.

In this case, the only switch user request that is accepted is one that specifies the user ID that established the trusted connection or the NULL user ID. If a switch to the user ID that established the trusted connection is made, this user ID does not inherit any trusted context role (neither the trusted context default role nor the trusted context user-specific role).

10. If the switch request is made with a user ID allowed on the trusted connection, but that user ID does not hold CONNECT privilege on the database, the connection is put in an unconnected state and an error message is returned (SQLSTATE 08004, SQLCODE -1060).
11. If the trusted context system authorization ID appears in the WITH USE FOR clause, the Db2 database system honors the authentication setting for the system authorization ID on switch user request to switch back to the system authorization ID. If the trusted context system authorization ID does not appear in the WITH USE FOR clause, then a switch user request to switch back to the system authorization ID is always allowed even without authentication.

**Note:** When the connection is put in the unconnected state, the only requests that are accepted and do not result in returning the error "The application state is in error. A database connection does not exist." (SQLCODE -900) are:

- A switch user request

- A COMMIT or ROLLBACK statement
- A DISCONNECT, CONNECT RESET or CONNECT request

**Note:** When the user ID on the trusted connection is switched to a new user ID, all traces of the connection environment under the old user are gone. In other words, the switching of user IDs results in an environment that is identical to a new connection environment. For example, if the old user ID on the connection had any temporary tables or WITH HOLD cursors open, these objects are completely lost when the user ID on that connection is switched to a new user ID.

**Note:** Java trusted connections do not have an unconnected state. If the switch user operation fails, Java will throw an exception and the connection will be disconnected.

---

## Trusted context problem determination

An explicit trusted connection is a connection that is successfully established by a specific, explicit request for a trusted connection. When you request an explicit trusted connection and you do not qualify for one, you get a regular connection and a warning (+20360).

To determine why a user could not establish a trusted connection, the security administrator needs to look at the trusted context definition in the system catalogs and at the connection attributes. In particular, the IP address from which the connection is established, the encryption level of the data stream or network, and the system authorization ID making the connection. The **-application** option of the **db2pd** utility returns this information, as well as the following additional information:

- Connection Trust Type: Indicates whether the connection is trusted or not. When the connection is trusted, this also indicates whether this is an explicit trusted connection or an implicit trusted connection.
- Trusted Context name: The name of the trusted context associated with the trusted connection.
- Role Inherited: The role inherited through the trusted connection.

The following are the most common causes of failing to obtain an explicit trusted connection:

- The client application is not using TCP/IP to communicate with the Db2 server. TCP/IP is the only supported protocol for a client application to communicate with the Db2 server that can be used to establish a trusted connection (explicit or implicit).
- The database server authentication type is set to CLIENT.
- The database server does not have an enabled trusted context object. The definition of the trusted context object must explicitly state ENABLE in order for that trusted context to be considered for matching the attributes of an incoming connection.
- The trusted context objects on the database server do not match the trust attributes that are presented. For example, one of the following situations may apply:
  - The system authorization ID of the connection does not match any trusted context object system authorization ID.
  - The IP address from which the connection originated does not match any IP address in the trusted context object considered for the connection.

- The data stream encryption used by the connection does not match the value of the ENCRYPTION attribute in the trusted context object considered for the connection.

You can use the **db2pd** tool to find out the IP address from which the connection is established, the encryption level of the data stream or network used by the connection, and the system authorization ID making the connection. You can consult the SYSCAT.CONTEXTS and SYSCAT.CONTEXTATTRIBUTES catalog views to find out the definition of a particular trusted context object, such as its system authorization ID, its set of allowed IP addresses and the value of its ENCRYPTION attribute.

The following are the most common causes of a switch user failure:

- The user ID to switch to does not have CONNECT privileges on the database. In this case, SQL1060N is returned.
- The user ID to switch to, or PUBLIC, is not defined in the WITH USE FOR clause of the trusted context object associated with the explicit trusted connection.
- Switching the user is allowed with authentication, but the user presents no credentials or the wrong credentials.
- A switch-user request is not made on a transaction boundary.
- The trusted context that is associated with a trusted connection has been disabled, dropped, or altered. In this case, only switching to the user ID that established the trusted connection is allowed.

---

## Chapter 4. Row and column access control (RCAC) overview

Db2 V10.1 introduces row and column access control (RCAC), as an additional layer of data security. Row and column access control is sometimes referred to as fine-grained access control or FGAC. RCAC controls access to a table at the row level, column level, or both. RCAC can be used to complement the table privileges model.

To comply with various government regulations, you might implement procedures and methods to ensure that information is adequately protected. Individuals in your organization are permitted access to only the subset of data that is required to perform their job tasks. For example, government regulations in your area might state that a doctor is authorized to view the medical records of their own patients, but not of other patients. The same regulations might also state that, unless a patient gives their consent, a healthcare provider is not permitted access to patient personal information, such as the patients home phone number.

You can use row and column access control to ensure that your users have access to only the data that is required for their work. For example, a hospital system running Db2 and RCAC can filter patient information and data to include only that data which a particular doctor requires. Other patients do not exist as far as the doctor is concerned. Similarly, when a patient service representative queries the patient table at the same hospital, they are able to view the patient name and telephone number columns, but the medical history column is masked for them. If data is masked, a NULL, or an alternate value is displayed, instead of the actual medical history.

Row and column access control, or RCAC, has the following advantages:

- No database user is inherently exempted from the row and column access control rules.

Even higher level authorities such as users with DATAACCESS authority are not exempt from these rules. Only users with security administrator (SECADM) authority can manage row and column access controls within a database. Therefore, you can use RCAC to prevent users with DATAACCESS authority from freely accessing all data in a database.

- Table data is protected regardless of how a table is accessed via SQL.

Applications, improvised query tools, and report generation tools are all subject to RCAC rules. The enforcement is data-centric.

- No application changes are required to take advantage of this additional layer of data security.

That is, row and column level access controls are established and defined in a way that is not apparent to existing applications. However, RCAC represents an important shift in paradigm in the sense that it is no longer what is being asked but rather who is asking what. Result sets for the same query change based on the context in which the query was asked and there is no warning or error returned. This behavior is the exact intent of the solution. It means that application designers and DBAs must be conscious that queries do not see the whole picture in terms of the data in the table, unless granted specific permissions to do so.

**Related information:**

## Row and column access control (RCAC) rules

Row and column access control (RCAC) places access control at the table level around the data itself. SQL rules created on rows and columns are the basis of the implementation of this capability.

Row and column access control is an access control model in which a security administrator manages privacy and security policies. RCAC permits all users to access the same table, as opposed to alternative views of a table. RCAC does however, restrict access to the table based upon individual user permissions or rules as specified by a policy associated with the table. There are two sets of rules, one set operates on rows, and the other on columns.

- Row permission
  - A row permission is a database object that expresses a row access control rule for a specific table.
  - A row access control rule is an SQL search condition that describes what set of rows a user has access to.
- Column mask
  - A column mask is a database object that expresses a column access control rule for a specific column in a table.
  - A column access control rule is an SQL CASE expression that describes what column values a user is permitted to see and under what conditions.

## SQL statements for managing RCAC rules

Using the following SQL statements, you can create, alter, and drop RCAC rules.

CREATE PERMISSION

ALTER PERMISSION

CREATE MASK

ALTER MASK

DROP

GRANT (database authorities)

REVOKE (database authorities)

AUDIT

COMMENT

CREATE TABLE

ALTER TABLE

RENAME

CREATE FUNCTION (external scalar)



CREATE FUNCTION (external table)

CREATE FUNCTION (OLE DB external table)

CREATE FUNCTION (SQL scalar, table, or row)

CREATE FUNCTION (sourced or template)

ALTER FUNCTION

CREATE TRIGGER

ALTER TRIGGER

## **Built-in functions for managing RCAC permissions and masks**

Use the following built-in scalar functions to express conditions in your permissions and masks. For example, a user must belong to one or more roles, or to one or more groups to access a particular row.

---

### **Scenario: ExampleHMO using row and column access control**

This scenario presents ExampleHMO, a national organization with a large and active list of patients, as a user of row and column access control. ExampleHMO uses row and column access control to ensure that their database policies reflect government regulatory requirements for privacy and security, as well as management business objectives.

Organizations that handle patient health information and their personal information, like ExampleHMO, must comply with government privacy and data protection regulations, for example the Health Insurance Portability and Accountability Act (HIPAA). These privacy and data protection regulations ensure that any sensitive patient medical or personal information is shared, viewed, and modified only by authorities who are privileged to do so. Any violation of the act results in huge penalties including civil and criminal suits.

ExampleHMO must ensure that the data stored in their database systems is secure and only privileged users have access to the data. According to typical privacy regulations, certain patient information can be accessed and modified by only privileged users.

### **Scenario: ExampleHMO using row and column access control - Security policies**

ExampleHMO implements a security strategy where data access to databases are made available according to certain security policies.

The security policies conform to government privacy and data protection regulations. The first column outlines the policies and the challenges faced by the organization, the second column outlines the row and column access control feature which addresses the challenge.

Security challenge	Row and column access control feature which addresses the security challenge
Limiting column access to only privileged users.  For example, Jane, who is a drug researcher at a partner company, is not permitted to view sensitive patient medical information or personal data like their insurance number.	Column masks can be used to filter or hide sensitive data from Jane.
Limiting row access to only privileged users. Dr. Lee is only permitted to view patient information for his own patients, not all patients in the ExampleHMO system.	Row permissions can be implemented to control which user can view any particular row.
Restricting data on a need-to-know basis.	Row permissions can help with this challenge as well by restricting table level data at the user level.
Restricting other database objects like UDFs, triggers, views on RCAC secured data.	Row and column access control protects data at the data level. It is this data-centric nature of the row and column access control solution that enforces security policies on even database objects like UDFs, triggers, and views.

## Scenario: ExampleHMO using row and column access control - Database users and roles

In this scenario, a number of different people create, secure, and use ExampleHMO data. These people have different user rights and database authorities.

ExampleHMO implemented their security strategy to classify the way data is accessed from the database. Internal and external access to data is based on the separation of duties to users who access the data and their data access privileges. ExampleHMO created the following database roles to separate these duties:

### **PCP**

For primary care physicians.

### **DRUG\_RESEARCH**

For researchers.

### **ACCOUNTING**

For accountants.

### **MEMBERSHIP**

For members who add patients for opt-in and opt-out.

### **PATIENT**

For patients.

The following people create, secure, and use ExampleHMO data:

### **Alex**

ExampleHMO Chief Security Administrator. He holds the SECADM authority.

### **Peter**

ExampleHMO Database Administrator. He holds the DBADM authority.

**Paul**

ExampleHMO Database Developer. He has the privileges to create triggers and user-defined functions.

**Dr. Lee**

ExampleHMO Physician. He belongs to the PCP role.

**Jane**

Drug researcher at Innovative Pharmaceutical Company, a ExampleHMO partner. She belongs to the DRUG\_RESEARCH role.

**John**

ExampleHMO Accounting Department. He belongs to the ACCOUNTING role.

**Tom**

ExampleHMO Membership Officer. He belongs to the MEMBERSHIP role.

**Bob**

ExampleHMO Patient. He belongs to the PATIENT role.

If you want to try any of the example SQL statements and commands presented in this scenario, create these user IDs with their listed authorities.

The following example SQL statements assume that the users have been created on the system. The SQL statements create each role and grant **SELECT** and **INSERT** permissions to the various tables in the ExampleHMO database to the users:

--Creating roles and granting authority

```
CREATE ROLE PCP;
```

```
CREATE ROLE DRUG_RESEARCH;
```

```
CREATE ROLE ACCOUNTING;
```

```
CREATE ROLE MEMBERSHIP;
```

```
CREATE ROLE PATIENT;
```

```
GRANT ROLE PCP TO USER LEE;
```

```
GRANT ROLE DRUG_RESEARCH TO USER JANE;
```

```
GRANT ROLE ACCOUNTING TO USER JOHN;
```

```
GRANT ROLE MEMBERSHIP TO USER TOM;
```

```
GRANT ROLE PATIENT TO USER BOB;
```

## **Scenario: ExampleHMO using row and column access control - Database tables**

This scenario focuses on two tables in the ExampleHMO database: the PATIENT table and the PATIENTCHOICE table.

The PATIENT table stores basic patient information and health information. This scenario considers the following columns within the PATIENT table:

**SSN**

The patient's insurance number. A patient's insurance number is considered personal information.

**NAME**

The patient's name. A patient's name is considered personal information.

**ADDRESS**

The patient's address. A patient's address is considered personal information.

**USERID**

The patient's database ID.

**PHARMACY**

The patient's medical information.

**ACCT\_BALANCE**

The patient's billing information.

**PCP\_ID**

The patient's primary care physician database ID

The PATIENTCHOICE table stores individual patient opt-in and opt-out information which decides whether a patient wants to expose his health information to outsiders for research purposes in this table. This scenario considers the following columns within the PATIENTCHOICE table:

**SSN**

The patient's insurance number is used to match patients with their choices.

**CHOICE**

The name of a choice a patient can make.

**VALUE**

The decision made by the patients about the choice.

For example, the row 123-45-6789, drug\_research, opt-in says that patient with SSN 123-45-6789 agrees to disclose their information for medical research purposes.

The following example SQL statements create the PATIENT, PATIENTCHOICE, and ACCT\_HISTORY tables. Authority is granted on the tables and data is inserted:

```
--Patient table storing information regarding patient
CREATE TABLE PATIENT (
 SSN CHAR(11),
 USERID VARCHAR(18),
 NAME VARCHAR(128),
 ADDRESS VARCHAR(128),
 PHARMACY VARCHAR(250),
 ACCT_BALANCE DECIMAL(12,2) WITH DEFAULT,
 PCP_ID VARCHAR(18)
);
```

```
--Patientchoice table which stores what patient opts
--to expose regarding his health information
```

```
CREATE TABLE PATIENTCHOICE (
 SSN CHAR(11),
 CHOICE VARCHAR(128),
 VALUE VARCHAR(128)
);
```

```
--Log table to track account balance
CREATE TABLE ACCT_HISTORY(
 SSN CHAR(11),
 BEFORE_BALANCE DECIMAL(12,2),
 AFTER_BALANCE DECIMAL(12,2),
 WHEN_DATE,
 BY_WHO VARCHAR(20)
);
```

```
--Grant authority
```

```

GRANT SELECT, UPDATE ON TABLE PATIENT TO ROLE PCP;

GRANT SELECT ON TABLE PATIENT TO ROLE DRUG_RESEARCH;

GRANT SELECT, UPDATE ON TABLE PATIENT TO ROLE ACCOUNTING;
GRANT SELECT ON TABLE ACCT_HISTORY TO ROLE ACCOUNTING;

GRANT SELECT, UPDATE, INSERT ON TABLE PATIENT TO ROLE MEMBERSHIP;
GRANT INSERT ON TABLE PATIENTCHOICE TO ROLE MEMBERSHIP;

GRANT SELECT ON TABLE PATIENT TO ROLE PATIENT;

GRANT SELECT, ALTER ON TABLE PATIENT TO USER ALEX;

GRANT ALTER, SELECT ON TABLE PATIENT TO USER PAUL;
GRANT INSERT ON TABLE ACCT_HISTORY TO USER PAUL;

--Insert patient data

INSERT INTO PATIENT
VALUES('123-55-1234', 'MAX', 'Max', 'First Strt', 'hypertension', 89.70, 'LEE');
INSERT INTO PATIENTCHOICE
VALUES('123-55-1234', 'drug-research', 'opt-out');

INSERT INTO PATIENT
VALUES('123-58-9812', 'MIKE', 'Mike', 'Long Strt', null, 8.30, 'JAMES');
INSERT INTO PATIENTCHOICE
VALUES('123-58-9812', 'drug-research', 'opt-out');

INSERT INTO PATIENT
VALUES('123-11-9856', 'SAM', 'Sam', 'Big Strt', null, 0.00, 'LEE');
INSERT INTO PATIENTCHOICE
VALUES('123-11-9856', 'drug-research', 'opt-in');

INSERT INTO PATIENT
VALUES('123-19-1454', 'DUG', 'Dug', 'Good Strt', null, 0.00, 'JAMES');
INSERT INTO PATIENTCHOICE
VALUES('123-19-1454', 'drug-research', 'opt-in');

```

## **Scenario: ExampleHMO using row and column access control** **- Security administration**

Security administration and the security administrator (SECADM) role play important parts in securing patient and company data at ExampleHMO. At ExampleHMO, management decided that different people hold database administration authority and security administration authority.

The management team at ExampleHMO decides to create a role for administering access to their data. The team also decides that even users with DATAACCESS authority are not able to view protected health and personal data by default.

The management team selects Alex to be the sole security administrator for ExampleHMO. From now on, Alex controls all data access authority. With this authority, Alex defines security rules such as row permissions, column masks, and whether functions and triggers are secure or not. These rules control which users have access to any given data under his control.

After Peter, the database administrator, creates the required tables and sets up the required roles, duties are separated. The database administration and security administration duties are separated by making Alex the security administrator.

Peter connects to the database and grants Alex SECADM authority. Peter can grant SECADM authority since he currently holds the DBADM, DATAACCESS, and SECADM authorities.

```
-- To separate duties of security administrator from system administrator,
-- the SECADMN Peter grants SECADM authority to user Alex.
```

```
GRANT SECADM ON DATABASE TO USER ALEX;
```

Alex, after receiving the SECADM authority, connects to the database and revokes the security administrator privilege from Peter. The duties are now separated and Alex becomes the sole authority to grant data access to others within and outside ExampleHMO. The following SQL statement shows how Alex revoked SECADM authority from Peter:

```
--revokes the SECADMIN authority for Peter
```

```
REVOKE SECADM ON DATABASE FROM USER PETER;
```

## Scenario: ExampleHMO using row and column access control

### - Row permissions

Alex, the security administrator, starts to restrict data access on the ExampleHMO database by using row permissions, a part of row and column access control. Row permissions filter the data returned to users by row.

Patients are permitted to view their own data. A physician is permitted to view the data of all his patients, but not the data of patients who see other physicians. Users belonging to the MEMBERSHIP, ACCOUNTING, or DRUG\_RESEARCH roles can access all patient information. Alex, the security administrator, is asked to implement these permissions to restrict who can see any given row on a need-to-know basis.

Row permissions restrict or filter rows based on the user who has logged on to the database. At ExampleHMO, the row permissions create a horizontal data restriction on the table named PATIENT.

Alex implements the following row permissions so that a user in each role is restricted to view a result set that they are privileged to view:

```
CREATE PERMISSION ROW_ACCESS ON PATIENT

-- Accounting information:
-- ROLE PATIENT is allowed to access his or her own row
-- ROLE PCP is allowed to access his or her patients' rows
-- ROLE MEMBERSHIP, ACCOUNTING, and DRUG_RESEARCH are
-- allowed to access all rows

FOR ROWS WHERE(VERIFY_ROLE_FOR_USER(SESSION_USER,'PATIENT') = 1
AND
PATIENT.USERID = SESSION_USER) OR
(VERIFY_ROLE_FOR_USER(SESSION_USER,'PCP') = 1
AND
PATIENT.PCP_ID = SESSION_USER) OR
(VERIFY_ROLE_FOR_USER(SESSION_USER,'MEMBERSHIP') = 1 OR
```

```

VERIFY_ROLE_FOR_USER(SESSION_USER, 'ACCOUNTING') = 1 OR
VERIFY_ROLE_FOR_USER(SESSION_USER, 'DRUG_RESEARCH') = 1)
ENFORCED FOR ALL ACCESS
ENABLE;

```

Alex observes that even after creating a row permission, all data can still be viewed by the other employees. A row permission is not applied until it is activated on the table for which it was defined. Alex must now activate the permission:

```

--Activate row access control to implement row permissions

ALTER TABLE PATIENT ACTIVATE ROW ACCESS CONTROL;

```

## Scenario: ExampleHMO using row and column access control

### - Column masks

Alex, the security administrator, further restricts data access on the ExampleHMO database by using column masks, a part of row and column access control. Column masks hide data returned to users by column unless they are permitted to view the data.

Patient payment details must only be accessible to the users in the accounts department. The account balance must not be seen by any other database users. Alex is asked to prevent access by anyone other than users belonging to the ACCOUNTING role.

Alex implements the following column mask so that a user in each role is restricted to view a result set that they are privileged to view:

```

--Create a Column MASK ON ACCT_BALANCE column on the PATIENT table

```

```

CREATE MASK ACCT_BALANCE_MASK ON PATIENT FOR

-- Accounting information:
-- Role ACCOUNTING is allowed to access the full information
-- on column ACCT_BALANCE.
-- Other roles accessing this column will strictly view a
-- zero value.

COLUMN ACCT_BALANCE RETURN
CASE WHEN VERIFY_ROLE_FOR_USER(SESSION_USER, 'ACCOUNTING') = 1
THEN ACCT_BALANCE
ELSE 0.00
END
ENABLE;

```

Alex observes that even after creating a column mask, the data can still be viewed by the other employees. A column mask is not applied until it is activated on the table for which it was defined. Alex must now activate the mask:

```

--Activate column access control to implement column masks

ALTER TABLE PATIENT ACTIVATE COLUMN ACCESS CONTROL;

```

Alex is asked by management to hide the insurance number of the patients. Only a patient, physician, accountant, or people in the MEMBERSHIP role can view the SSN column.

Also, to protect the PHARMACY detail of a patient, the information in the PHARMACY column must only be viewed by a drug researcher or a physician. Drug researchers can see the data only if the patient has agreed to disclose the information.

Alex implements the following column masks so that a user in each role is restricted to view a result set that they are privileged to view:

```
CREATE MASK SSN_MASK ON PATIENT FOR

-- Personal contact information:
-- Roles PATIENT, PCP, MEMBERSHIP, and ACCOUNTING are allowed
-- to access the full information on columns SSN, USERID, NAME,
-- and ADDRESS. Other roles accessing these columns will
-- strictly view a masked value.

COLUMN SSN RETURN
CASE WHEN
 VERIFY_ROLE_FOR_USER(SESSION_USER,'PATIENT') = 1 OR
 VERIFY_ROLE_FOR_USER(SESSION_USER,'PCP') = 1 OR
 VERIFY_ROLE_FOR_USER(SESSION_USER,'MEMBERSHIP') = 1 OR
 VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1
THEN SSN
ELSE CHAR('XXX-XX-' || SUBSTR(SSN,8,4)) END
ENABLE;

CREATE MASK PHARMACY_MASK ON PATIENT FOR

-- Medical information:
-- Role PCP is allowed to access the full information on
-- column PHARMACY.
-- For the purposes of drug research, Role DRUG_RESEARCH can
-- conditionally see a patient's medical information
-- provided that the patient has opted-in.
-- In all other cases, null values are rendered as column
-- values.

COLUMN PHARMACY RETURN
CASE WHEN
 VERIFY_ROLE_FOR_USER(SESSION_USER,'PCP') = 1 OR
 (VERIFY_ROLE_FOR_USER(SESSION_USER,'DRUG_RESEARCH')=1
 AND
 EXISTS (SELECT 1 FROM PATIENTCHOICE C
 WHERE PATIENT.SSN = C.SSN AND C.CHOICE = 'drug-research' AND C.VALUE = 'opt-in'))
THEN PHARMACY
ELSE NULL
END
ENABLE;
```

Alex observes that after creating these two column masks that the data is only viewable to the intended users. The PATIENT table already had column access control activated.

## Scenario: ExampleHMO using row and column access control - Data insertion

When a new patient is admitted for treatment in the hospital, the new patient record must be added to the ExampleHMO database.

Bob is a new patient, and his records must be added to the ExampleHMO database. A user with the required security authority must create the new record for Bob. Tom, from the ExampleHMO membership department, with the



MEMBERSHIP role, enrolls Bob as a new member. After connecting to the ExampleHMO database, Tom runs the following SQL statements to add Bob to the ExampleHMO database:

```
INSERT INTO PATIENT
VALUES('123-45-6789', 'BOB', 'Bob', '123 Some St.', 'hypertension', 9.00, 'LEE');
INSERT INTO PATIENTCHOICE
VALUES('123-45-6789', 'drug-research', 'opt-in');
```

Tom confirmed that Bob was added to the database by querying the same from the PATIENT table in the ExampleHMO database:

```
Select * FROM PATIENT WHERE NAME = 'Bob';
```

SSN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
123-45-6789	BOB	Bob	123 Some St.	XXXXXXXXXX	0.00	LEE

## Scenario: ExampleHMO using row and column access control

### - Data updates

While in the hospital, Bob gets his treatment changed. As a result his records in the ExampleHMO database need updating.

Dr. Lee, who is Bob's physician, advises a treatment change and changes Bob's medicine. Bob's record in the ExampleHMO systems must be updated. The row permission rules set in the ExampleHMO database specify that anyone who cannot view the data in a row cannot update the data in that row. Since Bob's PCPID contains Dr. Lee's ID, and the row permission is set, Dr. Lee can both view, and update Bob's record using the following example SQL statement:

```
UPDATE PATIENT SET PHARMACY = 'codeine' WHERE NAME = 'Bob';
```

Dr. Lee checks the update:

```
Select * FROM PATIENT WHERE NAME = 'Bob';
```

SSN	USERID	NAME	ADDRESS	PHARMACY	ACCT_BALANCE	PCP_ID
123-45-6789	BOB	Bob	123 Some St.	codeine	0.00	LEE

Dug is a patient who is under the care of Dr. James, one of Dr. Lee's colleagues. Dr. Lee attempts the same update on the record for Dug:

```
UPDATE PATIENT SET PHARMACY = 'codeine' WHERE NAME = 'Dug';
SQL0100W No row was found for FETCH, UPDATE or DELETE; or the result of a query
is an empty table. SQLSTATE=02000
```

Since Dug's PCPID does not contain Dr. Lee's ID, and the row permission is set, Dr. Lee cannot view, or update Dug's record.

## Scenario: ExampleHMO using row and column access control

### - Data queries

With row and column access control, people in different roles can have different result sets from the same database queries. For example, Peter, the database administrator with DATAACCESS authority, cannot see any data on the PATIENT table.

Peter, Bob, Dr. Lee, Tom, Jane, and John each connect to the database and try the following SQL query:

```
SELECT SSN, USERID, NAME, ADDRESS, PHARMACY, ACCT_BALANCE, PCP_ID FROM PATIENT;
```

Results of the query vary according to who runs the query. The row and column access control rules created by Alex are applied on these queries.

Here is the result set Peter sees:

SSN	USERID	NAME	ADDRESS	PHARMACY	ACC_BALANCE	PCP_ID
-----						
0 record(s) selected.						

Even though there is data in the table and Peter is the database administrator, he lacks the authority to see all data.

Here is the result set Bob sees:

SSN	USERID	NAME	ADDRESS	PHARMACY	ACC_BALANCE	PCP_ID
-----						
123-45-6789	BOB	Bob	123 Some St.	XXXXXXXXXX	0.00	LEE
1 record(s) selected.						

Bob, being a patient, can only see his own data. Bob belongs to the PATIENT role. The PHARMACY and ACC\_BALANCE column data have been hidden from him.

Here is the result set Dr. Lee sees:

SSN	USERID	NAME	ADDRESS	PHARMACY	ACC_BALANCE	PCP_ID
-----						
123-55-1234	MAX	Max	First Strt	hypertension	0.00	LEE
123-11-9856	SAM	Sam	Big Strt	High blood pressure	0.00	LEE
123-45-6789	BOB	Bob	123 Some St.	codeine	0.00	LEE
3 record(s) selected.						

Dr. Lee can see only the data for patients under his care. Dr. Lee belongs to the PCP role. The ACC\_BALANCE column data is hidden from him.

Here is the result set Tom sees:

SSN	USERID	NAME	ADDRESS	PHARMACY	ACC_BALANCE	PCP_ID
-----						
123-55-1234	MAX	Max	First Strt	XXXXXXXXXX	0.00	LEE
123-58-9812	MIKE	Mike	Long Strt	XXXXXXXXXX	0.00	JAMES
123-11-9856	SAM	Sam	Big Strt	XXXXXXXXXX	0.00	LEE
123-19-1454	DUG	Dug	Good Strt	XXXXXXXXXX	0.00	JAMES
123-45-6789	BOB	Bob	123 Some St.	XXXXXXXXXX	0.00	LEE
5 record(s) selected.						

Tom can see all members. Tom belongs to the membership role. He is not privileged to see any data in the PHARMACY and ACC\_BALANCE columns.

Here is the result set Jane sees:

SSN	USERID	NAME	ADDRESS	PHARMACY	ACC_BALANCE	PCP_ID
-----						
XXX-XX-1234	MAX	Max	First Strt	XXXXXXXXXX	0.00	LEE
XXX-XX-9812	MIKE	Mike	Long Strt	XXXXXXXXXX	0.00	JAMES

XXX-XX-9856	SAM	Sam	Big Strt	High blood pressure	0.00	LEE
XXX-XX-1454	DUG	Dug	Good Strt	Influenza	0.00	JAMES
XXX-XX-6789	BOB	Bob	123 Some St.	codeine	0.00	LEE

5 record(s) selected.

Jane can see all members. She belongs to the DRUG\_RESEARCH role. The SSN and ACC\_BALANCE column data are hidden from her. The PHARMACY data is only available if the patients have opted-in to share their data with drug research companies.

Here is the result set John sees:

SSN	USERID	NAME	ADDRESS	PHARMACY	ACC_BALANCE	PCP_ID
123-55-1234	MAX	Max	First Strt	XXXXXXXXXX	89.70	LEE
123-58-9812	MIKE	Mike	Long Strt	XXXXXXXXXX	8.30	JAMES
123-11-9856	SAM	Sam	Big Strt	XXXXXXXXXX	0.00	LEE
123-19-1454	DUG	Dug	Good Strt	XXXXXXXXXX	0.00	JAMES
123-45-6789	BOB	Bob	123 Some St.	XXXXXXXXXX	9.00	LEE

5 record(s) selected.

John can see all members. He belongs to the ACCOUNTING role. The PHARMACY column data is hidden from him.

## Scenario: ExampleHMO using row and column access control - View creation

Views can be created on tables that have row and column access control defined. Alex, the security administrator, is asked to create a view on the PATIENT table that medical researchers can use.

Researchers, that have a partnership with ExampleHMO, can have access to limited patient data if patients have opted-in to permit this access. Alex and the IT team are asked to create a view to list only specific information related to research of the patient. The report must contain the patient insurance number, name of the patient and the disclosure option chosen by the patient.

The view created fetches the patient basic information and the health condition disclosure option. This view ensures that patient information is protected and fetched only with their permission for any other purpose.

Alex and the IT team implement the following view:

```
CREATE VIEW PATIENT_INFO_VIEW AS
SELECT P.SSN, P.NAME FROM PATIENT P, PATIENTCHOICE C
WHERE P.SSN = C.SSN AND
 C.CHOICE = 'drug-research' AND
 C.VALUE = 'opt-in';
```

After Alex and his team create the view, users can query the view. They see data according to the row and column access control rules defined on the base tables on which the view is created.

Alex sees the following result-set from the following query on the view:

```
SELECT SSN, NAME FROM PATIENT_INFO_VIEW;
```

SSN	NAME
-----	-----

0 record(s) selected.

Dr. Lee sees the following result-set from the following query on the view:

```
SELECT SSN, NAME FROM PATIENT_INFO_VIEW;
```

SSN	NAME
-----	-----
123-11-9856	Sam
123-45-6789	Bob

2 record(s) selected.

Bob sees the following result-set from the following query on the view:

```
SELECT SSN, NAME FROM PATIENT_INFO_VIEW;
```

SSN	NAME
-----	-----
123-45-6789	Bob

1 record(s) selected.

## Scenario: ExampleHMO using row and column access control - Secure functions

Functions must be deemed secure before they can be called within row and column access control definitions. Alex, the security administrator, discusses how Paul, a database developer at ExampleHMO, can create a secure function for his new accounting application.

After the privacy and security policy went into effect at ExampleHMO, Alex is notified that the accounting department has developed a powerful accounting application. ExampleHMOAccountingUDF is a SQL scalar user-defined function (UDF) that is used in the column mask ACCT\_BALANCE\_MASK on the PATIENT.ACCT\_BALANCE table and row.

Only UDFs that are secure can be invoked within a column mask. Alex first discusses the UDF with Paul, who wrote the UDF, to ensure the operation inside the UDF is secure.

When Alex is satisfied that the function is secure, he grants a system privilege to Paul so Paul can alter the UDF to be secure:

```
GRANT CREATE_SECURE_OBJECT ON DATABASE TO USER PAUL;
```

To create a secured UDF, or alter a UDF to be secured, a developer must be granted CREATE\_SECURE\_OBJECT authority.

Paul creates the function:

```
CREATE FUNCTION EXAMPLEHMOACCOUNTINGUDF(X DECIMAL(12,2))
 RETURNS DECIMAL(12,2)
 LANGUAGE SQL
 CONTAINS SQL
 DETERMINISTIC
 NO EXTERNAL ACTION
 RETURN X*(1.0 + RAND(X));
```

Paul alters the function so it is secured:

```
ALTER FUNCTION EXAMPLEHMOACCOUNTINGUDF SECURED;
```

Alex now drops and recreates the mask ACC\_BALANCE\_MASK so the new UDF is used:

```
--Drop the mask to recreate
```

```
DROP MASK ACCT_BALANCE_MASK;
```

```
CREATE MASK EXAMPLEHMO.ACCT_BALANCE_MASK ONPATIENT FOR

-- Accounting information:
-- Role ACCOUNTING is allowed to invoke the secured UDF
-- ExampleHMOAccountingUDFL passing column ACCT_BALANCE as
-- the input argument
-- Other ROLES accessing this column will strictly view a
-- zero value.

COLUMN ACCT_BALANCE RETURN
CASE WHEN VERIFY_ROLE_FOR_USER(SESSION_USER,'ACCOUNTING') = 1
THEN EXAMPLEHMOACCOUNTINGUDF(ACCT_BALANCE)
ELSE 0.00
END
ENABLE;
```

Dr. Lee, who has the PCP role, must call a drug analysis user-defined function. DrugUDF returns patient drug information. In the past, Dr. Lee issues a SELECT statement that calls DrugUDF and receives the result set quickly. After the PATIENT table has been protected with row and column access control, the same query takes more time to return a result set.

Dr. Lee consults with the ExampleHMO IT staff and Alex, the security administrator, about this performance degradation. Alex tells Dr. Lee, if the UDF is not secure, the query cannot be optimized as well and it takes longer to return a result set.

Alex looks into the UDF with Dr. Lee and the owner, Paul, to ensure the operation inside the UDF is secure. Alex asks Paul to alter the UDF to be secure as Paul still has the CREATE\_SECURE\_OBJECT privilege granted by Alex:

```
--Function for ExampleHMO Pharmacy department
```

```
CREATE FUNCTION DRUGUDF(PHARMACY VARCHAR(5000))
 RETURNS VARCHAR(5000)
 NO EXTERNAL ACTION
 BEGIN ATOMIC
 IF PHARMACY IS NULL THEN
 RETURN NULL;
 ELSE
 RETURN 'Normal';
 END IF;
 END;
```

```
--Secure the UDF
```

```
ALTER FUNCTION DRUGUDF SECURED;
```

```
--Grant execute permissions to Dr.Lee
```

```
GRANT EXECUTE ON FUNCTION DRUGUDF TO USER LEE;
```

Dr. Lee can issue the query and the query can be optimized as expected:

--Querying after the function is secured

```
SELECT PHARMACY FROM PATIENT
 WHERE DRUGUDF(PHARMACY) = 'Normal' AND SSN = '123-45-6789';
```

PHARMACY

-----

codeine

1 record(s) selected.

## Scenario: ExampleHMO using row and column access control - Secure triggers

Triggers defined on a table with row or column access control activated must be secure. Alex, the security administrator, discusses how Paul, a database developer at ExampleHMO, can create a secure trigger for his new accounting application.

Alex speaks to the accounting department and learns that an AFTER UPDATE trigger is needed for the PATIENT table. This trigger monitors the history of the ACCT\_BALANCE column.

Alex explains to Paul, who has the necessary privileges to create the trigger, that any trigger defined on a row and column access protected table must be marked secure. Paul and Alex review the action of the new trigger and deem it to be secure.

ExampleHMO\_ACCT\_BALANCE\_TRIGGER monitors the ACCT\_BALANCE column in the PATIENT table. Every time that column is updated, the trigger is fired, and inserts the current account balance details into the ACCT\_HISTORY table.

Paul creates the trigger:

```
CREATE TRIGGER HOSPITAL.NETHMO_ACCT_BALANCE_TRIGGER
 AFTER UPDATE OF ACCT_BALANCE ON PATIENT
 REFERENCING OLD AS O NEW AS N
 FOR EACH ROW MODE DB2SQL SECURED
 BEGIN ATOMIC
 INSERT INTO ACCT_HISTORY
 (SSN, BEFORE_BALANCE, AFTER_BALANCE, WHEN, BY_WHO)
 VALUES(O.SSN, O.ACCT_BALANCE, N.ACCT_BALANCE,
 CURRENT_TIMESTAMP, SESSION_USER);
 END;
```

John, from the accounting department, must update the account balance for the patient Bob whose SSN is '123-45-6789'.

John looks at the data for Bob before running the update:

```
SELECT ACCT_BALANCE FROM PATIENT WHERE SSN = '123-45-6789';
```

ACCT\_BALANCE

-----

9.00

1 record(s) selected.

```
SELECT * FROM ACCT_HISTORY WHERE SSN = '123-45-6789';
```

SSN	BEFORE_BALANCE	AFTER_BALANCE	WHEN	BY_WHO
-----				

0 record(s) selected.

John then runs the update:

```
UPDATE PATIENT SET ACCT_BALANCE = ACCT_BALANCE * 0.9 WHERE SSN = '123-45-6789';
```

Since there is a trigger defined on the PATIENT table, the update fires the trigger. Since the trigger is defined SECURED, the update completes successfully. John looks at the data for Bob after running the update:

```
SELECT ACCT_BALANCE FROM PATIENT WHERE SSN = '123-45-6789';
```

ACCT_BALANCE
-----

8.10

1 record(s) selected.

```
SELECT * FROM ACCT_HISTORY WHERE SSN = '123-45-6789';
```

SSN	BEFORE_BALANCE	AFTER_BALANCE	WHEN	BY_WHO
-----				

123-45-6789	9.00	8.10	2010-10-10	JOHN
-------------	------	------	------------	------

1 record(s) selected.

## Scenario: ExampleHMO using row and column access control - Revoke authority

Alex, as security administrator, is responsible for controlling who can create secure objects. When developers are done creating secure objects, Alex revokes their authority on the database.

Paul, the database developer, is done with development activities. Alex immediately revokes the create authority from Paul:

```
REVOKE CREATE_SECURE_OBJECT ON DATABASE FROM USER PAUL;
```

If Paul must create secure objects in the future, he must speak to Alex to have the create authority granted again.

---

## Scenario: ExampleBANK using row and column access control

This scenario presents ExampleBANK, a banking institution with a large customer base spanning many branches, as a user of row and column access control. ExampleBANK uses row and column access control to ensure that their database policies reflect company requirements for privacy and security, as well as management business objectives.

Organizations that handle client investments, savings, and their personal information, like ExampleBANK, only share information within their organization on a must know basis. This data protection ensures that any sensitive client financial or personal information is shared, viewed, and modified only by employees who are privileged to do so.

## Scenario: ExampleBANK using row and column access control - Security policies

ExampleBANK implements a security strategy where data access to databases is made available according to certain security policies.

The security policies conform to privacy and data protection regulations at ExampleBANK. The first column outlines the policies and the challenges faced by ExampleBANK, the second column outlines the row and column access control (RCAC) feature which addresses the challenge.

Security challenge	Row and column access control feature which addresses the security challenge
Limiting row access to only authorized users. Tellers are only permitted to view client data that belong to their own branch, not all clients of ExampleBANK in the company-wide system.	Row permissions can be implemented to control which user can view any particular row.
The account number is accessible by customer service representatives only when they are using the account update application. This application is identified through stored procedure ACCOUNTS.ACCTUPDATE.	Column masks can be used to filter or hide sensitive data from customer service representatives if they query the data outside of the ACCOUNTS.ACCTUPDATE application.

## Scenario: ExampleBANK using row and column access control - Database users and roles

In this scenario, a number of different people use ExampleBANK data. These people have different user rights.

ExampleBANK implemented their security strategy to classify the way data is accessed from the database. Internal access to data is based on the separation of duties to users who access the data and their data access privileges. ExampleBANK created the following database roles to separate these duties:

### TELLER

For tellers of branch locations.

### TELEMARKETER

For telephone marketing and sales people.

### CSR

For customer service representatives.

The following people use ExampleBANK data:

### ZURBIE

A customer service representative at ExampleBANK. She belongs to the CSR role.

### NEWTON

A teller at an ExampleBANK branch. He belongs to the TELLER role.

### PLATO

A telephone marketing and sales person at ExampleBANK. He belongs to the TELEMARKETER role.



If you want to try any of the example SQL statements and commands presented in this scenario, create these user IDs with their listed authorities.

The following example SQL statements assume that the users have been created on the system. The SQL statements create each role and grant SELECT permission to the various tables in the ExampleBANK database to the users:

```
--Creating roles and granting authority

CREATE ROLE TELLER;

CREATE ROLE CSR;

CREATE ROLE TELEMARKETER;

GRANT ROLE TELLER TO USER NEWTON;
GRANT ROLE CSR TO USER ZURBIE;
GRANT ROLE TELEMARKETER TO USER PLATO;
```

## Scenario: ExampleBANK using row and column access control - Database tables

This scenario focuses on two tables in the ExampleBANK database: the CUSTOMER table and the INTERNAL\_INFO table.

The INTERNAL\_INFO table stores information about employees who work for ExampleBANK. This scenario considers the following columns within the INTERNAL\_INFO table:

### HOME\_BRANCH

The employee home branch ID.

### EMP\_ID

The employee ID.

The CUSTOMER table stores individual client information:

### ACCOUNT

The client account number.

### NAME

The client name.

### INCOME

The client income.

### BRANCH

The client branch ID.

The following example SQL statements create the customer, and INTERNAL\_INFO tables. Authority is granted on the tables and data is inserted:

```
--Client table storing information regarding client information
CREATE TABLE RACTSPM.CUSTOMER (
 ACCOUNT VARCHAR(19),
 NAME VARCHAR(20),
 INCOME INTEGER,
 BRANCH CHAR(1)
);

--Internal_info table which stores employee information
CREATE TABLE RACTSPM.INTERNAL_INFO (
 HOME_BRANCH CHAR(1),
```

```

EMP_ID VARCHAR(10));

--Grant authority

GRANT SELECT ON RACTSPM.CUSTOMER TO USER NEWTON, USER ZURBIE, USER PLATO;

--Insert data

INSERT INTO RACTSPM.CUSTOMER VALUES ('1111-2222-3333-4444', 'Alice', 22000, 'A');
INSERT INTO RACTSPM.CUSTOMER VALUES ('2222-3333-4444-5555', 'Bob', 71000, 'A');
INSERT INTO RACTSPM.CUSTOMER VALUES ('3333-4444-5555-6666', 'Carl', 123000, 'B');
INSERT INTO RACTSPM.CUSTOMER VALUES ('4444-5555-6666-7777', 'David', 172000, 'C');

INSERT INTO RACTSPM.INTERNAL_INFO VALUES ('A', 'NEWTON');
INSERT INTO RACTSPM.INTERNAL_INFO VALUES ('B', 'ZURBIE');
INSERT INTO RACTSPM.INTERNAL_INFO VALUES ('C', 'PLATO');

```

## Scenario: ExampleBANK using row and column access control - Row permissions

The security administrator at ExampleBANK, starts to restrict data access by using row permissions, a part of row and column access control. Row permissions filter the data returned to users by row.

Tellers are permitted to view client data only from their home branch. Telemarketers and CSRs are permitted to see all ExampleBANK clients in the system, but telemarketers cannot see the full account number.

Row permissions restrict or filter rows based on the user who has logged on to the database. At ExampleBANK, the row permissions create a horizontal data restriction on the CUSTOMER table.

The security administrator implements the following row permissions so that a user in each role is restricted to view a result set that they are privileged to view:

```

CREATE PERMISSION TELLER_ROW_ACCESS ON RACTSPM.CUSTOMER

-- Teller information:
-- ROLE TELLER is allowed to access client data only
-- in their branch.

FOR ROWS WHERE VERIFY_ROLE_FOR_USER(USER, 'TELLER') = 1
AND
BRANCH = (SELECT HOME_BRANCH FROM RACTSPM.INTERNAL_INFO WHERE EMP_ID = USER)
ENFORCED FOR ALL ACCESS
ENABLE;

CREATE PERMISSION CSR_ROW_ACCESS ON RACTSPM.CUSTOMER

-- CSR and telemarketer information:
-- ROLE TELEMARKETER and CSR are allowed to access all client
-- data rows in ExampleBANK.

FOR ROWS WHERE VERIFY_ROLE_FOR_USER (USER, 'CSR') = 1
OR
VERIFY_ROLE_FOR_USER (USER, 'TELEMARKETER') = 1
ENFORCED FOR ALL ACCESS
ENABLE;

```

The security administrator observes that even after creating a row permission, all data can still be viewed by the employees. A row permission is not applied until it is activated on the table for which it was defined. The security administrator must now activate the permission:

```
--Activate row access control to implement row permissions
```

```
ALTER TABLE RACTSPM.CUSTOMER ACTIVATE ROW ACCESS CONTROL;
```

## Scenario: ExampleBANK using row and column access control - Column masks

The ExampleBANK security administrator, further restricts data access by using column masks, a part of row and column access control. Column masks hide data returned to users or applications by column unless they are permitted to view the data.

Customer service representatives can see all clients in the ExampleBANK system, but, they are not permitted to view full account numbers unless they are using a specific application.

The security administrator implements the following column mask so that a customer service representative is restricted to view a result set that they are privileged to view:

```
CREATE MASK ACCOUNT_COL_MASK ON RACTSPM.CUSTOMER FOR

-- Account number information:
-- Role customer service representative (CSR) is allowed to
-- access account number information only when they are using
-- the account update application. This application is
-- identified through stored procedure ACCOUNTS.ACCTUPDATE.
-- If a CSR queries this data outside of this application, the
-- account information is masked and the first 12 digits are
-- replaced with "x".

COLUMN ACCOUNT RETURN
 CASE WHEN (VERIFY_ROLE_FOR_USER (USER, 'CSR') = 1 AND
 ROUTINE_SPECIFIC_NAME = 'ACCTUPDATE' AND
 ROUTINE_SCHEMA = 'ACCOUNTS' AND
 ROUTINE_TYPE = 'P')
 THEN ACCOUNT
 ELSE 'xxxx-xxxx-xxxx-' || SUBSTR(ACCOUNT,16,4)
END
ENABLE;
```

The security administrator observes that even after creating a column mask, the data can still be viewed by all employees. A column mask is not applied until it is activated on the table for which it was defined. The security administrator must now activate the mask:

```
--Activate column access control to implement column masks
```

```
ALTER TABLE RACTSPM.CUSTOMER ACTIVATE COLUMN ACCESS CONTROL;
```

## Scenario: ExampleBANK using row and column access control - Data queries

With row and column access control, people in different roles can have different result sets from the same database queries. For example, Newton, a teller, cannot see any data of clients outside of their branch.

Newton, Zurbie, and Plato each connect to the database and try the following SQL query:

```
SELECT * FROM RACTSPM.CUSTOMER;
```

Results of the query vary according to who runs the query. The row and column access control rules created by the security administrator are applied on these queries.

Here is the result set Newton sees:

ACCOUNT	NAME	INCOME	BRANCH
xxxx-xxxx-xxxx-4444	Alice	22000	A
xxxx-xxxx-xxxx-5555	Bob	71000	A

2 record(s) selected.

Newton, being a teller at branch A, can see only ExampleBANK clients that belong to that branch.

Here is the result set Zurbie sees:

ACCOUNT	NAME	INCOME	BRANCH
xxxx-xxxx-xxxx-4444	Alice	22000	A
xxxx-xxxx-xxxx-5555	Bob	71000	A
xxxx-xxxx-xxxx-6666	Carl	123000	B
xxxx-xxxx-xxxx-7777	David	172000	C

4 record(s) selected.

Zurbie, being a customer service representative, can see all ExampleBANK clients in the system, but not their full account number unless he uses the ACCOUNTS.ACCTUPDATE application. Since this query was issued outside of ACCOUNTS.ACCTUPDATE, part of that number is masked.

Here is the result set Plato sees:

ACCOUNT	NAME	INCOME	BRANCH
xxxx-xxxx-xxxx-4444	Alice	22000	A
xxxx-xxxx-xxxx-5555	Bob	71000	A
xxxx-xxxx-xxxx-6666	Carl	123000	B
xxxx-xxxx-xxxx-7777	David	172000	C

4 record(s) selected.

Plato, being a telemarketer, can see all ExampleBANK clients in the system.

---

## Chapter 5. Label-based access control (LBAC)

Label-based access control (LBAC) greatly increases the control you have over who can access your data. LBAC lets you decide exactly who has write access and who has read access to individual rows and individual columns.

### What LBAC does

The LBAC capability is very configurable and can be tailored to match your particular security environment. All LBAC configuration is performed by a *security administrator*, which is a user that has been granted the SECADM authority.

A security administrator configures the LBAC system by creating security label components. A *security label component* is a database object that represents a criterion you want to use to determine if a user should access a piece of data. For example, the criterion can be whether the user is in a certain department, or whether they are working on a certain project. A *security policy* describes the criteria that will be used to decide who has access to what data. A security policy contains one or more security label components. Only one security policy can be used to protect any one table but different tables can be protected by different security policies.

After creating a security policy, a security administrator creates objects, called *security labels* that are part of that policy. Security labels contain security label components. Exactly what makes up a security label is determined by the security policy and can be configured to represent the criteria that your organization uses to decide who should have access to particular data items. If you decide, for example, that you want to look at a person's position in the company and what projects they are part of to decide what data they should see, then you can configure your security labels so that each label can include that information. LBAC is flexible enough to let you set up anything from very complicated criteria, to a very simple system where each label represents either a "high" or a "low" level of trust.

Once created, a security label can be associated with individual columns and rows in a table to protect the data held there. Data that is protected by a security label is called *protected data*. A security administrator allows users access to protected data by granting them security labels. When a user tries to access protected data, that user's security label is compared to the security label protecting the data. The protecting label will block some security labels and not block others.

A user, a role, or a group is allowed to hold security labels for multiple security policies at once. For any given security policy, however, a user, a role, or a group can hold at most one label for read access and one label for write access.

A security administrator can also grant exemptions to users. An *exemption* allows you to access protected data that your security labels might otherwise prevent you from accessing. Together your security labels and exemptions are called your *LBAC credentials*.

If you try to access a protected column that your LBAC credentials do not allow you to access then the access will fail and you will get an error message.

If you try to read protected rows that your LBAC credentials do not allow you to read then Db2 acts as if those rows do not exist. Those rows cannot be selected as part of any SQL statement that you run, including SELECT, UPDATE, or DELETE. Even the aggregate functions ignore rows that your LBAC credentials do not allow you to read. The COUNT(\*) function, for example, will return a count only of the rows that you have read access to.

## Views and LBAC

You can define a view on a protected table the same way you can define one on a non-protected table. When such a view is accessed the LBAC protection on the underlying table is enforced. The LBAC credentials used are those of the session authorization ID. Two users accessing the same view might see different rows depending on their LBAC credentials.

## Referential integrity constraints and LBAC

The following rules explain how LBAC rules are enforced in the presence of referential integrity constraints:

- **Rule 1:** The LBAC read access rules are NOT applied for internally generated scans of child tables. This is to avoid having orphan children.
- **Rule 2:** The LBAC read access rules are NOT applied for internally generated scans of parent tables
- **Rule 3:** The LBAC write rules are applied when a CASCADE operation is performed on child tables. For example, If a user deletes a parent, but cannot delete any of the children because of an LBAC write rule violation, then the delete should be rolled-back and an error raised.

## Storage overhead when using LBAC

When you use LBAC to protect a table at the row level, the additional storage cost is the cost of the row security label column. This cost depends on the type of security label chosen. For example, if you create a security policy with two components to protect a table, a security label from that security policy will occupy 16 bytes (8 bytes for each component). Because the row security label column is treated as a not nullable VARCHAR column, the total cost in this case would be 20 bytes per row. In general, the total cost per row is  $(N*8 + 4)$  bytes where  $N$  is the number of components in the security policy protecting the table.

When you use LBAC to protect a table at the column level, the column security label is meta-data (that is, it is stored together with the column's meta-data in the SYSCOLUMNS catalog table). This meta-data is simply the ID of the security label protecting the column. The user table does not incur any storage overhead in this case.

## What LBAC does not do

- LBAC will never allow access to data that is forbidden by discretionary access control.

**Example:** If you do not have permission to read from a table then you will not be allowed to read data from that table—even the rows and columns to which LBAC would otherwise allow you access.

- Your LBAC credentials only limit your access to protected data. They have no effect on your access to unprotected data.

- LBAC credentials are not checked when you drop a table or a database, even if the table or database contains protected data.
- LBAC credentials are not checked when you back up your data. If you can run a backup on a table, which rows are backed up is not limited in any way by the LBAC protection on the data. Also, data on the backup media is not protected by LBAC. Only data in the database is protected.
- LBAC cannot be used to protect any of the following types of tables:
  - A staging table
  - A table that a staging table depends on
  - A typed table
- LBAC protection cannot be applied to a nickname.

## LBAC tutorial

A tutorial leading you through the basics of using LBAC is available online at <http://www.ibm.com/developerworks/data> and is called DB2 Label-Based Access Control, a practical guide.

---

## LBAC security policies

The security administrator uses a security policy to define criteria that determine who has write access and who has read access to individual rows and individual columns of tables.

A security policy includes this information:

- What security label components are used in the security labels that are part of the policy
- What rules are used when comparing those security label components
- Which of certain optional behaviors are used when accessing data protected by the policy
- What additional security labels and exemptions are to be considered when enforcing access to data protected by the security policy. For example, the option to consider or not to consider security labels granted to roles and groups is controlled through the security policy.

Every protected table must have one and only one security policy associated with it. Rows and columns in that table can only be protected with security labels that are part of that security policy and all access of protected data follows the rules of that policy. You can have multiple security policies in a single database but you cannot have more than one security policy protecting any given table.

## Creating a security policy

You must be a security administrator to create a security policy. You create a security policy with the SQL statement `CREATE SECURITY POLICY`. The security label components listed in a security policy must be created before the `CREATE SECURITY POLICY` statement is executed. The order in which the components are listed when a security policy is created does not indicate any sort of precedence or other relationship among the components but it is important to know the order when creating security labels with built-in functions like `SECLABEL`.

From the security policy you have created, you can create security labels to protect your data.

## Altering a security policy

A security administrator can use the ALTER SECURITY POLICY statement to modify a security policy.

## Dropping a security policy

You must be a security administrator to drop a security policy. You drop a security policy using the SQL statement DROP.

You cannot drop a security policy if it is associated with (added to) any table.

---

## LBAC security label components overview

A *security label component* is a database object that is part of label-based access control (LBAC). You use security label components to model your organization's security structure.

A security label component can represent any criteria that you might use to decide if a user should have access to a given piece of data. Typical examples of such criteria include:

- How well trusted the user is
- What department the user is in
- Whether the user is involved in a particular project

**Example:** If you want the department that a user is in to affect which data they can access, you could create a component named dept and define elements for that component that name the various departments in your company. You would then include the component dept in your security policy.

An *element* of a security label component is one particular "setting" that is allowed for that component.

**Example:** A security label component that represents a level of trust might have the four elements: Top Secret, Secret, Classified, and Unclassified.

## Creating a security label component

You must be a security administrator to create a security label component. You create security label components with the SQL statement CREATE SECURITY LABEL COMPONENT.

When you create a security label component you must provide:

- A name for the component
- What type of component it is (ARRAY, TREE, or SET)
- A complete list of allowed elements
- For types ARRAY and TREE you must describe how each element fits into the structure of the component

After creating your security label components, you can create a security policy based on these components. From this security policy, you can create security labels to protect your data.



## Types of components

There are three types of security label components:

- TREE: Each element represents a node in a tree structure
- ARRAY: Each element represents a point on a linear scale
- SET: Each element represents one member of a set

The types are used to model the different ways in which elements can relate to each other. For example, if you are creating a component to describe one or more departments in a company you would probably want to use a component type of TREE because most business structures are in the form of a tree. If you are creating a component to represent the level of trust that a person has, you would probably use a component of type ARRAY because for any two levels of trust, one will always be higher than the other.

The details of each type, including detailed descriptions of the relationships that the elements can have with each other, are described in their own section.

## Altering security label components

The security administrator can use the ALTER SECURITY LABEL COMPONENT statement to modify a security label component.

## Dropping a security label component

You must be a security administrator to drop a security label component. You drop a security label component with the SQL statement DROP.

## LBAC security label component type: SET

SET is one type of security label component that can be used in a label-based access control (LBAC) security policy.

Components of type SET are unordered lists of elements. The only comparison that can be made for elements of this type of component is whether or not a given element is in the list.

## LBAC security label component type: ARRAY

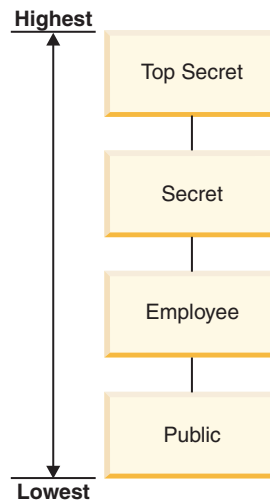
ARRAY is one type of security label component.

In the ARRAY type of component the order in which the elements are listed when the component is created defines a scale with the first element listed being the highest value and the last being the lowest.

**Example:** If the component mycomp is defined in this way:

```
CREATE SECURITY LABEL COMPONENT mycomp
 ARRAY ['Top Secret', 'Secret', 'Employee', 'Public']
```

Then the elements are treated as if they are organized in a structure like this:



In a component of type ARRAY, the elements can have these sorts of relationships to each other:

**Higher than**

Element A is higher than element B if element A is listed earlier in the ARRAY clause than element B.

**Lower than**

Element A is lower than element B if element A is listed later in the ARRAY clause than element B

## LBAC security label component type: TREE

TREE is one type of security label component that can be used in a label-based access control (LBAC) security policy.

In the TREE type of component the elements are treated as if they are arranged in a tree structure. When you specify an element that is part of a component of type TREE you must also specify which other element it is under. The one exception is the first element which must be specified as being the ROOT of the tree. This allows you to organize the elements in a tree structure.

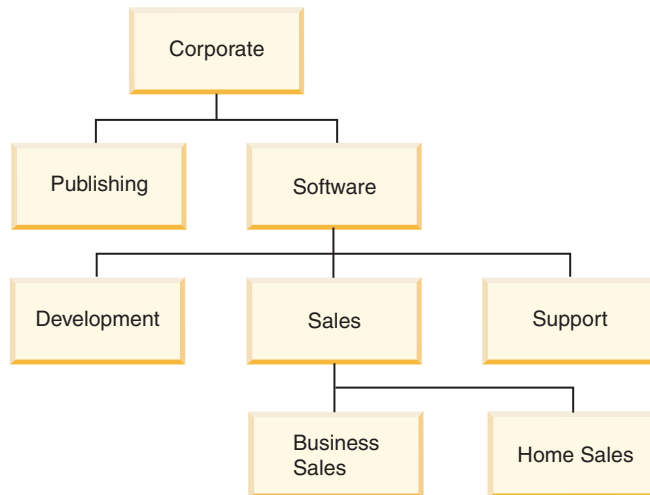
**Example:** If the component mycomp is defined this way:

```

CREATE SECURITY LABEL COMPONENT mycomp
TREE (
 'Corporate' ROOT,
 'Publishing' UNDER 'Corporate',
 'Software' UNDER 'Corporate',
 'Development' UNDER 'Software',
 'Sales' UNDER 'Software',
 'Support' UNDER 'Software'
 'Business Sales' UNDER 'Sales'
 'Home Sales' UNDER 'Sales'
)

```

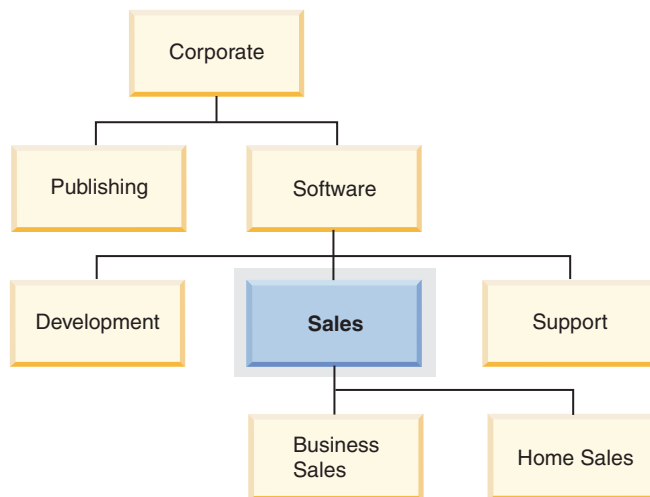
Then the elements are treated as if they are organized in a tree structure like this:



In a component of type TREE, the elements can have these types of relationships to each other:

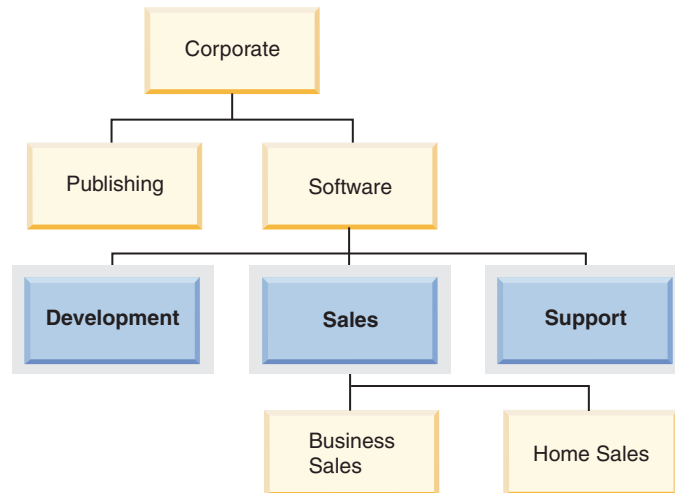
**Parent** Element A is a parent of element B if element B is UNDER element A.

**Example:** This diagram shows the parent of the Business Sales element:



**Child** Element A is a child of element B if element A is UNDER element B.

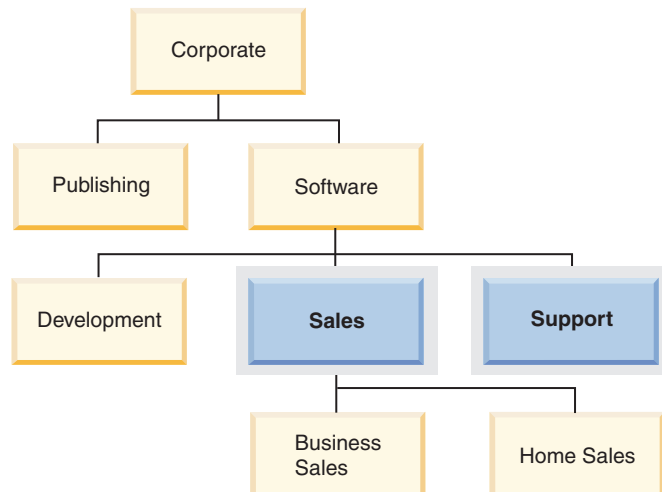
**Example:** This diagram shows the children of the Software element:



### Sibling

Two elements are siblings of each other if they have the same parent.

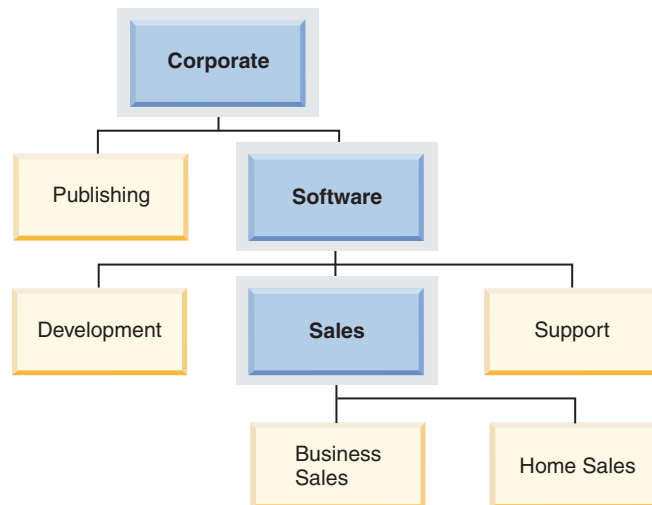
**Example:** This diagram shows the siblings of the Development element:



### Ancestor

Element A is an ancestor of element B if it is the parent of B, or if it is the parent of the parent of B, and so on. The root element is an ancestor of all other elements in the tree.

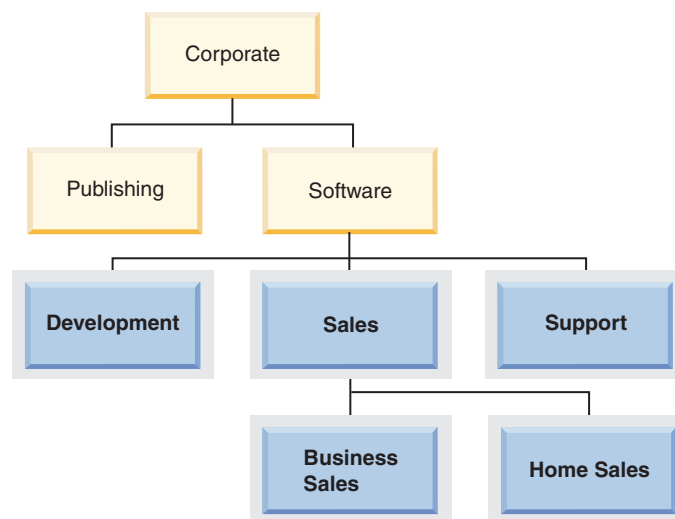
**Example:** This diagram shows the ancestors of the Home Sales element:



### Descendent

Element A is a descendent of element B if it is the child of B, or if it is the child of a child of B, and so on.

**Example:** This diagram shows the descendents of the Software element:



## LBAC security labels

In label-based access control (LBAC) a *security label* is a database object that describes a certain set of security criteria. Security labels are applied to data in order to protect the data. They are granted to users to allow them to access protected data.

When a user tries to access protected data, their security label is compared to the security label that is protecting the data. The protecting security label will block some security labels and not block others. If a user's security label is blocked then the user cannot access the data.

Every security label is part of exactly one security policy and includes one value for each component in that security policy. A *value* in the context of a security label component is a list of zero or more of the elements allowed by that component.

Values for ARRAY type components can contain zero or one element, values for other types can have zero or more elements. A value that does not include any elements is called an *empty value*.

**Example:** If a TREE type component has the three elements Human Resources, Sales, and Shipping then these are some of the valid values for that component:

- Human Resources (or any of the elements by itself)
- Human Resources, Shipping (or any other combination of the elements as long as no element is included more than once)
- *An empty value*

Whether a particular security label will block another is determined by the values of each component in the labels and the LBAC rule set that is specified in the security policy of the table. The details of how the comparison is made are given in the topic that discusses how LBAC security labels are compared.

When security labels are converted to a text string they use the format described in the topic that discusses the format for security label values.

## Creating security labels

You must be a security administrator to create a security label. You create a security label with the SQL statement CREATE SECURITY LABEL. When you create a security label you provide:

- A name for the label
- The security policy that the label is part of
- Values for one or more of the components included in the security policy

Any components for which a value is not specified is assumed to have an empty value. A security label must have at least one non-empty value.

## Altering security labels

Security labels cannot be altered. The only way to change a security label is to drop it and re-create it. However, the *components* of a security label can be modified by a security administrator (using the ALTER SECURITY LABEL COMPONENT statement).

## Dropping security labels

You must be a security administrator to drop a security label. You drop a security label with the SQL statement DROP. You cannot drop a security label that is being used to protect data anywhere in the database or that is currently held by one or more users.

## Granting security labels

You must be a security administrator to grant a security label to a user, a group, or a role. You grant a security label with the SQL statement GRANT SECURITY LABEL. When you grant a security label you can grant it for read access, for write access, or for both read and write access. A user, a group, or a role cannot hold more than one security label from the same security policy for the same type of access.

## Revoking security labels

You must be a security administrator to revoke a security label from a user, group, or role. To revoke a security label, use the SQL statement `REVOKE SECURITY LABEL`.

## Data types compatible with security labels

Security labels have a data type of `SYSPROC.DB2SECURITYLABEL`. Data conversion is supported between `SYSPROC.DB2SECURITYLABEL` and `VARCHAR(128) FOR BIT DATA`.

## Determining the security labels held by users

You can use the following query to determine the security labels that are held by users:

```
SELECT A.grantee, B.secpolicyname, c.seclabelname
FROM syscat.securitylabelaccess A, syscat.securitypolicies B, syscat.securitylabels C
WHERE A.seclabelid = C.seclabelid and B.secpolicyid = C.secpolicyid
```

---

## Format for security label values

Sometimes the values in a security label are represented in the form of a character string, for example when using the built-in function `SECLABEL`.

When the values in a security label are represented as a string, they are in the following format:

- The values of the components are listed from left to right in the same order that the components are listed in the `CREATE SECURITY POLICY` statement for the security policy
- An element is represented by the name of that element
- Elements for different components are separated by a colon (:)
- If more than one element are given for the same component the elements are enclosed in parentheses (()) and are separated by a comma (,)
- Empty values are represented by a set of empty parentheses (())

**Example:** A security label is part of a security policy that has these three components in this order: Level, Department, and Projects. The security label has these values:

*Table 10. Example values for a security label*

Component	Values
Level	Secret
Department	<i>Empty value</i>
Projects	<ul style="list-style-type: none"><li>• Epsilon 37</li><li>• Megaphone</li><li>• Cloverleaf</li></ul>

This security label values look like this as a string:

```
'Secret:(): (Epsilon 37,Megaphone,Cloverleaf)'
```

---

## How LBAC security labels are compared

When you try to access data protected by label-based access control (LBAC), your LBAC credentials are compared to one or more security labels to see if the access is blocked. Your LBAC credentials are any security labels you hold plus any exemptions that you hold.

There are only two types of comparison that can be made. Your LBAC credentials can be compared to a single security label for read access or your LBAC credentials compared to a single security label for write access. Updating and deleting are treated as being a read followed by a write. When an operation requires multiple comparisons to be made, each is made separately.

### Which of your security labels is used

Even though you might hold multiple security labels only one is compared to the protecting security label. The label used is the one that meets these criteria:

- It is part of the security policy that is protecting the table being accessed.
- It was granted for the type of access (read or write).

If you do not have a security label that meets these criteria then a default security label is assumed that has empty values for all components.

### How the comparison is made

Security labels are compared component by component. If a security label does not have a value for one of the components then an empty value is assumed. As each component is examined, the appropriate rules of the LBAC rule set are used to decide if the elements in your value for that component should be blocked by the elements in the value for the same component in the protecting label. If any of your values are blocked then your LBAC credentials are blocked by the protecting security label.

The LBAC rule set used in the comparison is designated in the security policy. To find out what the rules are and when each one is used, see the description of that rule set.

### How exemptions affect comparisons

If you hold an exemption for the rule that is being used to compare two values then that comparison is not done and the protecting value is assumed not to block the value in your security label.

**Example:** The LBAC rule set is DB2LBACRULES and the security policy has two components. One component is of type ARRAY and the other is of type TREE. The user has been granted an exemption on the rule DB2LBACREADTREE, which is the rule used for read access when comparing values of components of type TREE. If the user attempts to read protected data then whatever value the user has for the TREE component, even if it is an empty value, will not block access because that rule is not used. Whether the user can read the data depends entirely on the values of the ARRAY component of the labels.



---

## LBAC rule sets overview

An LBAC rule set is a predefined set of rules that are used when comparing security labels. When the values of a two security labels are being compared, one or more of the rules in the rule set will be used to determine if one value blocks another.

Each LBAC rule set is identified by a unique name. When you create a security policy you must specify the LBAC rule set that will be used with that policy. Any comparison of security labels that are part of that policy will use that LBAC rule set.

Each rule in a rule set is also identified by a unique name. You use the name of a rule when you are granting an exemption on that rule.

How many rules are in a set and when each rule is used can vary from rule set to rule set.

There is currently only one supported LBAC rule set. The name of that rule set is DB2LBACRULES.

### LBAC rule set: DB2LBACRULES

The DB2LBACRULES LBAC rule set provides a traditional set of rules for comparing the values of security label components. It protects from both write-up and write-down.

#### What are write-up and write down?

Write-up and write-down apply only to components of type ARRAY and only to write access. Write up occurs when the value protecting data that you are writing to is higher than your value. Write-down is when the value protecting the data is lower than yours. By default neither write-up nor write-down is allowed, meaning that you can only write data that is protected by the same value that you have.

When comparing two values for the same component, which rules are used depends on the type of the component (ARRAY, SET, or TREE) and what type of access is being attempted (read, or write). This table lists the rules, tells when each is used, and describes how the rule determines if access is blocked.

*Table 11. Summary of the DB2LBACRULES rules*

Rule name	Used to compare values of this type of component	Used for this type of access	Access is blocked when this condition is met
DB2LBACREADARRAY	ARRAY	Read	The user's value is lower than the protecting value.
DB2LBACREADSET	SET	Read	There are one or more protecting values that the user does not hold.
DB2LBACREADTREE	TREE	Read	None of the user's values is equal to or an ancestor of one of the protecting values.
DB2LBACWRITEARRAY	ARRAY	Write	The user's value is higher than the protecting value or lower than the protecting value. <sup>1</sup>

Table 11. Summary of the DB2LBACRULES rules (continued)

Rule name	Used to compare values of this type of component	Used for this type of access	Access is blocked when this condition is met
DB2LBACWRITESET	SET	Write	There are one or more protecting values that the user does not hold.
DB2LBACWRITETREE	TREE	Write	None of the user's values is equal to or an ancestor of one of the protecting values.

**Note:**

1. The DB2LBACWRITEARRAY rule can be thought of as being two different rules combined. One prevents writing to data that is higher than your level (write-up) and the other prevents writing to data that is lower than your level (write-down). When granting an exemption to this rule you can exempt the user from either of these rules or from both.

## How the rules handle empty values

All rules treat empty values the same way. An empty value blocks no other values and is blocked by any non-empty value.

## DB2LBACREADSET and DB2LBACWRITESET examples

These examples are valid for a user trying to read or trying to write protected data. They assume that the values are for a component of type SET that has these elements: one two three four

Table 12. Examples of applying the DB2LBACREADSET and DB2LBACWRITESET rules.

User's value	Protecting value	Access blocked?
'one'	'one'	Not blocked. The values are the same.
'(one,two,three)'	'one'	Not blocked. The user's value contains the element 'one'.
'(one,two)'	'(one,two,four)'	Blocked. The element 'four' is in the protecting value but not in the user's value.
'()'	'one'	Blocked. An empty value is blocked by any non-empty value.
'one'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

## DB2LBACREADTREE and DB2LBACWRITETREE

These examples are valid for both read access and write access. They assume that the values are for a component of type TREE that was defined in this way:

```
CREATE SECURITY LABEL COMPONENT mycomp
TREE (
 'Corporate' ROOT,
```

```

'Publishing' UNDER 'Corporate',
'Software' UNDER 'Corporate',
'Development' UNDER 'Software',
'Sales' UNDER 'Software',
'Support' UNDER 'Software'
'Business Sales' UNDER 'Sales'
'Home Sales' UNDER 'Sales'
)

```

This means the elements are in this arrangement:

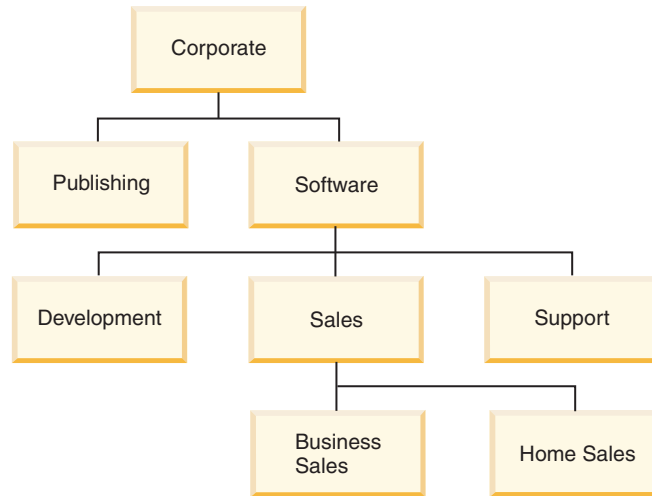


Table 13. Examples of applying the DB2LBACREADTREE and DB2LBACWRITETREE rules.

User's value	Protecting value	Access blocked?
'(Support,Sales)'	'Development'	Blocked. The element 'Development' is not one of the user's values and neither 'Support' nor 'Sales' is an ancestor of 'Development'.
'(Development,Software)'	'(Business Sales,Publishing)'	Not blocked. The element 'Software' is an ancestor of 'Business Sales'.
'(Publishing,Sales)'	'(Publishing,Support)'	Not blocked. The element 'Publishing' is in both sets of values.
'Corporate'	'Development'	Not blocked. The root value is an ancestor of all other values.
'()'	'Sales'	Blocked. An empty value is blocked by any non-empty value.
'Home Sales'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

### DB2LBACREADARRAY examples

These examples are for read access only. They assume that the values are for a component of type ARRAY that includes these elements in this arrangement:

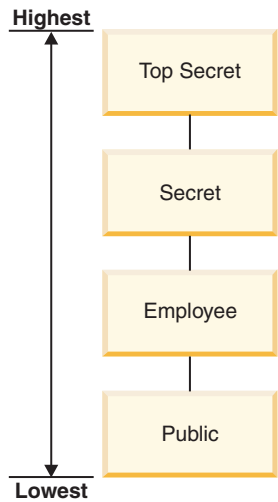


Table 14. Examples of applying the DB2LBACREADARRAY rule.

User's value	Protecting value	Read access blocked?
'Secret'	'Employee'	Not blocked. The element 'Secret' is higher than the element 'Employee'.
'Secret'	'Secret'	Not blocked. The values are the same.
'Secret'	'Top Secret'	Blocked. The element 'Top Secret' is higher than the element 'Secret'.
'()	'Public'	Blocked. An empty value is blocked by any non-empty value.
'Public'	'()	Not blocked. No value is blocked by an empty value.
'()	'()	Not blocked. No value is blocked by an empty value.

### DB2LBACWRITEARRAY examples

These examples are for write access only. They assume that the values are for a component of type ARRAY that includes these elements in this arrangement:

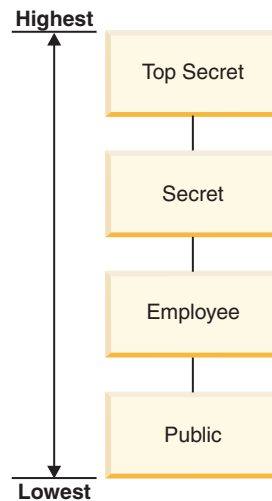


Table 15. Examples of applying the DB2LBACWRITEARRAY rule.

User's value	Protecting value	Write access blocked?
'Secret'	'Employee'	Blocked. The element 'Employee' is lower than the element 'Secret'.
'Secret'	'Secret'	Not blocked. The values are the same.
'Secret'	'Top Secret'	Blocked. The element 'Top Secret' is higher than the element 'Secret'.
'()	'Public'	Blocked. An empty value is blocked by any non-empty value.
'Public'	'()	Not blocked. No value is blocked by an empty value.
'()	'()	Not blocked. No value is blocked by an empty value.

## LBAC rule exemptions

When you hold an LBAC rule exemption on a particular rule of a particular security policy, that rule is not enforced when you try to access data protected by that security policy.

An exemption has no effect when comparing security labels of any security policy other than the one for which it was granted.

### Example:

There are two tables: T1 and T2. T1 is protected by security policy P1 and T2 is protected by security policy P2. Both security policies have one component. The component of each is of type ARRAY. T1 and T2 each contain only one row of data. The security label that you hold for read access under security policy P1 does not allow you access to the row in T1. The security label that you hold for read access under security policy P2 does not allow you read access to the row in T2.

Now you are granted an exemption on DB2LBACREADARRAY under P1. You can now read the row from T1 but not the row from T2 because T2 is protected by a different security policy and you do not hold an exemption to the DB2LBACREADARRAY rule in that policy.

You can hold multiple exemptions. If you hold an exemption to every rule used by a security policy then you will have complete access to all data protected by that security policy.

## Granting LBAC rule exemptions

You must be a security administrator to grant an LBAC rule exemption. To grant an LBAC rule exemption, use the SQL statement `GRANT EXEMPTION ON RULE`.

When you grant an LBAC rule exemption you provide this information:

- The rule or rules that the exemption is for
- The security policy that the exemption is for
- The user, group, or role to which you are granting the exemption

**Important:** LBAC rule exemptions provide very powerful access. Do not grant them without careful consideration.

## Revoking LBAC rule exemptions

You must be a security administrator to revoke an LBAC rule exemption. To revoke an LBAC rule exemption, use the SQL statement `REVOKE EXEMPTION ON RULE`.

## Determining the rule exemptions held by users

You can use the following query to determine the rule exemptions that are held by users:

```
SELECT A.grantee, A.accessrulename, B.secpolicyname
FROM syscat.securitypolicyexemptions A, syscat.securitypolicies B
WHERE A.secpolicyid = B.secpolicyid
```

---

## Built-in functions for managing LBAC security labels

The built-in functions `SECLABEL`, `SECLABEL_BY_NAME`, and `SECLABEL_TO_CHAR` are provided for managing label-based access control (LBAC) security labels.

Each is described briefly here and in detail in the *SQL Reference*

### SECLABEL

This built-in function is used to build a security label by specifying a security policy and values for each of the components in the label. The returned value has a data type of `DB2SECURITYLABEL` and is a security label that is part of the indicated security policy and has the indicated values for the components. It is not necessary that a security label with the indicated values already exists.

**Example:** Table T1 has two columns, the first has a data type of `DB2SECURITYLABEL` and the second has a data type of `INTEGER`. T1 is protected by security policy P1, which has three security label components: level, departments, and groups. If `UNCLASSIFIED` is an element of the component level, `ALPHA` and `SIGMA` are both elements of the component departments, and `G2` is an element of the component groups then a security label could be inserted like this:

```
INSERT INTO T1 VALUES
(SECLABEL('P1', 'UNCLASSIFIED:(ALPHA,SIGMA):G2'), 22)
```

## SECLABEL\_BY\_NAME

This built-in function accepts the name of a security policy and the name of a security label that is part of that security policy. It then returns the indicated security label as a DB2SECURITYLABEL. You must use this function when inserting an existing security label into a column that has a data type of DB2SECURITYLABEL.

**Example:** Table T1 has two columns, the first has a data type of DB2SECURITYLABEL and the second has a data type of INTEGER. The security label named L1 is part of security policy P1. This SQL inserts the security label:

```
INSERT INTO T1 VALUES (SECLABEL_BY_NAME('P1', 'L1'), 22)
```

This SQL statement does not work:

```
INSERT INTO T1 VALUES (P1.L1, 22) // Syntax Error!
```

## SECLABEL\_TO\_CHAR

This built-in function returns a string representation of the values that make up a security label.

**Example:** Column C1 in table T1 has a data type of DB2SECURITYLABEL. T1 is protected by security policy P1, which has three security label components: level, departments, and groups. There is one row in T1 and the value in column C1 that has these elements for each of the components:

Component	Elements
level	SECRET
departments	DELTA and SIGMA
groups	G3

A user that has LBAC credentials that allow reading the row executes this SQL statement:

```
SELECT SECLABEL_TO_CHAR('P1', C1) AS C1 FROM T1
```

The output looks like this:

```
C1
'SECRET:(DELTA,SIGMA):G3'
```

---

## Protection of data using LBAC

Label-based access control (LBAC) can be used to protect rows of data, columns of data, or both. Data in a table can only be protected by security labels that are part of the security policy protecting the table. Data protection, including adding a security policy, can be done when creating the table or later by altering the table.

You can add a security policy to a table and protect data in that table as part of the same CREATE TABLE or ALTER TABLE statement.

As a general rule you are not allowed to protect data in such a way that your current LBAC credentials do not allow you to write to that data.

## **Adding a security policy to a table**

You can add a security policy to a table when you create the table by using the SECURITY POLICY clause of the CREATE TABLE statement. You can add a security policy to an existing table by using the ADD SECURITY POLICY clause of the ALTER TABLE statement. You do not need to have SECADM authority or have LBAC credentials to add a security policy to a table.

Adding a security policy to a table does not activate row or column protection by itself. It simply associates a security policy with the table which is to be used when row or column protection is activated. Refer to the “Protecting rows” and “Protecting columns” sections below for more information on how to activate row and column protection. The value of the PROTECTIONGRANULARITY column in the SYSCAT.TABLES catalog view indicates what level of LBAC protection is currently active for a table.

Security policies cannot be added to types of tables that cannot be protected by LBAC. See the overview of LBAC for a list of table types that cannot be protected by LBAC.

No more than one security policy can be added to any table.

## **Protecting rows**

You can allow protected rows in a new table by including a column with a data type of DB2SECURITYLABEL when you create the table. The CREATE TABLE statement must also add a security policy to the table. You do not need to have SECADM authority or have any LBAC credentials to create such a table.

You can allow protected rows in an existing table by adding a column that has a data type of DB2SECURITYLABEL. To add such a column, either the table must already be protected by a security policy or the ALTER TABLE statement that adds the column must also add a security policy to the table. When the column is added, the security label you hold for write access is used to protect all existing rows. If you do not hold a security label for write access that is part of the security policy protecting the table then you cannot add a column that has a data type of DB2SECURITYLABEL.

After a table has a column of type DB2SECURITYLABEL you protect each new row of data by storing a security label in that column. The details of how this works are described in the topics about inserting and updating LBAC protected data. You must have LBAC credentials to insert rows into a table that has a column of type DB2SECURITYLABEL.

A column that has a data type of DB2SECURITYLABEL cannot be dropped and cannot be changed to any other data type.

## **Protecting columns**

You can protect a column when you create the table by using the SECURED WITH column option of the CREATE TABLE statement. You can add protection to an existing column by using the SECURED WITH option in an ALTER TABLE statement.



To protect a column with a particular security label you must have LBAC credentials that allow you to write to data protected by that security label. You do not have to have SECADM authority.

Columns can only be protected by security labels that are part of the security policy protecting the table. You cannot protect columns in a table that has no security policy. You are allowed to protect a table with a security policy and protect one or more columns in the same statement.

You can protect any number of the columns in a table but a column can be protected by no more than one security label.

---

## Reading of LBAC protected data

When you try to read data protected by label-based access control (LBAC), your LBAC credentials for reading are compared to the security label that is protecting the data. If the protecting label does not block your credentials you are allowed to read the data.

In the case of a protected column the protecting security label is defined in the schema of the table. The protecting security label for that column is the same for every row in the table. In the case of a protected row the protecting security label is stored in the row in a column of type DB2SECURITYLABEL. It can be different for every row in the table.

The details of how your LBAC credentials are compared to a security label are given in the topic about how LBAC security labels are compared.

### Reading protected columns

When you try to read from a protected column your LBAC credentials are compared with the security label protecting the column. Based on this comparison access will either be blocked or allowed. If access is blocked then an error is returned and the statement fails. Otherwise, the statement proceeds as usual.

Trying to read a column that your LBAC credentials do not allow you to read, causes the entire statement to fail.

#### Example:

Table T1 has two protected columns. The column C1 is protected by the security label L1. The column C2 is protected by the security label L2.

Assume that user Jyoti has LBAC credentials for reading that allow access to security label L1 but not to L2. If Jyoti issues the following SQL statement, the statement will fail:

```
SELECT * FROM T1
```

The statement fails because column C2 is included in the SELECT clause as part of the wildcard (\*).

If Jyoti issues the following SQL statement it will succeed:

```
SELECT C1 FROM T1
```

The only protected column in the SELECT clause is C1, and Jyoti's LBAC credentials allow her to read that column.

## Reading protected rows

If you do not have LBAC credentials that allow you to read a row it is as if that row does not exist for you.

When you read protected rows, only those rows to which your LBAC credentials allow read access are returned. This is true even if the column of type DB2SECURITYLABEL is not part of the SELECT clause.

Depending on their LBAC credentials, different users might see different rows in a table that has protected rows. For example, two users executing the statement `SELECT COUNT(*) FROM T1` may get different results if T1 has protected rows and the users have different LBAC credentials.

Your LBAC credentials affect not only SELECT statements but also other SQL statements like UPDATE, and DELETE. If you do not have LBAC credentials that allow you to read a row, you cannot affect that row.

### Example:

Table T1 has these rows and columns. The column ROWSECURITYLABEL has a data type of DB2SECURITYLABEL.

*Table 16. Example values in table T1*

LASTNAME	DEPTNO	ROWSECURITYLABEL
Rjaibi	55	L2
Miller	77	L1
Fielding	11	L3
Bird	55	L2

Assume that user Dan has LBAC credentials that allow him to read data that is protected by security label L1 but not data protected by L2 or L3.

Dan issues the following SQL statement:

```
SELECT * FROM T1
```

The SELECT statement returns only the row for Miller. No error messages or warning are returned.

Dan's view of table T1 is this:

*Table 17. Example values in view of table T1*

LASTNAME	DEPTNO	ROWSECURITYLABEL
Miller	77	L1

The rows for Rjaibi, Fielding, and Bird are not returned because read access is blocked by their security labels. Dan cannot delete or update these rows. They will also not be included in any aggregate functions. For Dan it is as if those rows do not exist.

Dan issues this SQL statement:

```
SELECT COUNT(*) FROM T1
```

The statement returns a value of 1 because only the row for Miller can be read by the user Dan.

## Reading protected rows that contain protected columns

Column access is checked before row access. If your LBAC credentials for read access are blocked by the security label protecting one of the columns you are selecting then the entire statement fails. If not, the statement continues and only the rows protected by security labels to which your LBAC credentials allow read access are returned.

### Example

The column LASTNAME of table T1 is protected with the security label L1. The column DEPTNO is protected with security label L2. The column ROWSECURITYLABEL has a data type of DB2SECURITYLABEL. T1, including the data, looks like this:

Table 18. Example values in table T1

LASTNAME <i>Protected by L1</i>	DEPTNO <i>Protected by L2</i>	ROWSECURITYLABEL
Rjaibi	55	L2
Miller	77	L1
Fielding	11	L3

Assume that user Sakari has LBAC credentials that allow reading data protected by security label L1 but not L2 or L3.

Sakari issues this SQL statement:

```
SELECT * FROM T1
```

The statement fails because the SELECT clause uses the wildcard (\*) which includes the column DEPTNO. The column DEPTNO is protected by security label L2, which Sakari's LBAC credentials do not allow her to read.

Sakari next issues this SQL statement:

```
SELECT LASTNAME, ROWSECURITYLABEL FROM T1
```

The select clause does not include any columns that Sakari is not able to read so the statement continues. Only one row is returned, however, because each of the other rows is protected by security label L2 or L3.

Table 19. Example output from query on table T1

LASTNAME	ROWSECURITYLABEL
Miller	L1

---

## Inserting of LBAC protected data

When you try to insert data into a protected column, or to insert a new row into a table with protected rows, your LBAC credentials determine how that INSERT statement is handled.

## Inserting to protected columns

When you try to insert data into a protected column your LBAC credentials for writing are compared with the security label protecting that column. Based on this comparison access will either be blocked or allowed.

The details of how two security labels are compared are given in the topic about how LBAC security labels are compared.

If access is allowed, the statement proceeds as usual. If access is blocked, then the insert fails and an error is returned.

If you are inserting a row but do not provide a value for a protected column then a default value is inserted if one is available. This happens even if your LBAC credentials do not allow write access to that column. A default is available in the following cases:

- The column was declared with the WITH DEFAULT option
- The column is a generated column
- The column has a default value that is given through a BEFORE trigger
- The column has a data type of DB2SECURITYLABEL, in which case security label that you hold for write access is the default value

## Inserting to protected rows

When you insert a new row into a table with protected rows, you do not have to provide a value for the column that is of type DB2SECURITYLABEL. If you do not provide a value for that column, the column is automatically populated with the security label you have been granted for write access. If you have not been granted a security label for write access, an error is returned and the insert fails.

By using built-in functions like SECLABEL, you can explicitly provide a security label to be inserted in a column of type DB2SECURITYLABEL. The provided security label is only used, however, if your LBAC credentials would allow you to write to data that is protected with the security label you are trying to insert.

If you provide a security label that you would not be able to write, then what happens depends on the security policy that is protecting the table. If the security policy has the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option, then the insert fails and an error is returned. If the security policy does not have the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option or if it instead has the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option, then the security label you provide is ignored and if you hold a security label for write access, it is used instead. If you do not hold a security label for write access, an error is returned.

## Examples

Table T1 is protected by a security policy named P1 that was created without the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option. Table T1 has two columns but no rows. The columns are LASTNAME and LABEL. The column LABEL has a data type of DB2SECURITYLABEL.

User Joe holds a security label L2 for write access. Assume that the security label L2 allows him to write to data protected by security label L2 but not to data protected by security labels L1 or L3.

Joe issues the following SQL statement:

```
INSERT INTO T1 (LASTNAME, DEPTNO) VALUES ('Rjaibi', 11)
```

Because no security label was included in the INSERT statement, Joe's security label for write access is inserted into the LABEL row.

Table T1 now looks like this:

*Table 20. Values in the example table T1 after first INSERT statement*

LASTNAME	LABEL
Rjaibi	L2

Joe issues the following SQL statement, in which he explicitly provides the security label to be inserted into the column LABEL:

```
INSERT INTO T1 VALUES ('Miller', SECLABEL_BY_NAME('P1', 'L1'))
```

The SECLABEL\_BY\_NAME function in the statement returns a security label that is part of security policy P1 and is named L1. Joe is not allowed to write to data that is protected with L1 so he is not allowed to insert L1 into the column LABEL.

Because the security policy protecting T1 was created without the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option the security label that Joe holds for writing is inserted instead. No error or message is returned.

The table now looks like this:

*Table 21. Values in example table T1 after second INSERT statement*

LASTNAME	LABEL
Rjaibi	L2
Miller	L2

If the security policy protecting the table had been created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option then the insert would have failed and an error would have been returned.

Next Joe is granted an exemption to one of the LBAC rules. Assume that his new LBAC credentials allow him to write to data that is protected with security labels L1 and L2. The security label granted to Joe for write access does not change, it is still L2.

Joe issues the following SQL statement:

```
INSERT INTO T1 VALUES ('Bird', SECLABEL_BY_NAME('P1', 'L1'))
```

Because of his new LBAC credentials Joe is able to write to data that is protected by the security label L1. The insertion of L1 is therefore allowed. The table now looks like this:

*Table 22. Values in example table T1 after third INSERT statement*

LASTNAME	LABEL
Rjaibi	L2
Miller	L2
Bird	L1

---

## Updating of LBAC protected data

Your LBAC credentials must allow you write access to data before you can update it. In the case of updating a protected row, your LBAC credentials must also allow read access to the row.

### Updating protected columns

When you try to update data in a protected column, your LBAC credentials are compared to the security label protecting the column. The comparison made is for write access. If write access is blocked then an error is returned and the statement fails, otherwise the update continues.

The details of how your LBAC credentials are compared to a security label are given in the topic about how LBAC security labels are compared.

#### Example:

Assume there is a table T1 in which column DEPTNO is protected by a security label L2 and column Payscale is protected by a security label L3. T1, including its data, looks like this:

*Table 23. Table T1*

EMPNO	LASTNAME	DEPTNO <i>Protected by L2</i>	PAYSCALE <i>Protected by L3</i>
1	Rjaibi	11	4
2	Miller	11	7
3	Bird	11	9

User Lhakpa has no LBAC credentials. He issues this SQL statement:

```
UPDATE T1 SET EMPNO = 4
WHERE LASTNAME = "Bird"
```

This statement executes without error because it does not update any protected columns. T1 now looks like this:

*Table 24. Table T1 After Update*

EMPNO	LASTNAME	DEPTNO <i>Protected by L2</i>	PAYSCALE <i>Protected by L3</i>
1	Rjaibi	11	4
2	Miller	11	7
4	Bird	11	9

Lhakpa next issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 55
WHERE LASTNAME = "Miller"
```

This statement fails and an error is returned because DEPTNO is protected and Lhakpa has no LBAC credentials.

Assume Lhakpa is granted LBAC credentials and that allow the access summarized in the following table. The details of what those credentials are and what elements are in the security labels are not important for this example.

Security label protecting the data	Can read?	Can Write?
L2	No	Yes
L3	No	No

Lhakpa issues this SQL statement again:

```
UPDATE T1 SET DEPTNO = 55
WHERE LASTNAME = "Miller"
```

This time the statement executes without error because Lhakpa's LBAC credentials allow him to write to data protected by the security label that is protecting the column DEPTNO. It does not matter that he is not able to read from that same column. The data in T1 now looks like this:

*Table 25. Table T1 After Second Update*

EMPNO	LASTNAME	DEPTNO <i>Protected by</i> L2	PAYSCALE <i>Protected by</i> L3
1	Rjaibi	11	4
2	Miller	55	7
4	Bird	11	9

Next Lhakpa issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 55, PAYSCALE = 4
WHERE LASTNAME = "Bird"
```

The column PAYSCALE is protected by the security label L3 and Lhakpa's LBAC credentials do not allow him to write to it. Because Lhakpa is unable to write to the column, the update fails and no data is changed.

## Updating protected rows

If your LBAC credentials do not allow you to read a row, then it is as if that row does not exist for you so there is no way for you to update that row. For rows that you are able to read, you must also be able to write to the row in order to update it.

When you try to update a row, your LBAC credentials for writing are compared to the security label protecting the row. If write access is blocked, the update fails and an error is returned. If write access is not blocked, then the update continues.

The update that is performed is done the same way as an update to a non-protected row except for the treatment of the column that has a data type of DB2SECURITYLABEL. If you do not explicitly set the value of that column, it is automatically set to the security label that you hold for write access. If you do not have a security label for write access, an error is returned and the statement fails.

If the update explicitly sets the column that has a data type of DB2SECURITYLABEL, then your LBAC credentials are checked again. If the update you are trying to perform would create a row that your current LBAC credentials would not allow you to write to, then what happens depends on the security policy that is protecting the table. If the security policy has the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option, then the update fails and an error is returned. If the security policy does not have the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option or if it instead has the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option, then the security label you provide is ignored and if you hold a security label for write access, it is used instead. If you do not hold a security label for write access, an error is returned.

**Example:**

Assume that table T1 is protected by a security policy named P1 and has a column named LABEL that has a data type of DB2SECURITYLABEL.

T1, including its data, looks like this:

*Table 26. Table T1*

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	11	L1
2	Miller	11	L2
3	Bird	11	L3

Assume that user Jenni has LBAC credentials that allow her to read and write data protected by the security labels L0 and L1 but not data protected by any other security labels. The security label she holds for both read and write is L0. The details of her full credentials and of what elements are in the labels are not important for this example.

Jenni issues this SQL statement:

```
SELECT * FROM T1
```

Jenni sees only one row in the table:

*Table 27. Jenni's SELECT Query Result*

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	11	L1

The rows protected by labels L2 and L3 are not included in the result set because Jenni's LBAC credentials do not allow her to read those rows. For Jenni it is as if those rows do not exist.

Jenni issues these SQL statements:

```
UPDATE T1 SET DEPTNO = 44 WHERE DEPTNO = 11;
SELECT * FROM T1;
```

The result set returned by the query looks like this:



Table 28. Jenni's UPDATE & SELECT Query Result

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L0

The actual data in the table looks like this:

Table 29. Table T1

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L0
2	Miller	11	L2
3	Bird	11	L3

The statement executed without error but affected only the first row. The second and third rows are not readable by Jenni so they are not selected for update by the statement even though they meet the condition in the WHERE clause.

Notice that the value of the LABEL column in the updated row has changed even though that column was not explicitly set in the UPDATE statement. The column was set to the security label that Jenni held for writing.

Now Jenni is granted LBAC credentials that allow her to read data protected by any security label. Her LBAC credentials for writing do not change. She is still only able to write to data protected by L0 and L1.

Jenni again issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 44 WHERE DEPTNO = 11
```

This time the update fails because of the second and third rows. Jenni is able to read those rows, so they are selected for update by the statement. She is not, however, able to write to them because they are protected by security labels L2 and L3. The update does not occur and an error is returned.

Jenni now issues this SQL statement:

```
UPDATE T1
SET DEPTNO = 55, LABEL = SECLABEL_BY_NAME('P1', 'L2')
WHERE LASTNAME = "Rjaibi"
```

The SECLABEL\_BY\_NAME function in the statement returns the security label named L2. Jenni is trying to explicitly set the security label protecting the first row. Jenni's LBAC credentials allow her to read the first row, so it is selected for update. Her LBAC credentials allow her to write to rows protected by the security label L0 so she is allowed to update the row. Her LBAC credentials would not, however, allow her to write to a row protected by the security label L2, so she is not allowed to set the column LABEL to that value. The statement fails and an error is returned. No columns in the row are updated.

Jenni now issues this SQL statement:

```
UPDATE T1 SET LABEL = SECLABEL_BY_NAME('P1', 'L1') WHERE LASTNAME = "Rjaibi"
```

The statement succeeds because she would be able to write to a row protected by the security label L1.

T1 now looks like this:

Table 30. Table T1

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L1
2	Miller	11	L2
3	Bird	11	L3

## Updating protected rows that contain protected columns

If you try to update protected columns in a table with protected rows then your LBAC credentials must allow writing to all of the protected columns affected by the update, otherwise the update fails and an error is returned. This is as described in section about updating protected columns, earlier. If you are allowed to update all of the protected columns affected by the update you will still only be able to update rows that your LBAC credentials allow you to both read from and write to. This is as described in the section about updating protected rows, earlier. The handling of a column with a data type of DB2SECURITYLABEL is the same whether the update affects protected columns or not.

If the column that has a data type of DB2SECURITYLABEL is itself a protected column then your LBAC credentials must allow you to write to that column or you cannot update any of the rows in the table.

---

## Deleting or dropping of LBAC protected data

Your ability to delete data in tables protected by LBAC depend on your LBAC credentials.

### Deleting protected rows

If your LBAC credentials do not allow you to read a row, it is as if that row does not exist for you so there is no way for you to delete it. To delete a row that you are able to read, your LBAC credentials must also allow you to write to the row. To delete any row in a table that has protected columns you must have LBAC credentials that allow you to write to all protected columns in the table.

When you try to delete a row, your LBAC credentials for writing are compared to the security label protecting the row. If the protecting security label blocks write access by your LBAC credentials, the DELETE statement fails, an error is returned, and no rows are deleted.

*Example*

Protected table T1 has these rows:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2
Fielding	77	L3

Assume that user Pat has LBAC credentials such that her access is as summarized in this table:

Security label	Read access?	Write access?
L1	Yes	Yes
L2	Yes	No
L3	No	No

The exact details of her LBAC credentials and of the security labels are unimportant for this example.

Pat issues the following SQL statement:

```
SELECT * FROM T1 WHERE DEPTNO != 999
```

The statement executes and returns this result set:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2

The last row of T1 is not included in the results because Pat does not have read access to that row. It is as if that row does not exist for Pat.

Pat issues this SQL statement:

```
DELETE FROM T1 WHERE DEPTNO != 999
```

Pat does not have write access to the first or third row, both of which are protected by L2. So even though she can read the rows she cannot delete them. The DELETE statement fails and no rows are deleted.

Pat issues this SQL statement:

```
DELETE FROM T1 WHERE DEPTNO = 77;
```

This statement succeeds because Pat is able to write to the row with Miller in the LASTNAME column. That is the only row selected by the statement. The row with Fielding in the LASTNAME column is not selected because Pat's LBAC credentials do not allow her to read that row. That row is never considered for the delete so no error occurs.

The actual rows of the table now look like this:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Bird	55	L2
Fielding	77	L3

## Deleting rows that have protected columns

To delete any row in a table that has protected columns you must have LBAC credentials that allow you to write to all protected columns in the table. If there is any row in the table that your LBAC credentials do not allow you to write to then the delete will fail and an error will be returned.

If the table has both protected columns and protected rows then to delete a particular row you must have LBAC credentials that allow you to write to every protected column in the table and also to read from and write to the row that you want to delete.

*Example*

In protected table T1, the column DEPTNO is protected by the security label L2. T1 contains these rows:

<b>LASTNAME</b>	<b>DEPTNO</b> <i>Protected by L2</i>	<b>LABEL</b>
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2
Fielding	77	L3

Assume that user Benny has LBAC credentials that allow him the access summarized in this table:

<b>Security label</b>	<b>Read access?</b>	<b>Write access?</b>
L1	Yes	Yes
L2	Yes	No
L3	No	No

The exact details of his LBAC credentials and of the security labels are unimportant for this example.

Benny issues the following SQL statement:

```
DELETE FROM T1 WHERE DEPTNO = 77
```

The statement fails because Benny does not have write access to the column DEPTNO.

Now Benny's LBAC credentials are changed so that he has access as summarized in this table:

<b>Security label</b>	<b>Read access?</b>	<b>Write access?</b>
L1	Yes	Yes
L2	Yes	Yes
L3	Yes	No

Benny issues this SQL statement again:

```
DELETE FROM T1 WHERE DEPTNO = 77
```

This time Benny has write access to the column DEPTNO so the delete continues. The delete statement selects only the row that has a value of Miller in the LASTNAME column. The row that has a value of Fielding in the LASTNAME column is not selected because Benny's LBAC credentials do not allow him to read that row. Because the row is not selected for deletion by the statement it does not matter that Benny is unable to write to the row.

The one row selected is protected by the security label L1. Benny's LBAC credentials allow him to write to data protected by L1 so the delete is successful.

The actual rows in table T1 now look like this:

LASTNAME	DEPTNO <i>Protected by L2</i>	LABEL
Rjaibi	55	L2
Bird	55	L2
Fielding	77	L3

## Dropping protected data

You cannot drop a column that is protected by a security label unless your LBAC credentials allow you to write to that column.

A column with a data type of DB2SECURITYLABEL cannot be dropped from a table. To remove it you must first drop the security policy from the table. When you drop the security policy the table is no longer protected with LBAC and the data type of the column is automatically changed from DB2SECURITYLABEL to VARCHAR(128) FOR BIT DATA. The column can then be dropped.

Your LBAC credentials do not prevent you from dropping entire tables or databases that contain protected data. If you would normally have permission to drop a table or a database you do not need any LBAC credentials to do so, even if the database contains protected data.

---

## Removal of LBAC protection from data

You must have SECADM authority to remove the security policy from a table. To remove the security policy from a table you use the DROP SECURITY POLICY clause of the ALTER TABLE statement. This also automatically removes protection from all rows and all columns of the table.

### Removing protection from rows

In a table that has protected rows every row must be protected by a security label. There is no way to remove LBAC protection from individual rows.

A column of type DB2SECURITYLABEL cannot be altered or removed except by removing the security policy from the table.

### Removing protection from columns

Protection of a column can be removed using the DROP COLUMN SECURITY clause of the SQL statement ALTER TABLE. To remove the protection from a column you must have LBAC credentials that allow you to read from and write to that column in addition to the normal privileges and authorities needed to alter a table.



---

## Chapter 6. Using the system catalog for security information

Information about each database is automatically maintained in a set of views called the system catalog, which is created when the database is created. This system catalog describes tables, columns, indexes, programs, privileges, and other objects.

The following views and table functions list information about privileges held by users, identities of users granting privileges, and object ownership:

### **SYSCAT.COLAUTH**

Lists the column privileges

### **SYSCAT.DBAUTH**

Lists the database privileges

### **SYSCAT.INDEXAUTH**

Lists the index privileges

### **SYSCAT.MODULEAUTH**

Lists the module privileges

### **SYSCAT.PACKAGEAUTH**

Lists the package privileges

### **SYSCAT.PASSTHRUAUTH**

Lists the server privilege

### **SYSCAT.ROLEAUTH**

Lists the role privileges

### **SYSCAT.ROUTINEAUTH**

Lists the routine (functions, methods, and stored procedures) privileges

### **SYSCAT.SCHEMAAUTH**

Lists the schema privileges

### **SYSCAT.SEQUENCEAUTH**

Lists the sequence privileges

### **SYSCAT.SURROGATEAUTHIDS**

Lists the authorization IDs for which another authorization ID can act as a surrogate.

### **SYSCAT.TABAUTH**

Lists the table and view privileges

### **SYSCAT.TBSPACEAUTH**

Lists the table space privileges

### **SYSCAT.VARIABLEAUTH**

Lists the variable privileges

### **SYSCAT.WORKLOADAUTH**

Lists the workload privileges

### **SYSCAT.XSROBJECTAUTH**

Lists the XSR object privileges

Privileges granted to users by the system will have SYSIBM as the grantor. SYSADM, SYSAINT, SYSCTRL, and SYSMON are not listed in the system catalog.

The CREATE and GRANT statements place privileges in the system catalog. Users with ACCESSCTRL and SECADM authority can grant and revoke SELECT privilege on the system catalog views.

---

## Retrieving authorization names with granted privileges

You can use the PRIVILEGES and other administrative views to retrieve information about the authorization names that have been granted privileges in a database.

### About this task

For example, the following query retrieves all explicit privileges and the authorization IDs to which they were granted, plus other information, from the PRIVILEGES administrative view:

```
SELECT AUTHID, PRIVILEGE, OBJECTNAME, OBJECTSCHEMA, OBJECTTYPE
FROM SYSIBMADM.PRIVILEGES
```

The following query uses the AUTHORIZATIONIDS administrative view to find all the authorization IDs that have been granted privileges or authorities, and to show their types:

```
SELECT AUTHID, AUTHIDTYPE FROM SYSIBMADM.AUTHORIZATIONIDS
```

You can also use the SYSIBMADM.OBJECTOWNERS administrative view and the SYSPROC.AUTH\_LIST\_GROUPS\_FOR\_AUTHID table function to find security-related information.

Prior to Version 9.1, no single system catalog view contained information about all privileges. For releases earlier than version 9.1, the following statement retrieves all authorization names with privileges:

```
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'DATABASE' FROM SYSCAT.DBAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'TABLE ' FROM SYSCAT.TABAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'PACKAGE ' FROM SYSCAT.PACKAGEAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'INDEX ' FROM SYSCAT.INDEXAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'COLUMN ' FROM SYSCAT.COLAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SCHEMA ' FROM SYSCAT.SCHEMAAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SERVER ' FROM SYSCAT.PASSTHROUGHAUTH
ORDER BY GRANTEE, GRANTEETYPE, 3
```

Periodically, the list retrieved by this statement should be compared with lists of user and group names defined in the system security facility. You can then identify those authorization names that are no longer valid.

**Note:** If you are supporting remote database clients, it is possible that the authorization name is defined at the remote client only and not on your database server machine.



---

## Retrieving all names with DBADM authority

The following statement retrieves all authorization names that have been directly granted DBADM authority:

### About this task

```
SELECT DISTINCT GRANTEE, GRANTEETYPE FROM SYSCAT.DBAUTH
WHERE DBADMAUTH = 'Y'
```

---

## Retrieving names authorized to access a table

You can use the PRIVILEGES and other administrative views to retrieve information about the authorization names that have been granted privileges in a database.

### About this task

The following statement retrieves all authorization names (and their types) that are directly authorized to access the table EMPLOYEE with the qualifier JAMES:

```
SELECT DISTINCT AUTHID, AUTHIDTYPE FROM SYSIBMADM.PRIVILEGES
WHERE OBJECTNAME = 'EMPLOYEE' AND OBJECTSCHEMA = 'JAMES'
```

For releases earlier than Version 9.1, the following query retrieves the same information:

```
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.TABAUTH
WHERE TABNAME = 'EMPLOYEE'
AND TABSCHEMA = 'JAMES'
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.COLAUTH
WHERE TABNAME = 'EMPLOYEE'
AND TABSCHEMA = 'JAMES'
```

To find out who can update the table EMPLOYEE with the qualifier JAMES, issue the following statement:

```
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.TABAUTH
WHERE TABNAME = 'EMPLOYEE' AND TABSCHEMA = 'JAMES' AND
(CONTROLAUTH = 'Y' OR
UPDATEAUTH IN ('G','Y'))
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.DBAUTH
WHERE DBADMAUTH = 'Y'
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.COLAUTH
WHERE TABNAME = 'EMPLOYEE' AND TABSCHEMA = 'JAMES' AND
PRIVTYPE = 'U'
```

This retrieves any authorization names with DBADM authority, as well as those names to which CONTROL or UPDATE privileges have been directly granted.

Remember that some of the authorization names may be groups, not just individual users.

---

## Retrieving all privileges granted to users

By making queries on the system catalog views, users can retrieve a list of the privileges they hold and a list of the privileges they have granted to other users.

## About this task

You can use the PRIVILEGES and other administrative views to retrieve information about the authorization names that have been granted privileges in a database. For example, the following query retrieves all the privileges granted to the current session authorization ID:

```
SELECT * FROM SYSIBMADM.PRIVILEGES
WHERE AUTHID = SESSION_USER AND AUTHIDTYPE = 'U'
```

The keyword SESSION\_USER in this statement is a special register that is equal to the value of the current user's authorization name.

For releases earlier than Version 9.1, the following examples provide similar information. For example, the following statement retrieves a list of the database privileges that have been directly granted to the individual authorization name JAMES:

```
SELECT * FROM SYSCAT.DBAUTH
WHERE GRANTEE = 'JAMES' AND GRANTEETYPE = 'U'
```

The following statement retrieves a list of the table privileges that were directly granted by the user JAMES:

```
SELECT * FROM SYSCAT.TABAUTH
WHERE GRANTOR = 'JAMES'
```

The following statement retrieves a list of the individual column privileges that were directly granted by the user JAMES:

```
SELECT * FROM SYSCAT.COLAUTH
WHERE GRANTOR = 'JAMES'
```

---

## Securing the system catalog view

Because the system catalog views describe every object in the database, if you have sensitive data, you might want to restrict their access.

### About this task

The following authorities have SELECT privilege on all catalog tables:

- ACCESSCTRL
- DATAACCESS
- DBADM
- SECADM
- SQLADM

In addition, the following instance level authorities have the ability to select from SYSCAT.BUFFERPOOLS, SYSCAT.DBPARTITIONGROUPS, SYSCAT.DBPARTITIONGROUPDEF, SYSCAT.PACKAGES, and SYSCAT.TABLES:

- SYSADM
- SYSCtrl
- SYSMAINT
- SYSMON

You can use the CREATE DATABASE ... RESTRICTIVE command to create a database in which no privileges are automatically granted to PUBLIC. In this case, none of the following normal default grant actions occur:

- CREATETAB
- BINDADD
- CONNECT
- IMPLICIT\_SCHEMA
- EXECUTE with GRANT on all procedures in schema SQLJ
- EXECUTE with GRANT on all functions and procedures in schema SYSPROC
- BIND on all packages created in the NULLID schema
- EXECUTE on all packages created in the NULLID schema
- CREATEIN on schema SQLJ
- CREATEIN on schema NULLID
- USE on table space USERSPACE1
- SELECT access to the SYSIBM catalog tables
- SELECT access to the SYSCAT catalog views
- SELECT access to the SYSIBMADM administrative views
- SELECT access to the SYSSTAT catalog views
- UPDATE access to the SYSSTAT catalog views

If you have created a database using the RESTRICTIVE option, no permissions are granted to PUBLIC. You can run the following query to verify that no schemas are accessibly by PUBLIC:

```
SELECT DISTINCT OBJECTSCHEMA FROM SYSIBMADM.PRIVILEGES WHERE AUTHID='PUBLIC'
OBJECTSCHEMA

```

For releases earlier than Version 9.1 of the Db2 database manager, during database creation, SELECT privilege on the system catalog views is granted to PUBLIC. In most cases, this does not present any security problems. For very sensitive data, however, it may be inappropriate, as these tables describe every object in the database. If this is the case, consider revoking the SELECT privilege from PUBLIC; then grant the SELECT privilege as required to specific users. Granting and revoking SELECT on the system catalog views is done in the same way as for any view, but you must have either ACCESSCTRL or SECADM authority to do this.

At a minimum, if you don't want any user to be able to know what objects other users have access to, you should consider restricting access to the following catalog and administrative views:

- SYSCAT.COLAUTH
- SYSCAT.DBAUTH
- SYSCAT.INDEXAUTH
- SYSCAT.PACKAGEAUTH
- SYSCAT.PASSTHRUAUTH
- SYSCAT.ROUTINEAUTH
- SYSCAT.SCHEMAAUTH
- SYSCAT.SECURITYLABELACCESS
- SYSCAT.SECURITYPOLICYEXEMPTIONS
- SYSCAT.SEQUENCEAUTH
- SYSCAT.SURROGATEAUTHIDS
- SYSCAT.TABAUTH
- SYSCAT.TBSPACEAUTH

- SYSCAT.XSROBJECTAUTH
- SYSIBMADM.AUTHORIZATIONIDS
- SYSIBMADM.OBJECTOWNERS
- SYSIBMADM.PRIVILEGES

This would prevent information about user privileges from becoming available to everyone with access to the database.

You should also examine the columns for which statistics are gathered. Some of the statistics recorded in the system catalog contain data values which could be sensitive information in your environment. If these statistics contain sensitive data, you may want to revoke SELECT privilege from PUBLIC for the SYSCAT.COLUMNS and SYSCAT.COLDIST catalog views.

If you want to limit access to the system catalog views, you could define views to let each authorization name retrieve information about its own privileges.

For example, the following view MYSELECTS includes the owner and name of every table on which a user's authorization name has been directly granted SELECT privilege:

```
CREATE VIEW MYSELECTS AS
 SELECT TABSCHEMA, TABNAME FROM SYSCAT.TABAUTH
 WHERE GRANTEETYPE = 'U'
 AND GRANTEE = USER
 AND SELECTAUTH = 'Y'
```

The keyword USER in this statement is equal to the value of the current session authorization name.

The following statement makes the view available to every authorization name:

```
GRANT SELECT ON TABLE MYSELECTS TO PUBLIC
```

And finally, remember to revoke SELECT privilege on the view and base table by issuing the following two statements:

```
REVOKE SELECT ON TABLE SYSCAT.TABAUTH FROM PUBLIC
REVOKE SELECT ON TABLE SYSIBM.SYSTABAUTH FROM PUBLIC
```

---

## Chapter 7. Firewall support

A *firewall* is a set of related programs, located at a network gateway server, that are used to prevent unauthorized access to a system or network.

There are four types of firewalls:

1. Network level, packet-filter, or screening router firewalls
2. Classical application level proxy firewalls
3. Circuit level or transparent proxy firewalls
4. Stateful multi-layer inspection (SMLI) firewalls

There are existing firewall products that incorporate one of the firewall types listed previously. There are many other firewall products that incorporate some combination of the types listed previously.

---

### Screening router firewalls

The screening router firewall is also known as a network level or packet-filter firewall. Such a firewall works by screening incoming packets by protocol attributes. The protocol attributes screened may include source or destination address, type of protocol, source or destination port, or some other protocol-specific attributes.

For all firewall solutions (except SOCKS), you need to ensure that all the ports used by Db2 database are open for incoming and outgoing packets. Db2 database uses port 523 for the Db2 Administration Server (DAS), which is used by the Db2 database tools. Determine the ports used by all your server instances by using the services file to map the service name in the server database manager configuration file to its port number.

In addition, for partitioned database environments and Db2 pureScale environments, connections must be allowed on all non-privileged ports between members of the same Db2 instance. Non-privileged ports have port numbers greater than or equal to 1024.

---

### Application proxy firewalls

A proxy or proxy server is a technique that acts as an intermediary between a Web client and a Web server. A proxy firewall acts as a gateway for requests arriving from clients.

When client requests are received at the firewall, the final server destination address is determined by the proxy software. The application proxy translates the address, performs additional access control checking and logging as necessary, and connects to the server on behalf of the client.

The Db2 Connect product on a firewall machine can act as a proxy to the destination server. Also, a Db2 database server on the firewall, acting as a hop server to the final destination server, acts like an application proxy.

---

## Circuit level firewalls

The circuit level firewall is also known as a transparent proxy firewall.

A transparent proxy firewall does not modify the request or response beyond what is required for proxy authentication and identification. An example of a transparent proxy firewall is SOCKS.

The Db2 database system supports SOCKS Version 4.

---

## Stateful multi-layer inspection (SMLI) firewalls

The stateful multi-layer inspection (SMLI) firewall uses a sophisticated form of packet-filtering that examines all seven layers of the Open System Interconnection (OSI) model.

Each packet is examined and compared against known states of friendly packets. While screening router firewalls only examine the packet header, SMLI firewalls examine the entire packet including the data.

---

## Chapter 8. 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.

### Group retrieval plug-in

Retrieves group membership information for a particular user.

### User ID/password authentication plug-in

The following authentication types are implemented using a user ID and password authentication plug-in:

- CLIENT
- SERVER
- SERVER\_ENCRYPT
- DATA\_ENCRYPT
- DATA\_ENCRYPT\_CMP

These authentication types determine how and where authentication of a user occurs. The authentication type that is used is determined by the following method:

- For connect or attach operations, if you specify a value for the **srvcon\_auth** configuration parameter, then that value takes precedence over the value of the **authentication** configuration parameter.
- In all other cases, the value of the **authentication** configuration parameter is used.

### GSS-API authentication plug-in

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). The Kerberos protocol is the predominant means of implementing the GSS-API authentication mechanism. 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. Client-side Kerberos support is available on Solaris, AIX, HP-UX (64-bit only), Windows, and Linux operating systems. For Windows operating systems, Kerberos support is enabled by default.

The Db2 database manager supports these plug-ins at both the client and the server.

**Note:** Authentication types determine how and where a user is authenticated. To use a particular authentication type, set the value of the **authentication** database manager configuration parameter.

You can use each of the plug-ins independently, or with the other plug-ins. For example, you might use a specific sever-side authentication plug-in, but accept the Db2 default values for client and group authentication. Alternatively, you might have only a group retrieval, or a client authentication plug-in, but without a server-side plug-in.

If you want to use GSS-API authentication, plug-ins are required on both the client and the server.

The default behavior for authentication is to use a user ID/password plug-in that implements an operating-system-level mechanism to authenticate.

The Db2 database product includes plug-ins for group retrieval, user ID/password authentication, and GSS-API authentication. You can customize Db2 client and server authentication behavior further by developing your own plug-ins, or by purchasing plug-ins from a third party.

## Deployment of security plug-ins on Db2 clients

Db2 clients can support one group retrieval plug-in and one user ID/password authentication plug-in.

Alternatively, clients using GSS-API authentication plug-in determine which plug-in to use by scanning the list of implemented GSS-API plug-ins on the Db2 server. The first authentication plug-in name that matches a GSS-API authentication plug-in implemented on the client is the one chosen. You specify the list of implemented server GSS-API plug-ins using the **srvcon\_gssplugin\_list** database manager configuration parameter. The following figure portrays the security plug-in infrastructure on a Db2 client:

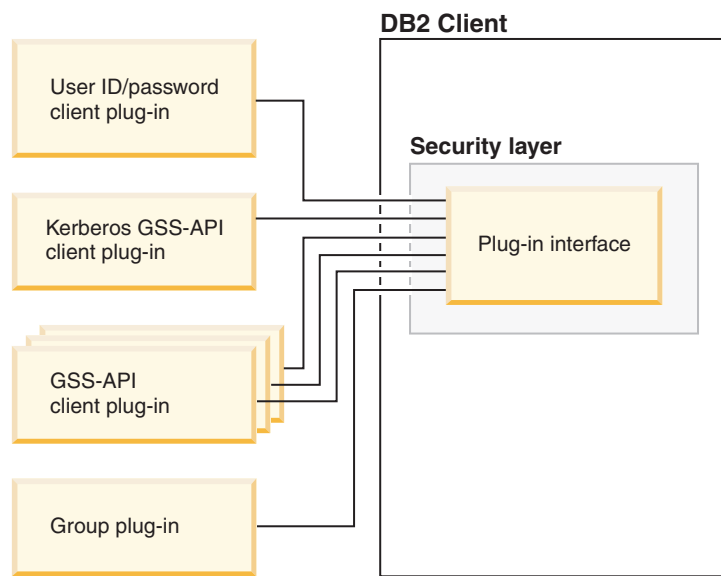


Figure 5. Deploying Security Plug-ins on Db2 Clients

## Deployment of security plug-ins on Db2 servers

Db2 servers can support one group retrieval plug-in, one user ID/password authentication plug-in, and multiple GSS-API plug-ins. You can specify the



available GSS-API plug-ins as a list of values for the **srvcon\_gssplugin\_list** database manager configuration parameter. However, only one GSS-API plug-in in this list can be a Kerberos plug-in.

In addition to deploying the server-side security plug-ins on your database server, you might have to deploy client authentication plug-ins on your database server. When you run instance-level operations, such as the **db2start** and **db2trc** commands, the Db2 database manager performs authorization checking for these operations using client authentication plug-ins. Therefore, you might need to install the client authentication plug-in that corresponds to the authentication plug-in on the server. This plug-in name is specified by the **authentication** database manager configuration parameter on the server.

You can set the **authentication** and **srvcon\_auth** configuration parameters to different values. This scenario causes one mechanism to be used to authenticate database connections and the other mechanism to be used for local authorization.

The most common method for this approach is to:

- Set the **srvcon\_auth** configuration parameter to GSSPLUGIN; and
- Set the **authentication** configuration parameter to SERVER.

The **srvcon\_auth** configuration parameter is a means to override the authentication type used by incoming connections. These connections use the authentication method specified by the **srvcon\_auth** configuration parameter, but if this value is left empty, the value of the **authentication** parameter is used instead.

If you do not use client authentication plug-ins on the database server, instance-level operations, such as the **db2start** command, fail.

The following figure outlines the security authentication plug-in infrastructure on a Db2 server:

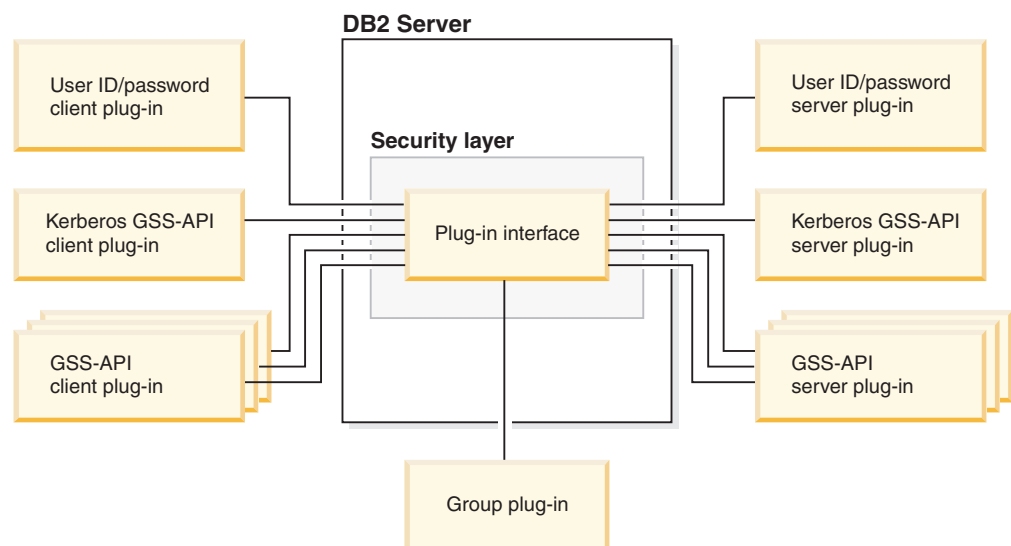


Figure 6. Deploying Security Plug-ins on Db2 Servers

## Enabling security plug-ins

You can specify the plug-ins to use for each authentication mechanism by setting database manager configuration parameters. The following table outlines these parameters:

*Table 31. Database Manager configuration parameters for security authentication plug-ins*

Description	Parameter name
Client Userid-Password Plugin	CLNT_PW_PLUGIN
Client Kerberos Plugin	CLNT_KRB_PLUGIN
Group Plugin	GROUP_PLUGIN
GSS Plugin for Local Authorization	LOCAL_GSSPLUGIN
Server Plugin Mode	SRV_PLUGIN_MODE
Server List of GSS Plugins	SRVCON_GSSPLUGIN_LIST
Server Userid-Password Plugin	SRVCON_PW_PLUGIN
Server Connection Authentication	SRVCON_AUTH
Database manager authentication	AUTHENTICATION

If you do not set the values for these parameters, the default plug-ins that the Db2 product supplies are used for group retrieval, user ID/password management, and Kerberos authentication (if the **authentication** parameter is set to KERBEROS on the server). However, a default GSS-API plug-in is not provided. Therefore, if you specify an authentication type of GSSPLUGIN for the **authentication** parameter, you must also specify a GSS-API authentication plug-in for the **srvcon\_gssplugin\_list** configuration parameter.

## Loading security plug-ins

All of the supported plug-ins that are identified by the database manager configuration parameters are loaded when the database manager starts.

During connect or attach operations, the Db2 client loads a plug-in that is appropriate for the security mechanism that the client negotiated with the server. 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. In this scenario, the client program makes an initial connection to server A that uses a GSS-API plug-in (G1). Server A sends a list of supported plug-ins to the client, and the matching G1 plug-in is loaded on the client. The client program then has another thread, which connects to server B that uses a GSS-API plug-in (G2). The client is informed about G2, which is then loaded, and now both G1 and G2 plug-ins are simultaneously in use on the client.

Actions other than connect or attach operations (such as updating the database manager configuration, starting and stopping the database manager, or turning a Db2 trace on and off) also require an authorization mechanism. For such actions, the Db2 client program loads a plug-in that is specified by another database manager configuration parameter:

- If you set the **authentication** configuration parameter to GSSPLUGIN, the Db2 database manager uses the plug-in specified by the **local\_gssplugin** configuration parameter.

- If you set the **authentication** configuration parameter to KERBEROS, the Db2 database manager uses the plug-in specified by the **cInt\_krb\_plugin** configuration parameter.
- Otherwise, the Db2 database manager uses the plug-in specified by the **cInt\_pw\_plugin** configuration parameter.

Security plug-ins are supported for connections made to the database server over both IPv4 and IPv6 address protocols.

## Developing security plug-ins

If you are developing a security authentication plug-in, you must implement the standard authentication functions used by the Db2 database manager. The functionality that you must implement for the three types of plug-ins:

### Group retrieval plug-in

- Find and return the list of groups to which a user belongs

### User ID/password authentication plug-in

- Identify the default security context (for a client plug-in only)
- Validate and, optionally, change a password
- Determine whether a particular string represents a valid user (for a server plug-in only)
- Modify the user ID or password that is provided on the client before it is sent to the server (for a client plug-in only)
- Return the Db2 authorization ID that is associated with a particular user

### GSS-API authentication plug-in

- Identify the default security context (for a client plug-in only)
- Implement the required GSS-API functions
- Generate initial credentials based on a user ID and password and, optionally, change a password (for a client plug-in only)
- Create and accept security tickets
- Return the Db2 authorization ID that is associated with a particular GSS-API security context

You can pass a user ID of up to 255 characters for a connect statement that you issue through the CLP or via a dynamic SQL statement.

**Important:** The integrity of your Db2 database system installation can be compromised if security plug-ins are not adequately coded, reviewed, and tested. The Db2 database product takes precautions against many common types of failures, but it cannot guarantee complete integrity if user-written security plug-ins are deployed.

---

## 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

**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

**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 F00somemethod, the plug-in could have a name like F00somemethod.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 32. 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.

---

## 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 specify the 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.

---

## 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 `sql1lib/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.



---

## Enabling plug-ins

### 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.

#### Before you begin

After you acquire a group retrieval plug-in that is suitable for your database management system, you can deploy it.

#### Procedure

- 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.

#### Before you begin

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.

## Procedure

- 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.

### Before you begin

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.

## Procedure

- 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 **srvcon\_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 **srvcon\_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 **srvcon\_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 Kerberos authentication behavior of the Db2 security system, you can develop your own Kerberos authentication plug-ins or purchase one from a third party.

### Before you begin

If you want to deploy a new version of an existing plug-in, you must stop the Db2 server and any applications using the plug-in. Undefined behaviors, including traps, occur if a process is using a plug-in when you deploy a new version of that plug-in (with the same name).

### About this task

The Kerberos authentication plug-in can be deployed on a database server or a database client.

### Procedure

- 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 into the server plug-in directory.
  2. Update the setting of the **srvcon\_gssplugin\_list** database manager configuration parameter, which is 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 there is no Kerberos plug-in in the list, an error is returned. If there is more than one Kerberos plug-in in the list, an error is returned. If the configuration parameter value is blank and the **authentication** configuration parameter is set to KERBEROS or KRB\_SVR\_ENCRYPT, the default Db2 Kerberos plug-in, IBMkrb5, is used.
  3. If necessary, set the value of the **srvcon\_auth** database manager configuration parameter. If you want to deploy a Kerberos plug-in, the acceptable values for the **srvcon\_auth** database manager configuration parameter are as follows:
    - KERBEROS
    - KRB\_SERVER\_ENCRYPT
    - GSSPLUGIN
    - GSS\_SERVER\_ENCRYPT
    - Blank, but only if the **authentication** configuration parameter is set to one of the previous values in this list.
- To deploy a Kerberos authentication plug-in on a database client, perform the following steps on the client:
  1. Copy the Kerberos authentication plug-in library into the client plug-in directory.
  2. Set the **clnt\_krb\_plugin** database manager configuration parameter to the name of the Kerberos plug-in. If the value of the **clnt\_krb\_plugin** configuration parameter is blank, the client cannot use Kerberos authentication. On Windows, the default value is IBMkrb5. It only needs to be altered for a customized Kerberos plugin. On UNIX, the value must be set since the default value is blank. 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. Set the **cInt\_krb\_plugin** database manager configuration parameter to 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 use only a Kerberos authentication plug-in. The following example catalogs the testdb database:

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION KERBEROS
 TARGET PRINCIPAL service/host@REALM
```

---

## LDAP-based authentication and group lookup support

The Db2 database manager and Db2 Connect support LDAP-based authentication and group lookup functionality through the use of LDAP security plug-in modules and also through transparent LDAP

LDAP-based authentication support has been enhanced on the AIX operating system. Starting with Db2 V9.7 Fix Pack 1, transparent LDAP support has also been extended to Linux operating systems at the same version levels that the Db2 product supports. LDAP now enables central management of user authentication and group membership using transparent LDAP authentication. You can configure Db2 instances to authenticate users and acquire their groups through the operating system. The operating system will, in turn, perform the authentication through an LDAP server. To enable transparent LDAP authentication, set the **DB2AUTH** miscellaneous registry variable to 0SAUTHDB. Supported operating systems are:

- AIX
- Linux

Another option for implementing LDAP-based authentication is through the use of LDAP security plug-ins. LDAP security plug-in modules allow the Db2 database manager to authenticate users defined in an LDAP directory, removing the requirement that users and groups be defined to the operating system at the same version levels that the Db2 product supports. Supported operating systems are:

- AIX
- Linux on IA32, x64, or zSeries hardware
- Windows

Supported LDAP servers for use with security plug-in modules are:

- IBM Lotus® Domino® LDAP Server, Version 8.0, and later
- IBM Tivoli® Directory Server (ITDS) Version 6.2 (with GSKit 7.0.4.20 and later), and later
- Microsoft Active Directory (MSAD) Version 2008, and later
- Novell eDirectory, Version 8.8, and later
- OpenLDAP server, Version 2.4, and later
- Sun Java System Directory Server Enterprise Edition, Version 5.2 FP4, and later
- z/OS Integrated Security Services LDAP Server Version V1R6, and later

**Note:** When you use the LDAP plug-in modules, all users associated with the database must be defined on the LDAP server. This includes both the Db2 instance owner ID as well as the fenced user. (These users are typically defined in the

operating system, but must also be defined in LDAP.) Similarly, if you use the LDAP group plug-in module, any groups required for authorization must be defined on the LDAP server. This includes the SYSADM, SYSMAINT, SYSCTRL and SYSMON groups defined in the database manager configuration.

Db2 security plug-in modules are available for server-side authentication, client-side authentication and group lookup, described later. Depending on your specific environment, you may need to use one, two or all three types of plug-in.

To use Db2 security plug-in modules, follow these steps:

1. Decide if you need server, client, or group plug-in modules, or a combination of these modules.
2. Configure the plug-in modules by setting values in the IBM LDAP security plug-in configuration file (default name is `IBMLDAPSecurity.ini`). You will need to consult with your LDAP administrator to determine appropriate values.
3. Enable the plug-in modules
4. Test connecting with various LDAP User IDs.

## Server authentication plugin

The server authentication plug-in module performs server validation of user IDs and passwords supplied by clients on `CONNECT` and `ATTACH` statements. It also provides a way to map LDAP user IDs to Db2 authorization IDs, if required. The server plug-in module is generally required if you want users to authenticate to the Db2 database manager using their LDAP user ID and password.

## Client authentication plug-in

The client authentication plug-in module is used where user ID and password validation occurs on the client system; that is, where the Db2 server is configured with `SRVCON_AUTH` or `AUTHENTICATION` settings of `CLIENT`. The client validates any user IDs and passwords supplied on `CONNECT` or `ATTACH` statements, and sends the user ID to the Db2 server. Note that `CLIENT` authentication is difficult to secure, and not generally recommended.

The client authentication plug-in module may also be required if the local operating system user IDs on the database server are different from the Db2 authorization IDs associated with those users. You can use the client-side plugin to map local operating system user IDs to Db2 authorization IDs before performing authorization checks for local commands on the database server, such as `for:db2start`.

## Group lookup plug-in

The group lookup plug-in module retrieves group membership information from the LDAP server for a particular user. It is required if you want to use LDAP to store your group definitions. The most common scenario is where:

- All users and groups are defined in the LDAP server
- Any users defined locally on the database server are also defined with the same user ID on the LDAP server (including the instance owner and the fenced user)
- Password validation occurs on the Db2 server (that is, an `AUTHENTICATION` or `SRVCON_AUTH` value of `SERVER`, `SERVER_ENCRYPT` or `DATA_ENCRYPT` is set in the server DBM config file).

It is generally sufficient to install only the server authentication plug-in module and the group lookup plug-in module on the server. Db2 clients typically do not need to have the LDAP plug-in module installed.

It is possible to use only the LDAP group lookup plug-in module in combination with some other form of authentication plug-in (such as Kerberos). In this case, the LDAP group lookup plug-in module will be provided the Db2 authorization IDs associated with a user. The plug-in module searches the LDAP directory for a user with a matching AUTHID\_ATTRIBUTE, then retrieves the groups associated with that user object.

## Configuring transparent LDAP for authentication and group lookup (AIX)

Starting in Db2 V9.7, transparent LDAP-based authentication and group look up are supported on the AIX operating system. Some configuration steps are required before this support is enabled.

### Before you begin

These steps assume that the LDAP server is RFC 2307 compliant and configured to store user and group information.

### Procedure

1. To configure your AIX client system for LDAP, perform the following steps:
  - a. Log in as a user with root authority.
  - b. Ensure that the LDAP client file set has been installed on your AIX system. AIX works with all versions of LDAP clients: ITDS V6.1 which ships with AIX V6.1, and ITDS V6.2 which ships with the AIX expansion pack. The following shows ITDS V5.2 file sets installed on an AIX system:

```
$ lsllpp -l "ldap*"
Fileset Level State Description

Path: /usr/lib/objrepos
 ldap.client.adt 5.2.0.0 COMMITTED Directory Client SDK
 ldap.client.rte 5.2.0.0 COMMITTED Directory Client Runtime (No
 SSL)
 ldap.html.en_US.config 5.2.0.0 COMMITTED Directory Install/Config
 Gd-U.S. English
 ldap.html.en_US.man 5.2.0.0 COMMITTED Directory Man Pages - U.S.
 English
 ldap.msg.en_US 5.2.0.0 COMMITTED Directory Messages - U.S.
 English
Path: /etc/objrepos
 ldap.client.rte 5.2.0.0 COMMITTED Directory Client Runtime (No
 SSL)
```

- c. Using the **mksecldap** command with the **-c** option, configure the client. For more information about the **mksecldap** command and how to use it to configure the client, see [http://www.ibm.com/support/knowledgecenter/ssw\\_aix\\_72/com.ibm.aix.security/setup\\_ldap\\_sec\\_info\\_server.htm](http://www.ibm.com/support/knowledgecenter/ssw_aix_72/com.ibm.aix.security/setup_ldap_sec_info_server.htm)
  - d. Update default stanza in the `/etc/security/user` file.

Once you are certain that LDAP is configured properly and that you have populated the LDAP directory with users, you must set the default user to use LDAP. This will ensure that you can log in to the AIX client with any user in the LDAP directory that is not restricted.

The **SYSTEM** and **REGISTRY** attributes in the `/etc/security/user` file are used to specify the authentication method and the database used for user management. To enable LDAP authentication and user management, set the **SYSTEM** and **REGISTRY** attributes in the default stanza to LDAP. For example:

```
chsec -f /etc/security/user -s default -a "SYSTEM=LDAP or files"
chsec -f /etc/security/user -s default -a "REGISTRY=LDAP"
```

Db2 supports the following **SYSTEM** attributes:

- LDAP
- files

Db2 supports the following **REGISTRY** attributes:

- LDAP
- KRB5LDAP
- KRB5ALDAP
- files
- KRB5files
- KRB5Afiles

Configurations that use other **SYSTEM** or **REGISTRY** attributes might work, but are not supported.

For more details on the stanza **SYSTEM** and **REGISTRY** attributes, refer to [http://www.ibm.com/support/knowledgecenter/ssw\\_aix\\_72/com.ibm.aix.security/user\\_authentication.htm](http://www.ibm.com/support/knowledgecenter/ssw_aix_72/com.ibm.aix.security/user_authentication.htm).

For more details, refer to the redbook titled, Integrating AIX into Heterogeneous LDAP Environments, at: <http://www.redbooks.ibm.com/abstracts/sg247165.html>

2. To configure transparent LDAP authentication on your Db2 instance:
  - a. Set the **DB2AUTH** miscellaneous registry variable to **0SAUTHDB**. As a user with **SYSADM** authority run **db2set DB2AUTH=0SAUTHDB**.
  - b. Using the **UPDATE DBM CFG** command, set the authentication on the database server instance to any one of the following:
    - **SERVER**
    - **SERVER\_ENCRYPT**
    - **DATA\_ENCRYPT**
  - c. Ensure that you are using the default Client Userid-Password Plugin (**clnt\_pw\_plugin**), Server Userid-Password Plugin (**srvcon\_pw\_plugin**) and Group Plugin (**group\_plugin**).
  - d. Restart the Db2 instance.

### Considerations when using various authentication methods

Transparent LDAP-based authentication and group look up support on AIX extends support to Kerberos authentication.

Additional work was done on AIX for using Kerberos authentication with Transparent LDAP. The following is what needs to be included in **/usr/lib/security/methods.cfg** and **/etc/security/users** when there is a need to manage accounts in different locations and use different authentication methods, such as Kerberos.

In **/usr/lib/security/methods.cfg** you need to have the following to have files, LDAP and Kerberos authentication.

**Note:** KRB5A is for using Microsoft Active Directory as the Kerberos Key Distribution Center (KDC).

For LDAP:

```
program = /usr/lib/security/LDAP
program_64 = /usr/lib/security/LDAP64
```

For KRB5A:



```
program = /usr/lib/security/KRB5A
program_64 = /usr/lib/security/KRB5A_64
options = tgt_verify=no,authonly,is_kadmind_compat=no
```

For KRB5:

```
program = /usr/lib/security/KRB5
program_64 = /usr/lib/security/KRB5_64
options = kadmind=no
```

For KRB5Afiles:

```
options = db=BUILTIN,auth=KRB5A
```

For KRB5files:

```
options = db=BUILTIN,auth=KRB5
```

For KRB5ALDAP:

```
options = db=LDAP,auth=KRB5A
```

For KRB5LDAP:

```
options = db=LDAP,auth=KRB5
```

## Example

The following example shows four accounts managed differently. Each uses different authentication methods.

If frank's account is stored on file and is authenticated using files, then this is what frank's stanza would look like in `/etc/security/users`.

```
frank:
 SYSTEM = files
 registry = files
```

If karen's account is stored on file and is authenticated using Kerberos, then this is what karen's stanza would look like in `/etc/security/users`.

```
karen:
 SYSTEM = KRB5files
 registry = KRB5files
```

If luke's account is stored on LDAP and is authenticated using Kerberos, then this is what luke's stanza would look like in `/etc/security/users`.

```
luke:
 SYSTEM = KRB5LDAP
 registry = KRB5LDAP
```

If lucy's account is stored on LDAP and is authenticated using LDAP, then this is what lucy's stanza would look like in `/etc/security/users`.

```
lucy:
 SYSTEM = LDAP
 registry = LDAP
```

To determine if a user is defined on LDAP you can use the following command to query a user.

```
$ lsuser -R LDAP lucy
lucy id=1234 pgrp=staff groups=staff home=/home/lucy shell=/bin/ksh registry=LDAP
```

## Configuring transparent LDAP for authentication and group lookup (Linux)

Starting in Db2 V9.7 Fix Pack 1 and later, to ensure the Db2 database server transparently uses LDAP-based authentication on the Linux operating system, use Pluggable Authentication Modules (PAM). Your LDAP server should already be configured to store user and group information.

### Before you begin

To enable support for transparent LDAP on the Db2 database, complete the following tasks:

1. Configure your operating system to authenticate users using PAM
2. Configure your Db2 instance

The steps assume that the LDAP server is RFC 2307 compliant.

### Procedure

1. To configure your operating system for LDAP and PAM, perform the following steps:

- a. Log in as a user with root authority.
- b. Ensure that the `nss_ldap` and `pam_ldap` packages are installed. These two packages appear as `libnss_ldap.so` and `libpam_ldap.so` in the `/lib(64)` or `/usr/lib(64)` directories.
- c. Set up your operating system to act as a LDAP client machine by modifying the `/etc/ldap.conf` file to enable the operating system to bind with a LDAP server. Here's a sample `/etc/ldap.conf` file:

```
host <host> # Address of ldap server
base <base> # The DN of the search base.
rootbinddn <binddn> # The bind DN to bind to LDAP
ldap_version 3 # LDAP version
pam_login_attribute uid # user ID attribute for pam user lookups
nss_base_group <group> # nsswitch configuration pertaining to group
 # search lookup
```

- d. Set your password in the `/etc/ldap.secret` file. Only the root user should be able to read or write to this file.
- e. Create or modify the PAM configuration file at `/etc/pam.d/db2`. The file should be only be readable and writable by root. You might have to modify the configuration file, depending on the version of the operating system that is being used. Here is a sample configuration file for SUSE Linux Enterprise Server 10:

```
auth sufficient pam_unix2.so
auth required pam_ldap.so use_first_pass
account sufficient pam_unix2.so
account required pam_ldap.so
password required pam_pwcheck.so
password sufficient pam_unix2.so use_authok use_first_pass
password required pam_ldap.so use_first_pass
session required pam_unix2.so
```

For Red Hat Enterprise Linux 5, modify the configuration file as follows:

```
##PAM-1.0
```

```
auth required pam_env.so
auth sufficient pam_unix.so likeauth nullok
auth sufficient pam_ldap.so use_first_pass
auth required pam_deny.so

account required pam_unix.so
account sufficient pam_succeed_if.so uid < 100 quiet
account sufficient pam_ldap.so
```

```
account required pam_permit.so
```

```
password requisite pam_cracklib.so retry=3 dcredit=-1 ucredit=-1
password sufficient pam_unix.so nullok use_authok md5 shadowremember=3
password sufficient pam_ldap.so use_first_pass
password required pam_deny.so
```

```
session required pam_limits.so
session required pam_unix.so
```

Db2 supports PAM configurations that use `pam_ldap.so`, `pam_unix.so`, and `pam_unix2.so`. Configurations that use other PAM modules might work, but are not supported.

- f. Setup your Linux system to perform group lookup through LDAP. Depending on the version of the operating system that is being used, you may be required to modify the **group** and **passwd** entries in `/etc/nsswitch.conf` file to ensure LDAP is entered as a lookup method. This action does not have to be done for RHEL7 and above. Here is an example of how the RHEL5 **group** and **passwd** should look:

```
group: files ldap
passwd: files ldap
```

2. To configure your Db2 instance to use transparent LDAP authentication, perform the following steps:
  - a. Set the **DB2AUTH** miscellaneous registry variable to `OSAUTHDB`. Issue the following command as a user with `SYSADM` authority:

```
db2set DB2AUTH=OSAUTHDB
```
  - b. Set the authentication on the server to any one of the following:
    - `SERVER`
    - `SERVER_ENCRYPT`
    - `DATA_ENCRYPT`
  - c. Ensure that you are using the default Client Userid-Password Plugin (`clnt_pw_plugin`), Server Userid-Password Plugin (`srvcon_pw_plugin`) and Group Plugin (`group_plugin`).
  - d. Restart the Db2 instance.

## Configuring the LDAP plug-in modules

To configure the LDAP plug-in modules, you need to update your IBM LDAP security plug-in configuration file to suit your environment. In most cases, you will need to consult with your LDAP administrator to determine the appropriate configuration values.

The default name and location for the IBM LDAP security plug-in configuration file is:

- On UNIX: `INSTHOME/sql1lib/cfg/IBMLDAPSecurity.ini`
- On Windows: `%DB2PATH%\cfg\IBMLDAPSecurity.ini`

Optionally, you can specify the location of this file using the `DB2LDAPSecurityConfig` environment variable. On Windows, you should set `DB2LDAPSecurityConfig` in the global system environment, to ensure it is picked up by the Db2 service.

The following tables provide information to help you determine appropriate configuration values.

Table 33. Server-related values

Parameter	Description
LDAP_HOST	The name of your LDAP server(s). This is a space separated list of LDAP server host names or IP addresses, with an optional port number for each one. For example: host1[:port] [host2[:port2] ... ] The default port number is 389, or 636 if SSL is enabled.
ENABLE_SSL	To enable SSL support, set ENABLE_SSL to TRUE (you must have the GSKit installed). This is an optional parameter; it defaults to FALSE (no SSL support).
SSL_KEYFILE	The path for the SSL keyring. A keyfile is only required if your LDAP server is using a certificate that is not automatically trusted by your GSKit installation. For example: SSL_KEYFILE = /home/db2inst1/IBMLDAPSecurity.kdb
SSL_PW	The SSL keyring password. For example: SSL_PW = keyfile-password
SECURITY_PROTOCOL	To enable TLS 1.2 support, set SECURITY_PROTOCOL to TLSV12. To enable TLS 1.0, 1.1, and 1.2 support, set SECURITY_PROTOCOL to ALL. By default, SECURITY_PROTOCOL is not set. This setting means TLS 1.2 is not supported.

Table 34. User-related values

Parameter	Description
USER_OBJECTCLASS	The LDAP object class used for users. Generally, set USER_OBJECTCLASS to inetOrgPerson (the user for Microsoft Active Directory) For example: USER_OBJECTCLASS = inetOrgPerson
USER_BASEDN	The LDAP base DN to use when searching for users. If not specified, user searches start at the root of the LDAP directory. Some LDAP servers require that you specify a value for this parameter. For example: USER_BASEDN = o=ibm
USERID_ATTRIBUTE	The LDAP user attribute that represents the user ID. The USERID_ATTRIBUTE attribute is combined with the USER_OBJECTCLASS and USER_BASEDN (if specified) to construct an LDAP search filter when a user issues a Db2 CONNECT statement with an unqualified user ID. For example, if USERID_ATTRIBUTE = uid, then issuing this statement: db2 connect to MYDB user bob using bobpass results in the following search filter: &(objectClass=inetOrgPerson)(uid=bob)
AUTHID_ATTRIBUTE	The LDAP user attribute that represents the Db2 authorization ID. Usually this is the same as the USERID_ATTRIBUTE. For example: AUTHID_ATTRIBUTE = uid

Table 35. Group-related values

Parameter	Description
GROUP_OBJECTCLASS	The LDAP object class used for groups. Generally this is groupOfNames or groupOfUniqueNames (for Microsoft Active Directory, it is group) For example: GROUP_OBJECTCLASS = groupOfNames

Table 35. Group-related values (continued)

Parameter	Description
GROUP_BASEDN	The LDAP base DN to use when searching for groups. If not specified, group searches start at the root of the LDAP directory. Some LDAP servers require that you specify a value for this parameter. For example: GROUP_BASEDN = o=ibm
GROUPNAME_ATTRIBUTE	The LDAP group attribute that represents the name of the group. For example: GROUPNAME_ATTRIBUTE = cn
GROUP_LOOKUP_METHOD	Determines the method used to find the group memberships for a user. Possible values are: <ul style="list-style-type: none"> <li>SEARCH_BY_DN Indicates to search for groups that list the user as a member. Membership is indicated by the group attribute defined as GROUP_LOOKUP_ATTRIBUTE (typically, member or uniqueMember).</li> <li>USER_ATTRIBUTE In this case, a user's groups are listed as attributes of the user object itself. This setting indicates to search for the user attribute defined as GROUP_LOOKUP_ATTRIBUTE to get the user's groups (typically memberOf for Microsoft Active Directory or ibm-allGroups for IBM Tivoli Directory Server).</li> </ul> For example: GROUP_LOOKUP_METHOD = SEARCH_BY_DN GROUP_LOOKUP_METHOD = USER_ATTRIBUTE
GROUP_LOOKUP_ATTRIBUTE	Name of the attribute used to determine group membership, as described for GROUP_LOOKUP_METHOD.  For example: GROUP_LOOKUP_ATTRIBUTE = member GROUP_LOOKUP_ATTRIBUTE = ibm-allGroups
NESTED_GROUPS	If NESTED_GROUPS is TRUE, the Db2 database manager recursively searches for group membership by attempting to look up the group memberships for every group that is found.  Cycles (such as A belongs to B, and B belongs to A) are handled correctly. This parameter is optional, and defaults to FALSE.

Table 36. Miscellaneous values

Parameter	Description
SEARCH_DN, SEARCH_PW	If your LDAP server does not support anonymous access, or if anonymous access is not sufficient when searching for users or groups, then you can optionally define a DN and password that will be used to perform searches.  For example: SEARCH_DN = cn=root SEARCH_PW = rootpassword
DEBUG	Set DEBUG to TRUE to write extra information to the <b>db2diag</b> log files to aid in debugging LDAP related issues.  Most of the additional information is logged at DIAGLEVEL 4 (INFO). DEBUG defaults to false.

## Enabling the LDAP plug-in modules

Compiled binary LDAP plug-in modules are found in your Db2 instance directory.

The following tables show where the LDAP plug-in modules are located on your Db2 instance.

*Table 37. For 64-bit UNIX and Linux systems*

Plug-in module type	Location
server	/sqllib/security64/plugin/IBM/server
client	/sqllib/security64/plugin/IBM/client
group	/sqllib/security64/plugin/IBM/group

*Table 38. For 32-bit UNIX and Linux systems*

Plug-in module type	Location
server	/sqllib/security32/plugin/IBM/server
client	/sqllib/security32/plugin/IBM/client
group	/sqllib/security32/plugin/IBM/group

*Table 39. For Windows systems (both 64-bit and 32-bit)*

Plug-in module type	Location
server	%DB2PATH%\security\plugin\IBM\instance-name\server
client	%DB2PATH%\security\plugin\IBM\instance-name\client
group	%DB2PATH%\security\plugin\IBM\instance-name\group

**Note:** 64-bit Windows plug-in modules include the digits 64 in the file name.

Use the Db2 command line processor to update the database manager configuration to enable the plug-in modules that you require:

- For the server plug-in module:  
UPDATE DBM CFG USING SRVCON\_PW\_PLUGIN IBMLDAPauthserver
- For the client plug-in module:  
UPDATE DBM CFG USING CLNT\_PW\_PLUGIN IBMLDAPauthclient
- For the group plug-in module:  
UPDATE DBM CFG USING GROUP\_PLUGIN IBMLDAPgroups

Terminate all running Db2 command line processor backend processes, by using the **db2 terminate** command, and then stop and restart the instance by using the **db2stop** and **db2start** commands.

## Connecting with an LDAP user ID

After the LDAP security plug-ins have been configured in a Db2 instance, a user can connect to the databases using a variety of different user strings.

The location of an object within an LDAP directory is defined by its distinguished name (DN). A DN is typically a multi-part name that reflects some sort of hierarchy, for example:

```
cn=John Smith, ou=Sales, o=WidgetCorp
```

A user's *user ID* is defined by an attribute associated with the user object (typically the **uid** attribute). It may be a simple string (such as `jsmith`), or look like an email address (such as `jsmith@sales.widgetcorp.com`), that reflects part of the organizational hierarchy.

A user's Db2 *authorization ID* is the name associated with that user within the Db2 database.

In the past, users were typically defined in the server's host operating system, and the user ID and authorization ID were the same (though the authorization ID is usually in uppercase). The Db2 LDAP plug-in modules give you the ability to associate different attributes of the LDAP user object with the user ID and the authorization ID. In most cases, the user ID and authorization ID can be the same string, and you can use the same attribute name for both the `USERID_ATTRIBUTE` and the `AUTHID_ATTRIBUTE`. However, if in your environment the user ID attribute typically contains extra information that you do not want to carry over to the authorization ID, you can configure a different `AUTHID_ATTRIBUTE` in the plug-in initialization file. The value of the `AUTHID_ATTRIBUTE` attribute is retrieved from the server and used as the internal Db2 representation of the user.

For example, if your LDAP user IDs look like email addresses (such as `jsmith@sales.widgetcorp.com`), but you would rather use just the user portion (`jsmith`) as the Db2 authorization ID, then you can:

1. Associate a new attribute containing the shorter name with all user objects on your LDAP server
2. Configure the `AUTHID_ATTRIBUTE` with the name of this new attribute

Users are then able to connect to a Db2 database by specifying their full LDAP user ID and password, for example:

```
db2 connect to MYDB user 'jsmith@sales.widgetcorp.com' using 'pswd'
```

But internally, the Db2 database manager refers to the user using the short name retrieved using the `AUTHID_ATTRIBUTE` (`jsmith` in this case).

After an LDAP plug-in module has been enabled and configured, a user can connect to a Db2 database using a variety of different strings:

- A full DN. For example:  

```
connect to MYDB user 'cn=John Smith, ou=Sales, o=WidgetCorp'
```
- A partial DN, provided that a search of the LDAP directory using the partial DN and the appropriate search base DN (if defined) results in exactly one match. For example:  

```
connect to MYDB user 'cn=John Smith' connect to MYDB user uid=jsmith
```
- A simple string (containing no equals signs). The string is qualified with the `USERID_ATTRIBUTE` and treated as a partial DN. For example:  

```
connect to MYDB user jsmith
```

**Note:** Any string supplied on a `CONNECT` statement or **ATTACH** command must be delimited with single quotation marks if it contains spaces or special characters.

You must configure the `CLNT_PW_PLUGIN` and `GROUP_PLUGIN` parameters on the Db2 client if you want to use full or partial DNs:

```
update dbm cfg using CLNT_PW_PLUGIN IBMLDAPauthclient
update dbm cfg using GROUP_PLUGIN IBMLDAPgroups
```

You must also update the LDAP plug-in configuration file, `IBMLDAPSecurityt.ini`.

## Considerations for group lookup

Group membership information is typically represented on an LDAP server either as an attribute of the user object, or as an attribute of the group object:

- As an attribute of the user object  
Each user object has an attribute called `GROUP_LOOKUP_ATTRIBUTE` that you can query to retrieve all of the group membership for that user.
- As an attribute of the group object  
Each group object has an attribute, also called `GROUP_LOOKUP_ATTRIBUTE`, that you can use to list all the user objects that are members of the group. You can enumerate the groups for a particular user by searching for all groups that list the user object as a member.

Many LDAP servers can be configured in either of these ways, and some support both methods at the same time. Consult with your LDAP administrator to determine how your LDAP server is configured.

When configuring the LDAP plug-in modules, you can use the `GROUP_LOOKUP_METHOD` parameter to specify how group lookup should be performed:

- If you need to use the `GROUP_LOOKUP_ATTRIBUTE` attribute of the user object to find group membership, set `GROUP_LOOKUP_METHOD = USER_ATTRIBUTE`
- If you need to use the `GROUP_LOOKUP_ATTRIBUTE` attribute of the group object to find group membership, set `GROUP_LOOKUP_METHOD = SEARCH_BY_DN`

Many LDAP servers use the `GROUP_LOOKUP_ATTRIBUTE` attribute of the group object to determine membership. They can be configured as shown in this example:

```
GROUP_LOOKUP_METHOD = SEARCH_BY_DN
GROUP_LOOKUP_ATTRIBUTE = groupOfNames
```

Microsoft Active Directory typically stores group membership as a user attribute, and could be configured as shown in this example:

```
GROUP_LOOKUP_METHOD = USER_ATTRIBUTE
GROUP_LOOKUP_ATTRIBUTE = memberOf
```

The IBM Tivoli Directory Server supports both methods at the same time. To query the group membership for a user you can make use of the special user attribute **ibm-allGroups**, as shown in this example:

```
GROUP_LOOKUP_METHOD = USER_ATTRIBUTE
GROUP_LOOKUP_ATTRIBUTE = ibm-allGroups
```

Other LDAP servers may offer similar special attributes to aid in retrieving group membership. In general, retrieving membership through a user attribute is faster than searching for groups that list the user as a member.

## Troubleshooting authenticating LDAP users or retrieving groups

If you encounter problems authenticating LDAP users or retrieving their groups, the **db2diag** log files and administration log are a good source of information to aid in troubleshooting.



The LDAP plug-in modules typically log LDAP return codes, search filters, and other useful data when a failure occurs. If you enable the `DEBUG` option in the LDAP plug-in configuration file, the plug-in modules will log even more information in the **db2diag** log files. While this might be an aid in troubleshooting, it is not recommended for extended use on production systems due to the overhead associated with writing all of the extra data to a single file.

Ensure that the **diaglevel** configuration parameter in the database manager is set to 4 so that all messages from the LDAP plug-in modules will be captured.

---

## Writing security plug-ins

### 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 plug-in 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
- Specify the pointers to the other functions in the library
- Specify 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.

## Restrictions for developing security plug-in libraries

There are certain restrictions that affect how you develop plug-in libraries.

The following list outlines 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 the Db2 signal handlers. Interfering with the Db2 signal handlers could seriously interfere with the ability for Db2 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 the error handling used in Db2.

### 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 dependent libraries are accessible through other methods, such as the following situations:

- 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_USERNAMESPACE_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 previously.

### 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.

**Fork** Plug-in libraries should not fork because file descriptors and sockets will be duplicated in the child process, and this can cause hangs or incorrect behavior. In particular, it can cause false file lock conflicts if child was forked when we had an open file descriptor on that file. There is also the possibility that the fork will inherit many other resources like semaphores.

## 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 authorization ID:** In Db2 Version 9.5 and later, you can have a 128-byte authorization ID. However, when the authorization ID is interpreted as an operating system user ID or group name, Db2 imposed naming restrictions apply. For example, the Linux and UNIX operating systems can contain up to 8 characters and the Windows operating systems can contain up to 30 characters for user IDs and group names. Therefore, if you want to connect as a user that has a 128-byte authorization ID, you need to write your own security plug-in. In the plug-in, you can use the extended sizes for the authorization ID. For example, you can give your security plug-in a 30-byte user ID and, during authentication, it returns a 128-byte authorization ID that you can connect to.

## **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:

```
xlcr -qmksrobj -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:

```
xlcr -qmksrobj -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.

## 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 40. 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_UNKNOWNEROR	The plug-in API encountered an unexpected error.	All
-2	DB2SEC_PLUGIN_BADUSER	The user ID passed in as input is not defined.	db2secGenerateInitialCred db2secValidatePassword db2secRemapUserid db2secGetGroupsForUser
-3	DB2SEC_PLUGIN_INVALIDUSERORGROUP	No such user or group.	db2secDoesAuthIDExist 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
-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

Table 40. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-6	DB2SEC_PLUGIN_UID_EXPIRED	User ID expired.	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-7	DB2SEC_PLUGIN_PWD_EXPIRED	Password expired.	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-8	DB2SEC_PLUGIN_USER_REVOKED	User revoked.	db2secValidatePassword db2GetGroupsForUser
-9	DB2SEC_PLUGIN_USER_SUSPENDED	User suspended.	db2secValidatePassword db2GetGroupsForUser
-10	DB2SEC_PLUGIN_BADPWD	Bad password.	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-11	DB2SEC_PLUGIN_BAD_NEWPASSWORD	Bad new password.	db2secValidatePassword db2secRemapUserid
-12	DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED	Change password not supported.	db2secValidatePassword db2secRemapUserid 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 db2secClientAuthPluginInit 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
-19	DB2SEC_PLUGIN_FILENOTFOUND	Plug-in is unable to open and read a file, because the file is missing from the file system.	All

Table 40. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-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 db2secServerAuthPluginInit
-22	DB2SEC_PLUGIN_CRED_EXPIRED	GSS API plug-in only: client credential has expired.	db2secGetDefaultLoginContext 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 db2secClientAuthPluginInit 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.

## 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 calls 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 calls 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 calls 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 negotiate user ID/password authentication (for example, when the **srvcon\_auth** parameter at the server is set to SERVER; SERVER\_ENCRYPT, DATA\_ENCRYPT, or DATA\_ENCRYPT\_CMP), the client application calls 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 negotiate userid/password authentication (for example, when the **srvcon\_auth** parameter at the server is set to SERVER; SERVER\_ENCRYPT, DATA\_ENCRYPT, or DATA\_ENCRYPT\_CMP), the client application calls the following security plug-in APIs:

- db2secRemapUserid();

### GSSAPI and Kerberos - implicit

On an implicit authentication, when the client and server negotiate GSS-API or Kerberos authentication (for example, when the **srvcon\_auth** parameter at the server is set to KERBEROS; KRB\_SERVER\_ENCRYPT, GSSPLUGIN, or GSS\_SERVER\_ENCRYPT), the client application calls the following security plug-in APIs. (The call to gss\_init\_sec\_context() uses 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 calls 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()` might 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 calls the following security plug-in APIs:

- `db2secGetDefaultLoginContext();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

These APIs are 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 calls 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 thread calls 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 calls 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 calls the following security plug-in API with only the authorization ID as input:

- db2secGetGroupsForUser();

There will be no token from other security plug-ins.

---

## Chapter 9. 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 list shows 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 enter in the value for the security plug-in API parameter.

## Output

Indicates that the security plug-in API will specify the value for the API parameter.

---

## 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 usernamespace,
 db2int32 usernamespace,
 const char *dbname,
 db2int32 dbname,
 const void *token,
 db2int32 tokentype,
);
};
```

```

db2int32 location,
const char *authpluginname,
db2int32 authpluginnamelen,
void **grouplist,
db2int32 *numgroups,
char **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secDoesGroupExist)
(
const char *groupname,
db2int32 groupnamelen,
char **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeGroupListMemory)
(
void *ptr,
char **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
char *msgtobefree
);

SQL_API_RC (SQL_API_FN * db2secPluginTerm)
(
char **errmsg,
db2int32 *errmsglen
);

} 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.

## db2secDoesGroupExist API - Check if group exists

Determines whether 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 **errmsg,
 db2int32 *errormsglen);
```

### db2secDoesGroupExist API parameters

#### groupname

Input. An authid, upper-cased, with no trailing blanks.

#### groupnamelen

Input. Length in bytes of the **groupname** 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 db2secDoesGroupExist 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.

## 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

#### errmsg

Input. A pointer to the error message allocated from a previous API call.

## 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 *errormsglen);
```

### 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.

#### errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in the **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 dbname,
 void *token,
 db2int32 tokentype,
 db2int32 location,
 const char *authpluginname,
 db2int32 authpluginname,
 void **group,
 db2int32 *numgroups,
 char **errmsg,
 db2int32 *errormsglen);
```

### 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 always provides 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, see 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.

**usernamepacelen**

Input. Length in bytes of the **usernamepace** parameter value.

**usernamepacetype**

Input. The type of namespace. Valid values for the **usernamepacetype** 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 **usernamepacetype** 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 the first byte). 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.

**errmsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

**Usage notes**

The following list describes the scenarios that which 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 rights stated previously 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 rights stated previously 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 scenarios described previously 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.

**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 **errmsg,
 db2int32 *errmsglen);
```

### 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 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 shows 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. So for example, if you use the DB2SEC\_LOG\_INFO value, the message text shows up in the **db2diag** log files only 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 example, 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 operating systems.

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN *db2secPluginTerm)
(char **errmsg,
 dbint32 *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.

---

## 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)
(
 dbint32 level,
 void *data,
 dbint32 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 do not need to 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 *usernamespacelen,
 db2int32 *usernamespacetype,
 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 *usernamespacelen,
 db2int32 *usernamespacetype,
 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 *usernamespace,
 db2int32 usernamespacelen,
 db2int32 usernamespacetype,
 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 **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
char *errmsg
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char **errmsg,
db2int32 *errmsglen
);
}

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 *usernamespacelen,
db2int32 *usernamespacetype,
const char *dbname,
db2int32 dbnamelen,
void **token,
char **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secProcessServerPrincipalName)
(
const void *data,
gss_name_t *gssName,
char **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secGenerateInitialCred)
(
const char *userid,
db2int32 useridlen,
const char *usernamespace,
db2int32 usernamespacelen,
db2int32 usernamespacetype,
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 a 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 TCPIP4 */
 db2UInt32 connect_info_bitmap;
 db2int32 dbnameLen;
 char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
 db2UInt32 clientIP6Address[4]; /* Set if protocol is TCPIP6 */
} 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 TCPIP4 */
 db2UInt32 connect_info_bitmap;
 db2int32 dbnameLen;
 char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
 db2UInt32 clientIP6Address[4]; /* Set if protocol is TCPIP6 */
 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\_TCTIP (IPv4, with v1 of the structure), SQL\_PROTOCOL\_TCTIP4 (IPv4, with v2 of the structure) or SQL\_PROTOCOL\_TCTIP6 (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 db2sec_userid_password_server_auth_functions_1
{
 db2int32 version;
 db2int32 plugin_type;

 /* 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 db2sec_gssapi_server_auth_functions_1
{
 db2int32 version;
 db2int32 plugin_type;
 gss_buffer_desc server_principal_name;
 gss_cred_id_t server_cred_handle;
 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 db2sec\_userid\_password\_server\_auth\_functions\_1 structure if you are writing a user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use the db2sec\_gssapi\_server\_auth\_functions\_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 userid/password authentication is used. The db2secGssapiClientAuthFunctions\_<version\_number> structure and db2secUseridPasswordClientAuthFunctions\_<version\_number> structure contain pointers to the APIs implemented for the GSS-API authentication plug-in and userid/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 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 appear in the **db2diag** log files only 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.

#### **errmsglen**

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 example, 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 operating systems.

### **API and data structure syntax**

```
SQL_API_RC (SQL_API_FN *db2secClientAuthPluginTerm)
(char **errmsg,
 dbint32 *errmsglen);
```

### **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.

#### **errmsglen**

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 *errormsglen);
```

### 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.

#### **errormsglen**

Output. A pointer to an integer that indicates the length 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.

## 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.

## 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 usernamespace,
 db2int32 usernamespace,
 const char *password,
 db2int32 passwordlen,
 const char *newpassword,
 db2int32 newpasswordlen,
 const char *dbname,
 db2int32 dbname,
 gss_cred_id_t *pgSSCredHandle,
 void **InitInfo,
 char **errmsg,
 db2int32 *errormsglen);
```

### 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.

**usernamepace**

Input. The namespace from which the user ID was obtained.

**usernamepace1en**

Input. Length in bytes of the **usernamepace** parameter value.

**usernamepacetype**

Input. The type of namespace.

**password**

Input. The password to be verified.

**password1en**

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 honor 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.

**newpassword1en**

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.

**dbname1en**

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 be returned only if there is an internal error in the API that prevented it from completing properly.

**errmsglen**

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 usernamespacelen,
 db2int32 usernamespacetype,
 const char *dbname,
 db2int32 dbname1en,
 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 *username1en,
 db2int32 *initsessionidtype,
 char **errmsg,
 db2int32 *errmsglen);
```

### 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.

**usernamespacelen**

Input. Length in bytes of the **usernamespace** parameter value.

**usernamespacetype**

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.

**dbname1en**

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 mentioned previously.

#### **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 example, 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 be used only for auditing and will be logged in the "User ID" field in the audit record for CONNECT statement. If the API does specify the **username** parameter, the Db2 database manager copies it from the **userid**.

#### **username1en**

Output. Length in bytes of the **username** parameter value.

#### **initsessionidtype**

Output. Session authid type indicating whether 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.

#### **errmsg1en**

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, that is, 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 *usernamespacetype,
 const char *dbname,
 db2int32 dbnamelen,
 void **token,
 char **errmsg,
 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 user ID. 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.

#### **usernamespacelen**

Output. Length in bytes of the **usernamespace** parameter value. Under the limitation that the **usernamespacetype** parameter must be set to the value DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE (defined in db2secPlugin.h), the maximum length currently supported is 15 bytes.

**usernamespace type**

Output. Namespace type 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.

**dbname len**

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.

**errmsg len**

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 the **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.

**errmsglen**

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 usernamespace) 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 usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
 db2int32 *userspacelen,
 db2int32 *userspacetype,
 char password[DB2SEC_MAX_PASSWORD_LENGTH],
 db2int32 *passwordlen,
 char newpasswd[DB2SEC_MAX_PASSWORD_LENGTH],
 db2int32 *newpasswdlen,
 const char *dbname,
 db2int32 dbnamelen,
 char **errmsg,
 db2int32 *errmsglen);
```

### 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.

**namespace**

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 **namespace** will be used by the Db2 database manager only for CLIENT type authentication and is disregarded for other authentication types.

**userspacelen**

Input or output. Length in bytes of the **namespace** parameter value. Under

the limitation that the **usernamespacetype** 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.

**dbname1en**

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.

## 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 **errmsg,
 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 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 **plugin\_type** 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 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 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 appear in the **db2diag** log files only 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 example, 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 operating systems.

### **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.

**errmsglen**

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 previous scenarios 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 **errmsg,
 db2int32 *errmsglen);
```

## 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.

**dbname1en**

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 can disallow implicit authentication by returning a `DB2SEC_PLUGIN_BADPASSWORD` error.

**token**

Input/output. 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.

## Required APIs and definitions for GSS-API authentication plug-ins

The following table 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 41. Required APIs and Definitions for GSS-API authentication plug-ins

API type	API 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.
Required definitions	GSS_C_EMPTY_BUFFER	Signifies that the gss_buffer_desc 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 gss_buffer_t variable does not point to a valid gss_buffer_desc structure.
Required definitions	GSS_C_NO_CHANNEL_BINDINGS	No communication channel bindings.
Required definitions	GSS_C_NO_CONTEXT	Signifies that the gss_ctx_id_t variable does not point to a valid context.

Table 41. Required APIs and Definitions for GSS-API authentication plug-ins (continued)

API type	API name	Description
Required definitions	GSS_C_NO_CREDENTIAL	Signifies that gss_cred_id_t variable does not point to a valid credential handle.
Required definitions	GSS_C_NO_NAME	Signifies that the gss_name_t 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 list describes the 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.

---

## Chapter 10. Communication buffer exit libraries

With communication buffer exit libraries, you can examine communication buffers to provide solutions such as auditing or other security solutions that are based on the contents of the buffers.

Db2 provides access to each buffer received from clients, and each buffer about to be sent to clients. Buffers are provided before they are encrypted with either `DATA_ENCRYPT` authentication or SSL. Db2 uses the DRDA protocol to communicate between clients and the server. The communication buffers that are passed to the communication buffer exit library are formatted according to the DRDA protocol. The communication buffer exit library must understand the DRDA protocol that is used for communication.

Db2 provides the buffers regardless of communication protocol. Communication buffer exit libraries work consistently with TCPIP (IPv4 and IPv6), SSL, Inter-Process Communication (IPC), and named pipe.

In addition to the buffers, Db2 also makes available identity information, including the user name and session authorization ID established for the connection to the database. This information is useful for scenarios that involve GSS-API plug-ins such as Kerberos. In this scenario, there is no standard user name, but rather generic tickets from which the database manager derives the user name. This detail is not available solely by looking at the communication buffer.

The database manager ensures that only trusted libraries are loaded. The libraries must be installed in a specific location that can be modified by only the instance owner. Furthermore, only a user with SYSADM authority can enable the library. This authority level is the same which is required to enable encryption (`DATA_ENCRYPT` or SSL).

The communication buffer exit library can terminate a connection if any buffer provided contains data that the library considers harmful. Both data that is sent to the server, and data that is returned to the client is included. For example, the communication buffer exit library might detect that the data returned from a select statement is inappropriate for the client to receive. A return code from the library indicates to the database manager that the connection must be terminated. The database managers stops that or any further communication buffers to the client and terminates the connection.

**Note:** Third-party vendors typically provide these communication buffer exit libraries. Db2 does provide samples of libraries in the `sqllib/samples/security/commexit` directory. You might choose to develop your own libraries with the samples as a guide.

---

### Communication exit library deployment

Certain considerations must be taken with the deployment of a communication exit library.

In typical scenarios communication exit libraries are provided by vendors. In these scenarios, the deployment of communication exit libraries is handled by the vendor supplied installation scripts. The deployment steps are outlined here so you

can deploy your own communication exit library if you choose to do so. The steps apply to the deployment of both runtime communication exit libraries and communication buffer exit libraries.

## Communication exit library location

Communication exit libraries must exist in specific directories.

The database manager looks for communication exit libraries in the following directories:

### Linux and UNIX 32-bit

\$DB2PATH/security32/plugin/commexit

### Linux and UNIX 64-bit

\$DB2PATH/security64/plugin/commexit

### Windows 32-bit and 64-bit

\$DB2PATH\security\plugin\commexit\instance\_name

**Note:** On Windows operating systems, the subdirectories instance\_name and commexit are not created automatically. The instance owner must manually create them.

## Naming conventions and permissions of communication exit libraries

Communication exit libraries must adhere to platform-specific naming and permission rules.

The maximum length of a communication exit library name, not including the file extension and the 64 suffix, is limited to 32 bytes.

The following list outlines the naming convention for the library file extension on each operating system:

### AIX

The extension must be .a or .so

**Note:** If both the .a and .so extensions exist, .a is used.

### Linux, HP IPF, and Solaris

The extension must be .so

### Windows

The extension must be .dll

The following list outlines the permission for the library file on each operating system:

### UNIX and Linux

Owned by the instance owner and readable and executable by only the instance owner.

### Windows

Owned by a member of the DB2AMINS group and readable and executable by a member of the DB2ADMINS group.

## Examples

The following example shows the communication exit library extensions on a library that is called `mycommexit` on all operating systems:

- AIX 64-bit `mycommexit.a` or `mycommexit.so`
- Solaris 64-bit, Linux 32-bit, or 64-bit, HP 64-bit on IPF: `mycommexit.so`
- Windows 32-bit: `mycommexit.dll`
- Windows 64-bit: `mycommexit64.dll`

**Note:** The file name suffix 64 is required only on the library name for Windows 64-bit.

When you update the database manager configuration with the name of a communication exit library, use the full name of the library without the 64 suffix. The file extension, and qualified path to the file, must not be specified either when you update the database manager configuration.

The following example updates the database manager configuration on a Windows 64-bit system that sets the `mycommexit64.dll` library as the communication exit library.

```
UPDATE DBM CFG USING COMM_EXIT_LIST mycommexit
```

**Note:** The `COMM_EXIT_LIST` name is case-sensitive, and must exactly match the library name.

## Enabling communication exit libraries outside of Db2 pureScale environments

The steps that are outlined in this task are typically run by third party supplied installation scripts. The steps are outlined to help you enable a communication exit library that you develop.

### Before you begin

You must have SYSADM authority to run the steps in this task.

Restrictions

The communication exit library files must follow strict file permission guidelines. For more information about these guidelines, see the related concepts.

### Procedure

To enable a communication exit library:

1. Stop the database manager. To stop the database manager, run the **db2stop** command.
2. Copy the communication exit library file to the correct directory. For more information about the location of communication exit libraries, see the related concepts. The file can be a symbolic link to another location if wanted.
3. Update the database manager configuration parameter **COMM\_EXIT\_LIST** with the name of the library. To update the configuration parameter, use the **UPDATE DBM CFG** command.
4. Start the database manager. To start the database manager, run the **db2start** command.

## Results

The library is loaded and initialized.

## Enabling communication exit libraries in Db2 pureScale environments

The steps that are outlined in this task are typically run by third party supplied installation scripts. The steps are outlined to help you enable a communication exit library that you develop.

### About this task

By using a communication exit library that contains a version number in the file name, and a symbolic link to this file for a library without the version number, it is possible to deploy the library on a member by member basis. In this scenario, it is not necessary to stop the whole instance, only individual members.

#### Restrictions

The communication exit library files must follow strict file permission guidelines. For more information about these guidelines, see the related concepts.

### Procedure

To enable a communication exit library:

1. Copy the communication exit library that contains the version number in file name to the correct directory. For more about the location of communication exit libraries, see the related concepts.
2. Create a symbolic link from the library without a version to the library that contains the version in the file name.
3. Update the database manager configuration parameter **comm\_exit\_list** with the name of the library. To update the configuration parameter, use the **UPDATE DBM CFG** command.
4. Stop each member individually. To stop each member, run the **db2stop** command on each member
5. Restart the stopped members. To start the stopped members, run the **db2start** command.

## Results

The library is loaded and initialized.

## Communication exit library problem determination

Some options are available to help diagnose problems with a communication exit library.

The communication exit library is not provided as part of Db2. Rather, it is a library that you install. It might be automatically installed and configured by a tool or application that you are using, or it might be written by you.

The name of the library that is specified in the database manager configuration parameter **comm\_exit\_list** gives some indication as to the source of the library.



If you experience any issue with the library, the documentation for the tool or application must be consulted. The tool or application documentation outlines what problem determination steps must be taken.

An interface to write to the db2diag log files is available to communication exit libraries. The db2diag log files can be checked whether there are concerns if the library is functioning properly.

If there are concerns about the performance of the communication exit library, monitoring wait times can be used to investigate how long the library is taking. For more information about these monitor tools, see the related reference.

---

## Communication exit library development

Certain considerations must be taken with the development of a communication exit library.

In typical scenarios communication exit libraries are provided by vendors. In these scenarios, the development of communication buffer exit libraries is handled by the vendor.

Communication exit libraries are C or C++ shared objects that are dynamically loaded into the process space of the database manager. You can develop your own library if you choose to do so.

### How a communication exit library is loaded

When the database manager is started, the communication exit library is dynamically loaded and initialized. The library must contain the initialization function `db2commexitInit`. This function is known as the library initialization function.

The library initialization function initializes the specified communication exit library. The initialization provides the database manager with the information needed to call the library functions. The library initialization function accepts the following parameters:

- The highest version number of the function pointer structure that the database instance which starts the library can support.
- A pointer to a structure which contains pointers to all the APIs that require 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 function signature for the initialization function is:

```
SQL_API_RC SQL_API_FN db2commexitInit
(db2int32 version,
 void *commexit_fns,
 db2commexitLogMessage *logMessage_fn,
 char **errmsg,
 db2int32 *errormsglen);
```

The initialization function is the only function in the library that uses a prescribed function name. The other library functions are referenced through function pointers that are returned from the initialization function.

The specific tasks of this function are:

- Cast the functions pointer to a pointer of an appropriate functions structure.
- Assign the pointers to the other functions in the library.
- Assign the version number of the function pointer structure that is returned.

The communication exit library infrastructure supports both the communication buffer exit library and the runtime communication exit library. The input version parameter contains the highest version numbers for both of these libraries. This function must use the `DB2COMMEXIT_GET_BUFFER_FN_VER` macro to obtain the highest supported version number of the function pointer structure for DRDA style functions. The function must also use the `DB2COMMEXIT_GET_RUNTIME_FN_VER` macro to obtain the highest supported version number of the function pointer structure for the runtime communication exit library functions.

To use the communication buffer exit library, this function must cast `commexit_fns` to `db2commexitFunctions_v1`. The function must also define the function pointers and call the `DB2COMMEXIT_SET_BUFFER_FN_VER` macro to set the version number. To use the runtime communication exit library, this function must cast `commexit_fns` to `db2commexitRuntimeFunctions_v1`. The function must also define the function pointers and call the `DB2COMMEXIT_SET_RUNTIME_FN_VER` macro to set the version number. Only one of the macros must be called by this function.

The function `db2commexitInit` must be declared `extern "C"` if the library is compiled as C++.

## Communication exit library APIs

APIs are implemented in the communication exit library. Some of the following APIs can be called by both communication buffer exit libraries and runtime communication exit libraries. Other APIs can be called by only one of those communication exit library types.

### db2commexitInit API - Initialization

When the database manager is started with the **db2start** command, the communication buffer exit library is loaded. Immediately following the load of the library, this function is called. This function is responsible for initializing the communication buffer exit library. The function is also responsible for returning all of the implemented functions back to the database manager. It can be called by both types of communication exit libraries.

This function must be declared `extern "C"` if the library is compiled as C++.

This function is not required to be threadsafe, since it is only called a single time.

### API header file

`db2commexit.h`

### API and data structure syntax

```
SQL_API_RC (SQL_API_FN * db2commexitInit)
(
 db2int32 version,
 void *commexit_fns,
```

```

 db2commexitLogMessage *logMessage_fn,
 char **errmsg,
 db2int32 *errmsglen
);

```

## db2commexitInit API Parameters

### version

Input. The highest version of the API supported by the instance loading that library. The value DB2COMMEXIT\_API\_VERSION, in db2commexit.h, contains the latest version number of the API that the database manager currently supports. If the library implements a communication buffer exit library, then db2commexitInit function must call the DB2COMMEXIT\_GET\_BUFFER\_FN\_VER macro to get the highest version of the API supported. If the library implements a runtime communication exit library, then db2commexitInit function must call the DB2COMMEXIT\_GET\_RUNTIME\_FN\_VER macro to get the highest version of the API supported.

### commexit\_fns

Output. A pointer to the db2commexitFunctions\_<version\_number> structure, that contains pointers to the APIs implemented for the communication buffer exit library. There might be different versions of the APIs, so the **commexit\_fns** parameter is cast to the db2commexitFunctions\_<version\_number> structure corresponding to the version implemented by the library. The first parameter of the db2commexitFunctions\_<version\_number> structure indicates the version of the APIs implemented by the plug-in. If the library implements a communication buffer exit library, the version number must be set by calling the DB2COMMEXIT\_SET\_BUFFER\_FN\_VER macro. If the library implements a runtime communication exit library, the version number must be set by calling the DB2COMMEXIT\_SET\_RUNTIME\_FN\_VER macro. Another member, nonSQLAPIVersion, of the structure tells the database manager which version number of the non-SQL APIs that the library can handle. Currently, only DB2COMMEXIT\_NONSQL\_API\_VERSION\_KEPLER is supported.

### logMessage\_fn

Input. A pointer to the db2commexitLogMessage API, which is implemented by the database manager. The db2commexitInit API can call the db2commexitLogMessage API to log messages to the db2diag log files for debugging or informational purposes. The first parameter of the db2commexitLogMessage API specifies the type of diagnostic errors that are recorded in the db2diag log files and the last two parameters are the message string and its length. The valid values for the first parameter of the db2commexitLogMessage API, defined in db2commexit.h, are:

- DB2COMMEXIT\_LOG\_NONE: (0) No logging
- DB2COMMEXIT\_LOG\_CRITICAL: (1) Severe Error encountered
- DB2COMMEXIT\_LOG\_ERROR: (2) Error encountered
- DB2COMMEXIT\_LOG\_WARNING: (3) Warning
- DB2COMMEXIT\_LOG\_INFO: (4) Informational

The message text is logged in the db2diag log files only if the value of the 'level' parameter of the db2commexitLogMessage API is less than or equal to the **diaglevel** database manager configuration parameter. For example, if you use the DB2COMMEXIT\_LOG\_INFO value, the message text is logged only if the **diaglevel** database manager configuration parameter is set to 4.

### errmsg

Output. A pointer to the address of an ASCII error message string. This pointer

is allocated by the plug-in and can be returned in this parameter if the function execution is not successful. This memory is freed by calling `db2commexitFreeErrorMsg`.

**errmsglen**

Output. A pointer to an integer that indicates the length, in bytes, of the error message string in the **errmsg** parameter.

## **db2commexitTerm API - Termination**

This function frees resources that are used by the communication exit library. It can be called by both types of communication exit libraries.

This API is called by the database manager just before it unloads the communication exit library during **db2stop** processing. The API must be implemented in a manner so it does a complete cleanup of any resources the library holds. For instance, the API must free any memory that is allocated by the library, close files that are still open, and close network connections. The library is responsible for tracking these resources to free them.

This function is not required to be threadsafe as it is called only one time.

### **API header file**

`db2commexit.h`

### **API and data structure syntax**

```
SQL_API_RC (SQL_API_FN * db2commexitTerm)
(
 char **errmsg,
 db2int32 *errmsglen
);
```

### **db2commexitTerm API Parameters**

**errmsg**

Output. A pointer to the address of an ASCII error message string that is allocated by the communication buffer exit library. This error messages string might be returned in this parameter if the function execution is not successful. This memory is freed by calling `db2commexitFreeErrorMsg`.

**errmsglen**

Output. A pointer to an integer that indicates the length, in bytes, of the error message string in the **errmsg** parameter.

## **db2commexitRegister API - Registration**

This function registers the agent to the connection. It is applicable only in communication buffer exit libraries.

This function is called by the database manager whenever an agent accepts a socket and starts receiving and sending data on the socket. This activity is typically associated with a new SQL connection to the database or instance attachment.

This function is also called when an idle connection is dispatched to an agent to handle a new request from the client.

This function is not directly associated with SQL connections to the database. An input parameter to the function differentiates between a new socket and existing one that is dispatched to a new agent.

## API header file

db2commexit.h

## API and data structure syntax

```
SQL_API_RC (SQL_API_FN * db2commexitRegister)
(
 void ** pConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 db2int32 state,
 db2int64 * pReservedFlags,
 char ** errmsg,
 db2int32 * errmsglen
);
```

## db2commexitRegister API Parameters

### pConnectionContext

Input/output. A pointer to communication buffer exit library-specific data. This pointer is specific to the inbound connection. This parameter is passed as input to each function call for that connection. The library might allocate and store connection-specific information and make it available in each function call. The memory for the parameter must be freed in the call to db2commexitDeregister. The database manager cannot access the memory pointed to by this parameter.

### pCommInfo

Input. A pointer to a structure that contains information which identifies the database server and protocol-specific information for the incoming connection. Some of the fields in the structure are not setup until multiple buffers are exchanged with the client. The fields are available in later calls to db2commexitRecv and db2commexitSend. This scenario applies specifically to **inbound\_appl\_id**, **outbound\_appl\_id**, and **connection\_type**. When these values are known, the **connection\_type** parameter indicates whether the connection is for a local database or a gateway connection.

### State

Input. Indicates under which condition the function called. Possible values are:

- **NEW\_CONNECTION** - indicates a new physical incoming client connection.
- **AGENT\_ASSOCIATION** - indicates an existing idle client connection that becomes active again and is associated with an agent to handle the request.

### pReservedFlags

Input/output. Reserved for future use. The value must be set to 0 on output.

### errmsg

Output. A pointer to the address of an ASCII error message string that is allocated by the communication buffer exit library. This error messages string might be returned in this parameter if the function execution is not successful. This memory is not freed by calling db2commexitFreeErrMsg.

### errmsglen

Output. A pointer to an integer that indicates the length, in bytes, of the error message string in the **errmsg** parameter.

## db2commexitDeregister API - Deregistration

This function releases the agent from the connection with which it was associated. It is applicable only in communication buffer exit libraries.

This function is called by the database manager whenever the agent stops handling requests on the connection. This situation occurs when the physical connection with the client is terminated, or the client is idle and the agent is disassociating with it.

## API header file

db2commexit.h

## API and data structure syntax

```
SQL_API_RC (SQL_API_FN * db2commexitDeregister)
(
 void * pConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 db2int32 state,
 db2int64 * pReservedFlags,
 char ** errmsg,
 db2int32 * errmsglen
);
```

## db2commexitDeregister API Parameters

### pConnectionContext

Input. A pointer to communication buffer exit library-specific data. This pointer is specific to the inbound connection. This parameter is passed as input to each function call for that connection. The database manager cannot access the memory pointed to by this parameter. This memory must be deallocated by this function.

### pCommInfo

Input. A pointer to a structure that contains information which identifies the database server and protocol-specific information for the incoming connection.

### State

Input. Indicates under which condition the function is called. Possible values are

- CONNECTION\_TERM - indicates that the physical connection with the client is terminated.
- AGENT\_DISASSOCIATION - indicates that the client connection is idle and the agent is disassociated from it.

### pReservedFlags

Input/Output. Reserved for future use. The value must be set to 0 on output.

### errmsg

Output. A pointer to the address of an ASCII error message string that is allocated by the communication buffer exit library. This error messages string might be returned in this parameter if the function execution is not successful. This memory is not freed by calling db2commexitFreeErrMsg.

### errmsglen

Output. A pointer to an integer that indicates the length, in bytes, of the error message string in the **errmsg** parameter.

## db2commexitRecv API - Receive

This function is called for each buffer that the database manager receives from a client. It is applicable only in communication buffer exit libraries.

This function is called by the database manager immediately after it receives a communication buffer from the client. The function is called after the buffer is decrypted so that the communication buffer exit library can access the unencrypted buffer.

## API header file

db2commexit.h

## API and data structure syntax

```
SQL_API_RC (SQL_API_FN * db2commexitRecv)
(
 void * pConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 const db2commexitBuffer * pBuffer,
 db2int64 * pReservedFlags,
 char ** errmsg,
 db2int32 * errmsglen
);
```

## db2commexitRecv API Parameters

### pConnectionContext

Input. A pointer to communication buffer exit library-specific data. This pointer is specific to the inbound connection. This parameter is passed as input to each function call for that connection. The database manager cannot access the memory pointed to by this parameter. This memory must be deallocated by this function.

### pCommInfo

Input. A pointer to a structure that contains information which identifies the database server and protocol-specific information for the incoming connection. Some of the fields in the structure are not setup until multiple buffers are exchanged with the client. The fields are available in later calls to db2commexitRecv and db2commexitSend. This scenario applies specifically to **inbound\_app1\_id**, **outbound\_app1\_id**, and **connection\_type**.

### pBuffer

Input. A pointer to a structure that contains the length of the buffer that is received by the database manager and a pointer to the buffer. If the buffer is encrypted, it is unencrypted before this function is called.

### pReservedFlags

Input/Output. The bit DB2COMMEXIT\_RECV\_IN\_FLAG\_END\_DECRYPT is set to indicate that this call is the final call to this function for a DSS that is encrypted. The length of the DSS that is passed as input indicates the length of the encrypted DSS. However, the DSS is then unencrypted and the padding removed. Since there is always padding, the length of the DSS is less than indicated. The length indicated in the pBuffer structure is the final data for the DSS. It is possible that it is zero if a full block size of padding is added.

For output, this value is reserved for future use. The value must be set to 0 on output.

### errmsg

Output. A pointer to the address of an ASCII error message string that is allocated by the communication buffer exit library. This error messages string might be returned in this parameter if the function execution is not successful. This memory is not freed by calling db2commexitFreeErrormsg.

**errmsglen**

Output. A pointer to an integer that indicates the length, in bytes, of the error message string in the **errmsg** parameter.

**db2commexitSend API - Send**

This function is called for each buffer that the database manager sends to a client. It is applicable only in communication buffer exit libraries.

This function is called by the database manager immediately before a communication buffer is sent to the client. The function is called before the buffer is encrypted so that the communication buffer exit library can access the unencrypted buffer.

**API header file**

db2commexit.h

**API and data structure syntax**

```
SQL_API_RC (SQL_API_FN * db2commexitSend)
(
 void * pConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 const db2commexitBuffer * pBuffer,
 db2int64 * pReservedFlags,
 char ** errmsg,
 db2int32 * errmsglen
);
```

**db2commexitSend API Parameters****pConnectionContext**

Input. A pointer to communication buffer exit library-specific data. This pointer is specific to the inbound connection. This parameter is passed as input to each function call for that connection. The database manager cannot access the memory pointed to by this parameter.

**pCommInfo**

Input. A pointer to a structure that contains information which identifies the database server and protocol-specific information for the incoming connection. Some of the fields in the structure are not setup until multiple buffers are exchanged with the client. The fields are available in later calls to db2commexitRecv and db2commexitSend. This scenario applies specifically to **inbound\_app1\_id**, **outbound\_app1\_id**, and **connection\_type**.

**pBuffer**

Input. A pointer to a structure that contains the length of the buffer that is sent to the client and a pointer to the buffer. If the buffer is encrypted, it is unencrypted before this function is called.

**pReservedFlags**

Input/Output. The bit DB2COMMEXIT\_SEND\_IN\_FLAG\_PURGE is set if the database manager encounters an error and must purge some buffers that were prepared to send to the client. Some of these buffers might be input to the communication buffer exit library.

For output, this value is reserved for future use. The value must be set to 0 on output.

**errmsg**

Output. A pointer to the address of an ASCII error message string that is



allocated by the communication buffer exit library. This error messages string might be returned in this parameter if the function execution is not successful. This memory is not freed by calling `db2commexitFreeErrorMsg`.

#### **errorMsgLen**

Output. A pointer to an integer that indicates the length, in bytes, of the error message string in the **errorMsg** parameter.

### **db2commexitUserIdentity API - User identity**

This function is called to provide the identity of the user for the current socket. It is applicable only in communication buffer exit libraries.

This function is called to inform the communication buffer exit library of the user name and session authorized ID used to establish the connection. The function is also called if these parameters change because of a trusted context switch user or SET SESSION AUTHORIZATION. The user name and session authorization ID are not determined until after the database manager authenticates the user. This function is not called until `db2commexitRegister` and multiple `db2commexitSend` and `db2commexitRecv` functions are called during authentication.

#### **API header file**

`db2commexit.h`

#### **API and data structure syntax**

```
SQL_API_RC (SQL_API_FN * db2commexitUserIdentity)
(
 void * pConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 db2int32 state,
 db2int32 usernameLen,
 const char * pUsername,
 db2int32 sessionAuthidLen,
 const char * pSessionAuthid,
 db2int64 * pReservedFlags,
 char ** errorMsg,
 db2int32 * errorMsgLen
);
```

### **db2commexitUserIdentity API Parameters**

#### **pConnectionContext**

Input. A pointer to communication buffer exit library-specific data. This pointer is specific to the inbound connection. This parameter is passed as input to each function call for that connection. The database manager does not access the memory pointed to by this parameter.

#### **pCommInfo**

Input. A pointer to a structure that contains information which identifies the database server and protocol-specific information for the incoming connection. Some of the fields in the structure are not setup until multiple buffers are exchanged with the client. The fields are available in later calls to `db2commexitRecv` and `db2commexitSend`. This scenario applies specifically to **inbound\_app1\_id**, **outbound\_app1\_id**, and **connection\_type**.

#### **State**

Input. Indicates under which condition the function is called. Possible values are:

- `DB2COMMEXIT_USERIDENT_NEW_CONNECTION` - a new connection.

- DB2COMMEXIT\_USERIDENT\_TC\_SWITCH\_USER - a trusted context switch user is issued.
- DB2COMMEXIT\_USERIDENT\_SET\_SESSION\_USER - SET SESSION AUTHORIZATION SQL statement is issued to change the session authorization ID.

**usernameLen**

Input. The length of **pUsername**.

**pUsername**

Input. The user name that is used to establish the connection.

**sessionAuthidLen**

Input. The length of **pSessionAuthid**.

**pSessionAuthid**

Input. The session authorization ID established for this connection.

**pReservedFlags**

Input/Output. Reserved for future use. The value must be set to 0 on output.

**errmsg**

Output. A pointer to the address of an ASCII error message string that is allocated by the communication buffer exit library. This error messages string might be returned in this parameter if the function execution is not successful. This memory is not freed by calling `db2commexitFreeErrMsg`.

**errmsglen**

Output. A pointer to an integer that indicates the length, in bytes, of the error message string in the **errmsg** parameter.

## **db2commexitFreeErrMsg API - Free error message memory**

This function frees the memory that is used to hold an error message from a previous API call. It is applicable for both types of communication exit libraries.

### **API header file**

`db2commexit.h`

### **API and data structure syntax**

```
SQL_API_RC (SQL_API_FN * db2commexitFreeErrMsg)
(char * errmsg);
```

### **db2commexitFreeErrMsg API Parameters**

**errmsg**

Input. A pointer to the error message returned from a previous API call.

## **Communication buffer exit library functions structure**

The `db2commexitInit` function takes a `void * commexit_fns` parameter. This parameter is cast to the version-specific structure which contains all of the functions that are implemented by the communication buffer exit library. The `db2commexitInit` function must assign the function pointers so that the database manager can call those functions.

The structure that must be completed, including a function pointer for each API, follows.

```

struct db2commexitFunctions_v1
{
 db2int32 version;

 SQL_API_RC (SQL_API_FN * db2commexitTerm)
 (
 char **errmsg,
 db2int32 *errmsglen
);

 SQL_API_RC (SQL_API_FN * db2commexitRegister)
 (
 void ** ppConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 db2int32 state,
 db2int64 * pReservedFlags,
 char ** errmsg,
 db2int32 * errmsglen
);

 SQL_API_RC (SQL_API_FN * db2commexitDeregister)
 (
 void * pConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 db2int32 state,
 db2int64 * pReservedFlags,
 char ** errmsg,
 db2int32 * errmsglen
);

 SQL_API_RC (SQL_API_FN * db2commexitRecv)
 (
 void * pConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 const db2commexitBuffer * pBuffer,
 db2int64 * pReservedFlags,
 char ** errmsg,
 db2int32 * errmsglen
);

 SQL_API_RC (SQL_API_FN * db2commexitSend)
 (
 void * pConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 const db2commexitBuffer * pBuffer,
 db2int64 * pReservedFlags,
 char ** errmsg,
 db2int32 * errmsglen
);

 SQL_API_RC (SQL_API_FN * db2commexitUserIdentity)
 (
 void * pConnectionContext,
 const db2commexitCommInfo_v1 * pCommInfo,
 db2int32 state,
 db2int32 usernameLen,
 const char * pUsername,
 db2int32 sessionAuthidLen,
 const char * pSessionAuthid,
 db2int64 * pReservedFlags,
 char ** errmsg,
 db2int32 * errmsglen
);

 SQL_API_RC (SQL_API_FN * db2commexitFreeErrorMsg)
 (

```

```

 char * errmsg;
);
};

```

## Communication buffer exit library information structure

The information structure indicates the communication protocol information for the current physical connection.

The db2commexitCommInfo\_v1 structure that is passed to each communication buffer exit library function follows. This structure is included in the db2commexit.h file.

```

struct db2commexitIPV4Info
{
 sockaddr_in client_sockaddr;
 sockaddr_in server_sockaddr;
};

struct db2commexitIPV6Info
{
 sockaddr_in6 client_sockaddr;
 sockaddr_in6 server_sockaddr;
};

struct db2commexitIPCInfo
{
 void * pSharedMemSegmentHandle;
};

struct db2commexitNamedPipeInfo
{
 void * handle;
};

struct db2commexitCommInfo_v1
{
 db2int32 clientProtocol; // SQL_PROTOCOL_ ...
 db2int32 connectionType; // unknown, local or gateway

 db2int32 hostnameLen;
 db2int32 instanceLen;
 db2int32 dbnameLen;
 db2int32 dbaliasLen;
 db2int32 inbound_appl_id_len;
 db2int32 outbound_appl_id_len;

 db2int32 clientPID; // Client PID
 db2int32 reserved2;

 db2NodeType member;

 char hostname[SQL_HOSTNAME_SZ+1];
 char instance[DB2COMMEXIT_INSTANCE_SZ + 1];
 char dbname[DB2COMMEXIT_DBNAME_SZ + 1];
 char dbalias[DB2COMMEXIT_DBNAME_SZ + 1];
 char inbound_appl_id[SQLM_APPLID_SZ + 1];
 char outbound_appl_id[SQLM_APPLID_SZ + 1];

 char reservedChar1[128];

 union
 {
 db2commexitIPV4Info ipv4Info;
 db2commexitIPV6Info ipv6Info;
 }
};

```

```

 db2commexitIPCInfo ipcInfo;
 db2commexitNamedPipeInfo namedPipeInfo;
 }
};

```

## Communication exit library buffer structure

The buffer structure is the structure that is passed as input to communication exit library functions.

The buffer structure follows:

```

struct db2commexitBuffer
{
 const unsigned char * pBuffer;
 db2int64 buffer_len;

 db2int32 reserved1;
 db2int32 reserved2;
};

```

## Communication buffer exit library control over connections

The communication buffer exit library can force a drop of the connection to the client at any time.

If the communication buffer exit library returns the appropriate error return code on any of the calls to `db2commexitUserIdentity`, `db2commexitRegister`, `db2commexitDeregister`, `db2commexitRecv`, or `db2commexitSend`, the database manager immediately closes the connection with the client.

This capability allows the communication buffer exit library to determine, based on the buffers reviewed, if some inappropriate activity is taking place. If such a determination is made, any further action by the database manager for that connection can be prevented.

## Communication exit library API versions

The Db2 database system supports multiple versions of communication exit library APIs. Versions are numbered with integers that start with 1.

The version number that the database manager passes to the security library initialization function is the highest supported version number of the API.

The communication exit library infrastructure supports both communication buffer exit libraries and runtime communication exit libraries. The input version parameter contains the highest version numbers for both sets of the libraries. Functions must use the `DB2COMMEXIT_GET_BUFFER_FN_VER` macro to obtain the highest supported version number of the function pointer structure for DRDA style functions. Functions must use the `DB2COMMEXIT_GET_RUNTIME_FN_VER` macro to obtain the highest supported version number of the function pointer structure for the runtime communication exit library functions.

This function must cast `commexit_fns` to `db2commexitFunctions_v1`, define the function pointers, and call the `DB2COMMEXIT_SET_BUFFER_FN_VER` macro to set the version number. To use the runtime communication exit library, this function must cast `commexit_fns` to `db2commexitRuntimeFunctions_v1`. The function must also define the function pointers, and call the `DB2COMMEXIT_SET_RUNTIME_FN_VER` macro to set the version number. Only one of the macros must be called by this function.

The version numbers of the communication exit library APIs change only when necessary. For example, when there are changes to the parameters of the APIs. Version numbers are not automatically changed with database manager release numbers.

The version numbers allow the introduction of new or changed APIs. Library support for older versions is maintained

## Communication exit library error handling and return codes

When an error occurs in a communication exit library API, the API can return an ASCII text string in the **errmsg** field. That ASCII text string provides a more specific description of the problem than the return code. The database manager writes this entire string into the db2diag log files.

The memory for these error messages must be allocated by the communication exit library. Therefore, the library must also provide an API to free this memory: db2commexitFreeErrorMsg.

In addition to the **errmsg** field, at initialization time a message log function pointer, logMessage\_fn, is passed to the communication buffer exit library. The library can use the function to log any debugging information to the db2diag log files. For example:

```
// Log an message indicate init successful
(*(logMessage_fn))(DB2COMMEXIT_LOG_CRITICAL,
 "comm exit initialization successful",
 strlen("comm. exit initialization successful"));
```

For more details about each parameter for the db2secLogMessage function, refer to the initialization API db2commexitInit in the related reference.

## Return codes

Table 42. Return codes that a communication exit library can return to the database manager.

Return code	Define value	Details
0	DB2COMMEXIT_SUCCESS	Successful execution
-1	DB2COMMEXIT_ERR_UNKNOWN	The library encountered an unexpected error.
-2	DB2COMMEXIT_ERR_DROP_CONNECTION	The library determined that the connection for which it was called must be terminated.

## Communication exit library development restrictions

Certain restrictions and considerations must be taken when you develop a communication exit library.

### Restrictions

#### C-linkage

The communication exit library must be written in C or C++ and linked with C-linkage. Header files that provide the prototypes, data structures that are required to implement the libraries, and error code definitions are provided only for C and C++. The function db2commexitInit must be declared extern "C" if the library is compiled as C++.

#### Signal handlers

The communication exit library must not install signal handlers or change the signal mask. Doing so interferes with the signal handlers of the database

manager. Interfering with the database manager signal handlers might seriously interfere with the ability to report and recover from errors.

#### **Exceptions**

The communication exit library APIs must not throw C++ exceptions. Such exceptions can interfere with database manager error handling.

#### **Thread-safe**

The communication exit library functions must be thread-safe. The `db2commexitInit` and `db2commexitTerm` functions are the only APIs that are not required to be thread-safe.

#### **Exit handlers**

The communication exit library must not install exit handlers or `pthread_atfork` handlers. The use of exit handlers is not supported because the communication exit library is unloaded before the database manager process exits.

#### **Fork/threads**

The communication exit library must not call, fork, or create new threads. This situation can lead to undefined behavior such as traps in the database manager.

#### **Library dependencies**

On Linux and UNIX, the communication exit library is loaded from a process that is `setuid` or `setgid`. It cannot rely on the `LD_LIBRARY_PATH`, `SHLIB_PATH`, or `LIBPATH` environment variables to find dependent libraries. Therefore, the library must not depend on more libraries, unless any dependent libraries are accessible through other methods, such as:

- The dependent libraries exist in `/lib` or `/usr/lib`.
- The directories in which dependent libraries are found are specified OS-wide (such as in the `ld.so.conf` file on Linux).
- Dependent libraries are specified in the `RPATH` in the library itself.

#### **Symbol collisions**

When possible, communication exit libraries might be compiled and linked with any available options that reduce the likelihood of symbol collisions. Such as, options that reduce unbound external symbolic references. For example, use of the `-Bsymbolic` linker option on HP, Solaris, and Linux can help prevent problems that are related to symbol collisions. However, for libraries that are written on AIX, do not use the `-brtl` linker option explicitly or implicitly.

#### **32-bit versus 64-bit considerations**

The database manager has both 32-bit and 64-bit versions, depending on the operating system. A 32-bit communication exit library must be enabled on a 32-bit database manager. A 64-bit communication exit library must be enabled on a 64-bit database manager. You cannot mix the two.

#### **Stored procedures, triggers, and other internal SQL**

Stored procedure interaction with the server is passed onto the communication exit library. Much of the interaction does not occur over standard communication channels and does not fit the model that is used for the exit library. Similarly, triggers, and other sources of internal SQL do not pass over standard communication channels and are not passed onto the communication exit library.

#### **Communication buffers must not be manipulated**

It is expected that the communication exit library does not manipulate or change the buffers that it is passed.

### **Rolling updates support**

Db2 supports updating the fix pack level of individual members in Db2 pureScale environments without stopping other members. This operation is known as rolling updates. Similarly, it is possible to update the level of the library that is used on individual members. It is possible that two different versions of the communication exit library might be running simultaneously on two different members. Similarly, each member can be at a different fix pack level. The communication exit library must tolerate such conditions without error.

### **Loading plug-in libraries on AIX with an extension of .a or .so**

On AIX, security plug-in libraries can have a file name extension of .a or .so. The mechanism that is 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 which contain shared object members. These members must be named shr.o for 32-bit or shr64.o for 64-bit. A single archive can contain both the 32-bit and 64-bit members, allowing it to be deployed on both types of operating systems.
- 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 32-bit or 64-bit depending on the compiler and linker options that are used when it was built.

On all operating systems, other than AIX, security plug-in libraries are always assumed to be dynamically loadable shared objects.

## **Communication exit library API calling sequences**

API calling sequences differ depending on specific scenarios and which exit library you use.

The following topics outline specific scenarios that you must be aware of when you develop communication exit libraries. Some topics apply to only communication buffer exit libraries. Some topics apply to only runtime communication exit libraries. Some topics apply to both types of communication exit libraries. The topics help you determine the calling sequence most appropriate for your environment.

### **API calling sequence - Normal connect in a single agent**

The most typical scenario is a client that connects to the database manager, issuing some SQL, and then disconnecting. This API calling sequence applies to communication buffer exit libraries.

In this case, a single agent, or thread handles the connection, and the following calls are made:

1. db2commexitRegister for a new socket connection.
2. db2commexitRecv and db2commexitSend to handle authentication, possibly multiple times.
3. db2commexitUserIdentity for a new connection
4. db2commexitRecv and db2commexitSend to handle clients SQL requests, possibly multiple times.
5. db2commexitDeregister to terminate socket connection.



## API calling sequence - Connect without a connect reset

This scenario covers a connect over an existing socket. The client might initiate another SQL connection without first issuing a connect reset. Two API calling sequences are illustrated. One shows how to implement the sequence for a runtime communication exit library. The other shows how to implement the sequence for a communication buffer exit library.

### Communication buffer exit library

When the database manager receives the SQL connect statement from the client, it implicitly drives an internal connect reset before it continues with the connect. Regular requests and replies flow back and forth as there is no change to the status of the socket. In this case, a single agent is handling all requests. As the buffers that contain the connect request from the client is made available through db2commexitRecv, the communication buffer exit library is able to determine a new connect is started when the buffer is parsed. The following calls are made:

1. db2commexitRegister for a new socket connection.
2. db2commexitRecv and db2commexitSend to handle authentication, possibly multiple times.
3. db2commexitUserIdentity for a new connection.
4. db2commexitRecv and db2commexitSend to handle client SQL requests, possibly multiple times.
5. db2commexitRecv and db2commexitSend to handle authentication, possibly multiple times.
6. db2commexitUserIdentity for a new connection.
7. db2commexitRecv and db2commexitSend to handle client SQL requests, possibly multiple times.
8. db2commexitDeregister to terminate socket connection.

#### Note:

db2commexitRegister and db2commexitDeregister are called only a single time each, even though the database manager processed two SQL connections.

### Runtime communication exit library

When the database manager receives the SQL connect statement from the client, it implicitly drives an internal connect reset before it continues with the connect. In this case, a single agent is handling all requests. The following calls are made:

1. db2commexitSessionInit for a new connection.
2. db2commexitSQL\* to handle SQL requests.
3. db2commexitSessionTerm to close a connection.
4. db2commexitSessionInit for a new connection.
5. db2commexitSQL\* to handle SQL requests.
6. db2commexitSessionTerm to close a connection.

## API calling sequence - Trusted context and switch user

This scenario is similar to connecting without a connect reset. The difference is the client requests a trusted context switch user rather than sending a new SQL connect request. Two API calling sequences are illustrated. One shows how to implement the sequence for a runtime communication exit library. The other shows how to implement the sequence for a communication buffer exit library.

## Communication buffer exit library

The following calls are made:

1. db2commexitRegister for a new socket connection.
2. db2commexitRecv and db2commexitSend to handle authentication, possibly multiple times.
3. db2commexitUserIdentity for a new connection
4. db2commexitRecv and db2commexitSend to handle clients SQL requests, possibly multiple times.
5. db2commexitRecv and db2commexitSend to handle authentication, possibly multiple times.

At some future point, the client sends a trusted context switch user request to the server to switch the user for the connection.

6. db2commexitUserIdentity for a trusted context switch user.
7. db2commexitRecv and db2commexitSend to handle clients SQL requests, possibly multiple times.
8. db2commexitDeregister to terminate socket connection.

## Runtime communication exit library

The following calls are made:

1. db2commexitSessionInit for a new connection.
2. db2commexitSQL\* to handle SQL requests.
3. db2commexitSessionTerm to close a user session.
4. db2commexitSessionInit for a new user session.
5. db2commexitSQL\* to handle SQL requests.
6. db2commexitSessionTerm to close a connection.

## API calling sequence - Connection concentrator

This scenario covers the API calling sequence when connection concentrator is used. The connection concentrator feature allows the database manager to handle many more clients than there are coordinating agents or threads. This API calling sequence applies to communication buffer exit libraries.

When a client reaches a unit of work boundary and does not send another request immediately, client sockets are placed into an idle pool. The agent that previously handled client requests moves on to another client. When the idle socket has data to read, a dispatcher finds an idle agent to handle it. Over the life of an SQL connection, there might be multiple agents that handle the client requests. Each time the socket is moved in and out of the idle pool, db2commexitDeregister and db2commexitRegister are called. The following calls are made:

1. db2commexitRegister for a new socket connection.
2. db2commexitRecv and db2commexitSend to handle authentication, possibly multiple times.
3. db2commexitUserIdentity for a new connection
4. db2commexitRecv and db2commexitSend to handle client SQL requests, possibly multiple times.

The client does not send another request immediately and the socket is placed into an idle pool.

5. db2commexitDeregister to disassociate with the agent.

At some future point, the client sends another request, at which point the dispatcher chooses an idle agent. The agent is likely a different one than used previously:

6. `db2commexitRegister` to associate an agent.
7. `db2commexitRecv` and `db2commexitSend` to handle client SQL requests, possibly multiple times.
8. `db2commexitDeregister` to terminate socket connection.

**Note:** There are multiple calls to `db2commexitRegister` and `db2commexitDeregister` for a single SQL connection.

### **API calling sequence - SET SESSION AUTHORIZATION statement**

This scenario covers the API calling sequence when the SET SESSION AUTHORIZATION statement is used. Two API calling sequences are illustrated. One shows how to implement the sequence for a runtime communication exit library. The other shows how to implement the sequence for a communication buffer exit library.

#### **Communication buffer exit library**

The SET SESSION AUTHORIZATION statement changes the session authorization ID in use for the current connection. `Db2commexitUserIdentity` is called to inform the communication buffer exit library that identity information changed for the current connection. The following calls are made:

1. `db2commexitRegister` for a new socket connection.
2. `db2commexitRecv` and `db2commexitSend` to handle authentication, possibly multiple times.
3. `db2commexitUserIdentity` for a new connection.
4. `db2commexitRecv` and `db2commexitSend` to handle clients SQL requests, possibly multiple times.

The user issues a SET SESSION AUTHORIZATION statement. This request is passed to `db2commexitRecv`. It is no different from other SQL statement.

5. `db2commexitUserIdentity` for a SET SESSION AUTHORIZATION.
6. `db2commexitRecv` and `db2commexitSend` to handle client SQL requests, possibly multiple times.
7. `db2commexitDeregister` to terminate socket connection.

#### **Runtime communication exit library**

The SET SESSION AUTHORIZATION statement changes the session authorization ID in use for the current connection. `Db2commexitSetSessionAuth` is called to inform the communication buffer exit library that identity information changed for the current connection. The following calls are made:

1. `db2commexitSessionInit` for a new connection.
2. `db2commexitSQL*` to handle new SQL requests.

The user issues a SET SESSION AUTHORIZATION statement. This request is no different from other SQL statement

3. `db2commexitSetSessionAuth` for a SET SESSION AUTHORIZATION.
4. `db2commexitSQL*` to handle new SQL requests.
5. `db2commexitSessionTerm` to close the connection.

## Considerations for setting the target logical node

Considerations must be taken when you set the target logical node with the *DB2NODE* variable, or with the **SET CLIENT** command. The information applies to communication buffer exit libraries.

In a partitioned database environment, if the client specifies a member through the *DB2NODE* variable that is not the member it is configured to connect to, the database manager switches the connection to the new member specified in the variable. The client connection is forwarded through the connected member to the remote member. In this case, the communication buffer exit library is called at both members. There are a few features to note:

- At the connected member, the client address reflects the actual client.
- At the remote member, the client address reflects the connected member.
- The outbound application id at the connected member is the same as the inbound application id at the remote member.
- At the connected member, the database alias used reflects the database alias provided by the actual client.
- At the remote member, the database alias used is the actual database name.

When the application IDs are established, the **connectionType** in the *db2commexitCommInfo\_v1* structure is set to GATEWAY.

## Considerations for a connect gateway

Considerations must be taken when the database manager acts as a connect gateway to another DRDA database server.

When Db2 acts as a connect gateway, the communication exit library is called in the same manner as a standard connection. When authentication is complete and the application IDs are established, the **connectionType** in the *db2commexitCommInfo\_v1* structure is set to GATEWAY. The *outbound\_application\_id* matches the application ID for the connection at the DRDA database server.

## Considerations for DATA\_ENCRYPT

Considerations must be taken when the *DATA\_ENCRYPT* authentication type is used. The information applies to only communication buffer exit libraries.

The handling of communications that is protected with the authentication type *DATA\_ENCRYPT* requires special mention. Unlike SSL, the encryption and decryption necessary to support *DATA\_ENCRYPT* is run by the database manager. It is run after data is received from the client and before a reply is sent to the client.

## Receive and DATA\_ENCRYPT

When an encrypted DSS is received from the client, the buffer is decrypted as needed by the database manager. That is, the whole buffer is not decrypted all at one time. The communication buffer exit library is called with the decrypted data as it is decrypted.

The DSS length, or the DSS continuation length if the DSS is longer than a logical record, contains the length of the encrypted DSS. It does not contain the length of the decrypted buffer. As the encryption always adds padding, this length is always larger than the plaintext length. The length of the padding for DSS is a maximum of 8 bytes.

When the final call to `db2CommexitRecv` is made, the `DB2COMMEXIT_RECV_IN_FLAG_END_DECRYPT` flag is passed as input to indicate the end of the encrypted DSS.

**Note:** It is possible the length in such a case is 0, indicating that a full block size of padding is added.

### **Send and DATA\_ENCRYPT**

When a DSS reply to the client is encrypted, multiple plaintext DSS and encrypted DSS might make up the buffer which is sent to the client. As these DSS are prepared, they are passed as input to the `db2commexitSend` routine. These passes are done one at a time as the plaintext data must be used as input before encryption. The database manager might receive an error condition which requires it to purge previously prepared, but not sent, DSS. The communication buffer exit library might already know about these libraries. The `db2CommexitSend` function is called with a length of 0 and a flag `DB2COMMEXIT_SEND_IN_FLAG_PURGE` indicating that a purge occurred.



---

## Chapter 11. Audit facility record layouts

When an audit record is extracted from the audit log, each record has one of the formats shown in the following tables. Each table is preceded by a sample record.

The description of each item in the record is shown one row at a time in the associated table. Each item is shown in the table in the same order as it is output in the delimited file after the extract operation.

**Note:**

1. Depending on the audit event, not all fields in the audit records will have values. When there is no values in the field, the field will not be shown in the audit output.
2. Some fields such as “Access Attempted” are stored in the delimited ASCII format as bit maps. In this flat report file, however, these fields appear as a set of strings representing the bit map values.

---

### Audit record object types

The following table shows for each audit record object type whether it can generate CHECKING, OBJMAINT, and SECMAINT events.

*Table 43. Audit Record Object Types Based on Audit Events*

Object type	CHECKING events	OBJMAINT events	SECMAINT events
ACCESS_RULE			X
ALIAS	X	X	
ALL	X		
AUDIT_POLICY	X	X	
BUFFERPOOL	X	X	
CHECK_CONSTRAINT		X	
DATABASE	X		X
DATA TYPE		X	
EVENT_MONITOR	X	X	
FOREIGN_KEY		X	
FUNCTION	X	X	X
FUNCTION MAPPING	X	X	
GLOBAL_VARIABLE	X	X	X
HISTOGRAM TEMPLATE	X	X	
INDEX	X	X	X
INDEX EXTENSION		X	
INSTANCE	X		
JAR_FILE		X	
MASK	X	X	X
METHOD_BODY	X	X	X
MODULE	X	X	X

Table 43. Audit Record Object Types Based on Audit Events (continued)

Object type	CHECKING events	OBJMAINT events	SECMAINT events
NICKNAME	X	X	X
NODEGROUP	X	X	
NONE	X	X	X
OPTIMIZATION PROFILE	X		
PACKAGE	X	X	X
PACKAGE CACHE	X		
PERMISSION	X	X	X
PRIMARY_KEY		X	
REOPT_VALUES	X		
ROLE	X	X	X
SCHEMA	X	X	X
SECURITY LABEL		X	X
SECURITY LABEL COMPONENT		X	
SECURITY POLICY		X	X
SEQUENCE	X	X	
SERVER	X	X	X
SERVER OPTION	X	X	
SERVICE CLASS	X	X	
STORED_PROCEDURE	X	X	X
SUMMARY TABLES	X	X	X
TABLE	X	X	X
TABLESPACE	X	X	X
THRESHOLD	X	X	
TRIGGER		X	
TRUSTED CONTEXT	X	X	X
TYPE MAPPING	X	X	
TYPE&TRANSFORM	X	X	
UNIQUE_CONSTRAINT		X	
USER MAPPING	X	X	
USER_TEMPORARY_TABLE	X	X	X
VIEW	X	X	X
WORK ACTION SET	X	X	
WORK CLASS SET	X	X	
WORKLOAD	X	X	X
WRAPPER	X	X	
XSR object	X	X	X



## Audit record layout for AUDIT events

The following table shows the layout of the audit record for AUDIT events.

Sample audit record:

```
timestamp=2007-04-10-08.29.52.000001;
category=AUDIT;
audit_event=START;
event_correlator=0;
event_status=0;
userid=newton;
authid=NEWTON;
application_id=*LOCAL_APPLICATION;
application_name=db2audit.exe;
```

Table 44. Audit Record Layout for AUDIT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are:  AUDIT
Audit Event	VARCHAR(32)	Specific Audit Event.  For a list of possible values, refer to the section for the AUDIT category in “Audit events” on page 354.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where  Successful event > = 0 Failed event < 0
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
Origin Node Number	SMALLINT	Member number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Member number of the coordinator member.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section	SMALLINT	Section number in package being used at the time the audit event occurred
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.

Table 44. Audit Record Layout for AUDIT Events (continued)

NAME	FORMAT	DESCRIPTION
Client Workstation Name	VARCHAR(255)	The value of the CURRENT_CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT_CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT_CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(255)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	CHAR(1)	Possible values are:  " - NONE '1' - IMPLICIT_TRUSTED_CONNECTION '2' - EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.
Policy Name	VARCHAR(128)	The audit policy name.
Policy Association Object Type	CHAR(1)	The type of the object that the audit policy is associated with. Possible values include: <ul style="list-style-type: none"> <li>• N = Nickname</li> <li>• S = MQT</li> <li>• T = Table (untyped)</li> <li>• i = Authorization ID</li> <li>• g= Authority</li> <li>• x = Trusted context</li> <li>• blank = Database</li> </ul>
Policy Association Subobject Type	CHAR(1)	The type of sub-object that the audit policy is associated with. If the Object Type is ? (authorization id), then possible values are: <ul style="list-style-type: none"> <li>• U = User</li> <li>• G = Group</li> <li>• R = Role</li> </ul>
Policy Association Object Name	VARCHAR(128)	The name of the object that the audit policy is associated with.
Policy Association Object Schema	VARCHAR(128)	The schema name of the object that the audit policy is associated with. This is NULL if the Policy Association Object Type identifies an object to which a schema does not apply.
Audit Status	CHAR(1)	The status of the AUDIT category in an audit policy. Possible values are: <ul style="list-style-type: none"> <li>• B-Both</li> <li>• F-Failure</li> <li>• N-None</li> <li>• S-Success</li> </ul>
Checking Status	CHAR(1)	The status of the CHECKING category in an audit policy. Possible values are: <ul style="list-style-type: none"> <li>• B-Both</li> <li>• F-Failure</li> <li>• N-None</li> <li>• S-Success</li> </ul>

Table 44. Audit Record Layout for AUDIT Events (continued)

NAME	FORMAT	DESCRIPTION
Context Status	CHAR(1)	The status of the CONTEXT category in an audit policy. Possible values are: <ul style="list-style-type: none"> <li>• B-Both</li> <li>• F-Failure</li> <li>• N-None</li> <li>• S-Success</li> </ul>
Execute Status	CHAR(1)	The status of the EXECUTE category in an audit policy. Possible values are: <ul style="list-style-type: none"> <li>• B-Both</li> <li>• F-Failure</li> <li>• N-None</li> <li>• S-Success</li> </ul>
Execute With Data	CHAR(1)	The WITH DATA option of the EXECUTE category in the audit policy. Possible values are: <ul style="list-style-type: none"> <li>• Y-WITH DATA</li> <li>• N-WITHOUT DATA</li> </ul>
Objmaint Status	CHAR(1)	The status of the OBJMAINT category in an audit policy. Possible values are: <ul style="list-style-type: none"> <li>• B-Both</li> <li>• F-Failure</li> <li>• N-None</li> <li>• S-Success</li> </ul>
Secmaint Status	CHAR(1)	The status of the SECMAINT category in an audit policy. See Audit Status field for possible values.
Sysadmin Status	CHAR(1)	The status of the SYSADMIN category in an audit policy. Possible values are: <ul style="list-style-type: none"> <li>• B-Both</li> <li>• F-Failure</li> <li>• N-None</li> <li>• S-Success</li> </ul>
Validate Status	CHAR(1)	The status of the VALIDATE category in an audit policy. Possible values are: <ul style="list-style-type: none"> <li>• B-Both</li> <li>• F-Failure</li> <li>• N-None</li> <li>• S-Success</li> </ul>
Error Type	CHAR(8)	The error type in an audit policy. Possible values are: AUDIT and NORMAL.
Data Path	VARCHAR(1024)	The path to the active audit logs specified on the <b>db2audit configure</b> command.
Archive Path	VARCHAR(1024)	The path to the archived audit logs specified on the <b>db2audit configure</b> command
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.

## Audit record layout for CHECKING events

The format of the audit record for CHECKING events is shown in the following table.

Sample audit record:

```
timestamp=1998-06-24-08.42.11.622984;
category=CHECKING;
audit_event=CHECKING_OBJECT;
event_correlator=2;
event_status=0;
database=F00;
userid=boss;
authid=BOSS;
application_id=*LOCAL.newton.980624124210;
application_name=testapp;
package_schema=NULLID;
package_name=SYSSH200;
package_section=0;
object_schema=GSTAGER;
object_name=NONE;
object_type=REOPT_VALUES;
access_approval_reason=DBADM;
access_attempted=STORE;
```

Table 45. Audit record layout for CHECKING events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are:  CHECKING
Audit Event	VARCHAR(32)	Specific Audit Event.  For a list of possible values, refer to the section for the CHECKING category in “Audit events” on page 354.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where  Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Member number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Member number of the coordinator Member.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.

Table 45. Audit record layout for CHECKING events (continued)

NAME	FORMAT	DESCRIPTION
Object Schema	VARCHAR(128)	Schema of the object for which the audit event was generated.
Object Name	VARCHAR(128)	Name of object for which the audit event was generated.
Object Type	VARCHAR(32)	Type of object for which the audit event was generated. Possible values include: those shown in the topic titled "Audit record object types".
Access Approval Reason	CHAR(34)	Indicates the reason why access was approved for this audit event. Possible values include: those shown in the topic titled "List of possible CHECKING access approval reasons".
Access Attempted	CHAR(34)	Indicates the type of access that was attempted. Possible values include: those shown in the topic titled "List of possible CHECKING access attempted types".
Package Version	VARCHAR (64)	Version of the package in use at the time that the audit event occurred.
Checked Authorization ID	VARCHAR(128)	Authorization ID is checked when it is different than the authorization ID at the time of the audit event. For example, this can be the target owner in a TRANSFER OWNERSHIP statement.  When the audit event is SWITCH_USER, this field represents the authorization ID that is switched to.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(255)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	CHAR(1)	Possible values are:  " - NONE '1' - IMPLICIT_TRUSTED_CONNECTION '2' - EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.

## CHECKING access approval reasons

The following list shows the possible CHECKING access approval reasons.

Note that an audit record might contain multiple access approval reasons, for example: access approval reason=DATAACCESS,ACCESSCTRL;. When multiple access approval reasons are present, the user must have all stated authorities and privileges in order to pass the authorization check for the attempted access.

**0x00000000000000000000000000000001 ACCESS DENIED**

Access is not approved; rather, it was denied.

**0x00000000000000000000000000000002 SYSADM**

Access is approved; the application or user has SYSADM authority.

**0x00000000000000000000000000000004 SYSCTRL**

Access is approved; the application or user has SYSCTRL authority.

**0x00000000000000000000000000000008 SYSMANT**

Access is approved; the application or user has SYSMANT authority.

**0x00000000000000000000000000000010 DBADM**

Access is approved; the application or user has DBADM authority.

**0x00000000000000000000000000000020 DATABASE**

Access is approved; the application or user has an explicit privilege on the database.

**0x00000000000000000000000000000040 OBJECT**

Access is approved; the application or user has a privilege on the object or function.

**0x00000000000000000000000000000080 DEFINER**

Access is approved; the application or user is the definer of the object or function.

**0x00000000000000000000000000000100 OWNER**

Access is approved; the application or user is the owner of the object or function.

**0x00000000000000000000000000000200 CONTROL**

Access is approved; the application or user has CONTROL privilege on the object or function.

**0x00000000000000000000000000000400 BIND**

Access is approved; the application or user has bind privilege on the package.

**0x00000000000000000000000000000800 SYSQUIESCE**

Access is approved; if the instance or database is in quiesce mode, the application or user may connect or attach.

**0x00000000000000000000000000001000 SYSMON**

Access is approved; the application or user has SYSMON authority.

**0x00000000000000000000000000002000 SECADM**

Access is approved; the application or user has SECADM authority.

**0x00000000000000000000000000004000 SETSESSIONUSER**

Access is approved; the application or user has SETSESSIONUSER authority.

**0x00000000000000000000000000008000 TRUSTED\_CONTEXT\_MATCH**

Connection attributes matched the attributes of a unique trusted context defined at the Db2 server.

**0x00000000000000000000000000010000 TRUSTED\_CONTEXT\_USE**

Access is approved to use a trusted context.

Access is approved; the application or user has SQLADM authority.

Access is approved; the application or user has WLMADM authority.

Access is approved; the application or user has EXPLAIN authority.

Access is approved; the application or user has DATAACCESS authority.

Access is approved; the application or user has ACCESSCTRL authority.

Access is approved; the application or user has the SECUREOBJECTAUTH authority.

## CHECKING access attempted types

The following list shows the possible CHECKING access attempted types.

If Audit Event is CHECKING\_TRANSFER, then the audit entry reflects that a privilege is held or not.

Attempt to verify whether CONTROL privilege is held.

Attempt to alter an object or to verify whether ALTER privilege is held if Audit Event is CHECKING\_TRANSFER.

Attempt to delete an object or to verify whether DELETE privilege is held if Audit Event is CHECKING TRANSFER.

Attempt to use an index or to verify whether INDEX privilege is held if Audit Event is CHECKING TRANSFER.

Attempt to insert into an object or to verify whether INSERT privilege is held if Audit Event is CHECKING TRANSFER.

Attempt to query a table or view or to verify whether SELECT privilege is held if Audit Event is CHECKING TRANSFER.

Attempt to update data in an object or to verify whether UPDATE privilege is held if Audit Event is CHECKING\_TRANSFER.

Attempt to establish referential constraints between objects or to verify whether REFERENCE privilege is held if Audit Event is CHECKING TRANSFER.

Attempt to create an object.

Attempt to drop an object.

330 Db2 11.1 for Linux, UNIX, and Windows: Database Security Guide



```
0x00 FLUSH
 Attempt to execute the FLUSH statement.

0x00 STORE
 Attempt to view the values of a reoptimized statement in the
 EXPLAIN_PREDICATE table.

0x00 SET_OWNER
 Attempt to set an owner that does not match the current user when
 binding a package.

0x00 SET_PASSTHRU
 Attempt to issue the SET PASSTHRU statement.

0x00 TRANSFER
 Attempt to transfer an object.

0x00 ALTER_WITH_GRANT
 Attempt to verify whether ALTER with GRANT privilege is held.

0x00 DELETE_WITH_GRANT
 Attempt to verify whether DELETE with GRANT privilege is held.

0x00 INDEX_WITH_GRANT
 Attempt to verify whether INDEX with GRANT privilege is held

0x00 INSERT_WITH_GRANT
 Attempt to verify whether INSERT with GRANT privilege is held.

0x00 SELECT_WITH_GRANT
 Attempt to verify whether SELECT with GRANT privilege is held.

0x00 UPDATE_WITH_GRANT
 Attempt to verify whether UPDATE with GRANT privilege is held.

0x00 REFERENCE_WITH_GRANT
 Attempt to verify whether REFERENCE with GRANT privilege is held.

0x00 USAGE
 Attempt to use a sequence or an XSR object or to verify whether USAGE
 privilege is held if Audit Event is CHECKING_TRANSFER.

0x00 SET_ROLE
 Attempt to set a role.

0x00 EXPLICIT_TRUSTED_CONNECTION
 Attempt to establish an explicit trusted connection.

0x00 IMPLICIT_TRUSTED_CONNECTION
 Attempt to establish an implicit trusted connection.

0x00 READ
 Attempt to read a global variable.

0x00 WRITE
 Attempt to write a global variable.

0x00 SWITCH_USER
 Attempt to switch a user ID on an explicit trusted connection.

0x00 AUDIT_USING
 Attempt to associate an audit policy with an object.

0x00 AUDIT_REPLACE
 Attempt to replace an audit policy association with an object.
```

0x00000000000000000000000080000000000000	AUDIT_REMOVE
Attempt to remove an audit policy association with an object.	
0x00000000000000000000000010000000000000	AUDIT_ARCHIVE
Attempt to archive the audit log.	
0x00000000000000000000000020000000000000	AUDIT_EXTRACT
Attempt to extract the audit log.	
0x00000000000000000000000040000000000000	AUDIT_LIST_LOGS
Attempt to list the audit logs.	
0x00000000000000000000000080000000000000	IGNORE_TRIGGERS
Attempt to ignore the triggers associated with a database object.	
0x00000000000000000000000010000000000000	PREPARE
Attempt to prepare an SQL statement and the user does not hold the necessary object level privilege or DATAACCESS authority.	
0x00000000000000000000000020000000000000	DESCRIBE
Attempt to describe a statement and the user does not hold the necessary object level privilege or DATAACCESS authority.	
0x00000000000000000000000040000000000000	SET_USAGELIST
Attempt to set the usage list state.	

## Audit record layout for OBJMAINT events

The format of the audit record for OBJMAINT events is shown in the following table.

Sample audit record:

```
timestamp=1998-06-24-08.42.41.957524;
category=OBJMAINT;
audit event=CREATE_OBJECT;
event correlator=3;
event status=0;
database=F00;
userid=boss;
authid=BOSS;
application id=*LOCAL.newton.980624124210;
application name=testapp;
package schema=NULLID;
package name=SQLC28A1;
package section=0;
object schema=BOSS;
object name=AUDIT;
object type=TABLE;
```

*Table 46. Audit Record Layout for OBJMAINT Events*

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are:  OBJMAINT
Audit Event	VARCHAR(32)	Specific Audit Event.  For a list of possible values, refer to the section for the OBJMAINT category in “Audit events” on page 354.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.

Table 46. Audit Record Layout for OBJMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where  Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Member number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Member number of the coordinator member.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(256)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR(128)	Schema of the object for which the audit event was generated.
Object Name	VARCHAR(128)	Name of object for which the audit event was generated.
Object Type	VARCHAR(32)	Type of object for which the audit event was generated. Possible values include: those shown in the topic titled "Audit record object types".
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Security Policy Name	VARCHAR(128)	The name of the security policy if the object type is TABLE and that table is associated with a security policy.

Table 46. Audit Record Layout for OBJMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Alter Action	VARCHAR(32)	Specific Alter operation  Possible values include: <ul style="list-style-type: none"> <li>• ADD_PROTECTED_COLUMN</li> <li>• ADD_COLUMN_PROTECTION</li> <li>• DROP_COLUMN_PROTECTION</li> <li>• ADD_ROW_PROTECTION</li> <li>• ADD_SECURITY_POLICY</li> <li>• ADD_ELEMENT</li> <li>• ADD COMPONENT</li> <li>• USE GROUP AUTHORIZATIONS</li> <li>• IGNORE GROUP AUTHORIZATIONS</li> <li>• USE ROLE AUTHORIZATIONS</li> <li>• IGNORE ROLE AUTHORIZATIONS</li> <li>• OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL</li> <li>• RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL</li> <li>• SECURE</li> <li>• UNSECURE</li> <li>• ENABLE</li> <li>• DISABLE</li> <li>• ACTIVATE_ROW_ACCESS_CONTROL</li> <li>• ACTIVATE_COLUMN_ACCESS_CONTROL</li> <li>• ACTIVATE_ROW_COLUMN_ACCESS_CONTROL</li> </ul>
Protected Column Name	VARCHAR(128)	If the Alter Action is ADD_COLUMN_PROTECTION or DROP_COLUMN_PROTECTION this is the name of the affected column.
Column Security Label	VARCHAR(128)	The security label protecting the column specified in the field Column Name.
Security Label Column Name	VARCHAR(128)	Name of the column containing the security label protecting the row.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT_USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(255)	The name of the trusted context associated with the trusted connection.

Table 46. Audit Record Layout for OBJMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Connection Trust Type	CHAR(1)	Possible values are:  " - NONE '1' - IMPLICIT_TRUSTED_CONNECTION '2' - EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.
Object Module	VARCHAR(128)	Name of module to which the object belongs.
Associated Object Name	VARCHAR(128)	Name of the object for which an association exists. The meaning of the association depends on the Object Type for the event. If the Object Type is PERMISSION or MASK, then the associated object is the table on which the permission or mask has been created.
Associated Object Schema	VARCHAR(128)	Name of the object schema for which an association exists. The meaning of the association depends on the Object Type for the event.
Associated Object Type	VARCHAR(128)	The type of the object for which an association exists. The meaning of the association depends on the Object Type for the event.
Associated Subobject Type	VARCHAR(128)	The type of the subobject for which an association exists. The meaning of the association depends on the Object Type for the event. If the Object Type is MASK and the associated object type is TABLE, then the associated subobject is the column of the table on which the mask has been created.
Associated Subobject Name	VARCHAR(128)	Name of the subobject for which an association exists. The meaning of the association depends on the Object Type for the event.
Secured	VARCHAR(32)	Specifies if the object is a secured object.
State	VARCHAR(32)	The state of the object. The state depends on the Object Type. Possible values include: • ENABLED • DISABLED
Access Control	VARCHAR(32)	Specifies what access control the object is protected with. Possible values include: • ROW - Row access control has been activated on the object • COLUMN - Column access control has been activated on the object • ROW_COLUMN - Row and column access control has been activated on the object
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.

## Audit record layout for SECMAINT events

The format of the audit record for SECMAINT events is shown in the following table.

Sample audit record:

```
timestamp=1998-06-24-11.57.45.188101;
category=SECMAINT;
audit event=GRANT;
event correlator=4;
event status=0;
database=F00;
```

```

userid=boss;
authid=BOSS;
application id=*LOCAL.boss.980624155728;
application name=db2bp;
package schema=NULLID;
package name=SQLC28A1;
package section=0;
object schema=BOSS;
object name=T1;
object type=TABLE;
grantor=BOSS;
grantee=WORKER;
grantee type=USER;
privilege=SELECT;

```

Table 47. Audit Record Layout for SECMAINT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are:  SECMAINT
Audit Event	VARCHAR(32)	Specific Audit Event.  For a list of possible values, refer to the section for the SECMAINT category in “Audit events” on page 354.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where  Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Member number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Member number of the coordinator member.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR(128)	Schema of the object for which the audit event was generated.  If the object type field is ACCESS_RULE then this field contains the security policy name associated with the rule. The name of the rule is stored in the field Object Name.  If the object type field is SECURITY_LABEL, then this field contains the name of the security policy that the security label is part of. The name of the security label is stored in the field Object Name.

Table 47. Audit Record Layout for SECMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Object Name	VARCHAR(128)	<p>Name of object for which the audit event was generated.</p> <p>Represents a role name when the audit event is any of:</p> <ul style="list-style-type: none"> <li>• ADD_DEFAULT_ROLE</li> <li>• DROP_DEFAULT_ROLE</li> <li>• ALTER_DEFAULT_ROLE</li> <li>• ADD_USER</li> <li>• DROP_USER</li> <li>• ALTER_USER_ADD_ROLE</li> <li>• ALTER_USER_DROP_ROLE</li> <li>• ALTER_USER_AUTHENTICATION</li> </ul> <p>If the object type field is ACCESS_RULE then this field contains the name of the rule. The security policy name associated with the rule is stored in the field Object Schema.</p> <p>If the object type field is SECURITY_LABEL, then this field contains the name of the security label. The name of the security policy that it is part of is stored in the field Object Schema.</p>
Object Type	VARCHAR(32)	<p>Type of object for which the audit event was generated. Possible values include: those shown in the topic titled "Audit record object types".</p> <p>The value is ROLE when the audit event is any of:</p> <ul style="list-style-type: none"> <li>• ADD_DEFAULT_ROLE</li> <li>• DROP_DEFAULT_ROLE</li> <li>• ALTER_DEFAULT_ROLE</li> <li>• ADD_USER</li> <li>• DROP_USER</li> <li>• ALTER_USER_ADD_ROLE</li> <li>• ALTER_USER_DROP_ROLE</li> <li>• ALTER_USER_AUTHENTICATION</li> </ul>
Grantor	VARCHAR(128)	The ID of the grantor or the revoker of the privilege or authority.
Grantee	VARCHAR(128)	<p>Grantee ID for which a privilege or authority was granted or revoked.</p> <p>Represents a trusted context object when the audit event is any of:</p> <ul style="list-style-type: none"> <li>• ADD_DEFAULT_ROLE</li> <li>• DROP_DEFAULT_ROLE</li> <li>• ALTER_DEFAULT_ROLE</li> <li>• ADD_USER, DROP_USER</li> <li>• ALTER_USER_ADD_ROLE</li> <li>• ALTER_USER_DROP_ROLE</li> <li>• ALTER_USER_AUTHENTICATION</li> </ul>

Table 47. Audit Record Layout for SECMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Grantee Type	VARCHAR(32)	Type of the grantee that was granted to or revoked from. Possible values include: USER, GROUP, ROLE, AMBIGUOUS, or is TRUSTED_CONTEXT when the audit event is any of: <ul style="list-style-type: none"> <li>• ADD_DEFAULT_ROLE</li> <li>• DROP_DEFAULT_ROLE</li> <li>• ALTER_DEFAULT_ROLE</li> <li>• ADD_USER</li> <li>• DROP_USER</li> <li>• ALTER_USER_ADD_ROLE</li> <li>• ALTER_USER_DROP_ROLE</li> <li>• ALTER_USER_AUTHENTICATION</li> </ul>
Privilege or Authority	CHAR(34)	Indicates the type of privilege or authority granted or revoked. Possible values include: those shown in the topic titled "List of possible SECMAINT privileges or authorities". <p>The value is ROLE MEMBERSHIP when the audit event is any of the following:</p> <ul style="list-style-type: none"> <li>• ADD_DEFAULT_ROLE, DROP_DEFAULT_ROLE</li> <li>• ALTER_DEFAULT_ROLE</li> <li>• ADD_USER</li> <li>• DROP_USER</li> <li>• ALTER_USER_ADD_ROLE</li> <li>• ALTER_USER_DROP_ROLE</li> <li>• ALTER_USER_AUTHENTICATION</li> </ul>
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Access Type	VARCHAR(32)	The access type for which a security label is granted. <p>Possible values:</p> <ul style="list-style-type: none"> <li>• READ</li> <li>• WRITE</li> <li>• ALL</li> </ul> <p>The access type for which a security policy is altered. Possible values:</p> <ul style="list-style-type: none"> <li>• USE GROUP AUTHORIZATIONS</li> <li>• IGNORE GROUP AUTHORIZATIONS</li> <li>• USE ROLE AUTHORIZATIONS</li> <li>• IGNORE ROLE AUTHORIZATIONS</li> <li>• OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL</li> <li>• RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL</li> </ul>
Assumable Authid	VARCHAR(128)	When the privilege granted is a SETSESSIONUSER privilege this is the authorization ID that the grantee is allowed to set as the session user.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.



Table 47. Audit Record Layout for SECMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Grantor Type	VARCHAR(32)	Type of the grantor. Possible values include: USER.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context User	VARCHAR(128)	Identifies a trusted context user when the audit event is ADD_USER or DROP_USER.
Trusted Context User Authentication	INTEGER	Specifies the authentication setting for a trusted context user when the audit event is ADD_USER, DROP_USER or ALTER_USER_AUTHENTICATION 1 : Authentication is required 0 : Authentication is not required
Trusted Context Name	VARCHAR(255)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	CHAR(1)	Possible values are:  " - NONE '1' - IMPLICIT_TRUSTED_CONNECTION '2' - EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.
Associated Object Name	VARCHAR(128)	Name of the object for which an association exists. The meaning of the association depends on the Object Type for the event. If the Object Type is PERMISSION or MASK, then the Associated Object is the table on which that permission or mask has been created.
Associated Object Schema	VARCHAR(128)	Name of the object schema for which an association exists. The meaning of the association depends on the Object Type of the event.
Associated Object Type	VARCHAR(128)	The type of the object for which an association exists. The meaning of the association depends on the Object Type of the event.
Associated Subobject Type	VARCHAR(128)	The type of the subobject for which an association exists. The meaning of the association depends on the Object Type of the event. If the Object Type is MASK and the Associated Object type is TABLE, then the associated subobject is the column of the table on which the mask has been created.
Associated Subobject Name	VARCHAR(128)	Name of the subobject for which an association exists. The meaning of the association depends on the Object Type of the event.
Alter Action	VARCHAR(32)	Specific Alter Action.  Possible values include: <ul style="list-style-type: none"> <li>• SECURE</li> <li>• UNSECURE</li> <li>• ENABLE</li> <li>• DISABLE</li> <li>• ACTIVATE_ROW_ACCESS_CONTROL</li> <li>• ACTIVATE_COLUMN_ACCESS_CONTROL</li> <li>• ACTIVATE_ROW_COLUMN_ACCESS_CONTROL</li> </ul>



0x00000000000000000000000000000000400 **Table SELECT with GRANT**  
Privilege granted or revoked on or from a select on a table with granting of privileges allowed.

0x00000000000000000000000000000000800 **Table UPDATE**  
Privilege granted or revoked on or from an update on a table or view.

0x000000000000000000000000000000001000 **Table UPDATE with GRANT**  
Privilege granted or revoked on or from an update on a table or view with granting of privileges allowed.

0x000000000000000000000000000000002000 **Table REFERENCE**  
Privilege granted or revoked on or from a reference on a table.

0x000000000000000000000000000000004000 **Table REFERENCE with GRANT**  
Privilege granted or revoked on or from a reference on a table with granting of privileges allowed.

0x0000000000000000000000000000000020000 **CREATEIN Schema**  
CREATEIN privilege granted or revoked on or from a schema.

0x0000000000000000000000000000000040000 **CREATEIN Schema with GRANT**  
CREATEIN privilege granted or revoked on or from a schema with granting of privileges allowed.

0x0000000000000000000000000000000080000 **DROPIN Schema**  
DROPIN privilege granted or revoked on or from a schema.

0x00000000000000000000000000000000100000 **DROPIN Schema with GRANT**  
DROPIN privilege granted or revoked on or from a schema with granting of privileges allowed.

0x00000000000000000000000000000000200000 **ALTERIN Schema**  
ALTERIN privilege granted or revoked on or from a schema.

0x00000000000000000000000000000000400000 **ALTERIN Schema with GRANT**  
ALTERIN privilege granted or revoked on or from a schema with granting of privileges allowed.

0x00000000000000000000000000000000800000 **DBADM Authority**  
DBADM authority granted or revoked.

0x000000000000000000000000000000001000000 **CREATETAB Authority**  
Createtab authority granted or revoked.

0x000000000000000000000000000000002000000 **BINDADD Authority**  
Bindadd authority granted or revoked.

0x000000000000000000000000000000004000000 **CONNECT Authority**  
CONNECT authority granted or revoked.

0x000000000000000000000000000000008000000 **Create not fenced Authority**  
Create not fenced authority granted or revoked.

0x00000000000000000000000000000000100000000 **Implicit Schema Authority**  
Implicit schema authority granted or revoked.

0x00000000000000000000000000000000200000000 **Server PASSTHRU**  
Privilege granted or revoked to use the pass-through facility with this server (federated database data source).

0x00000000000000000000000000000000400000000 **ESTABLISH TRUSTED CONNECTION**  
Trusted connection was created

0x00000000000000000000000000000000	<b>Table Space USE</b>	Privilege granted or revoked to create a table in a table space.
0x000000000000000000000000000000002000000000	<b>Table Space USE with GRANT</b>	Privilege granted or revoked to create a table in a table space with granting of privileges allowed.
0x000000000000000000000000000000004000000000	<b>Column UPDATE</b>	Privilege granted or revoked on or from an update on one or more specific columns of a table.
0x000000000000000000000000000000008000000000	<b>Column UPDATE with GRANT</b>	Privilege granted or revoked on or from an update on one or more specific columns of a table with granting of privileges allowed.
0x0000000000000000000000000000000010000000000	<b>Column REFERENCE</b>	Privilege granted or revoked on or from a reference on one or more specific columns of a table.
0x0000000000000000000000000000000020000000000	<b>Column REFERENCE with GRANT</b>	Privilege granted or revoked on or from a reference on one or more specific columns of a table with granting of privileges allowed.
0x0000000000000000000000000000000040000000000	<b>LOAD Authority</b>	LOAD authority granted or revoked.
0x0000000000000000000000000000000080000000000	<b>Package BIND</b>	BIND privilege granted or revoked on or from a package.
0x00000000000000000000000000000000100000000000	<b>Package BIND with GRANT</b>	BIND privilege granted or revoked on or from a package with granting of privileges allowed.
0x00000000000000000000000000000000200000000000	<b>EXECUTE</b>	EXECUTE privilege granted or revoked on or from a package or a routine.
0x00000000000000000000000000000000400000000000	<b>EXECUTE with GRANT</b>	EXECUTE privilege granted or revoked on or from a package or a routine with granting of privileges allowed.
0x00000000000000000000000000000000800000000000	<b>EXECUTE IN SCHEMA</b>	EXECUTE privilege granted or revoked for all routines in a schema.
0x000000000000000000000000000000001000000000000	<b>EXECUTE IN SCHEMA with GRANT</b>	EXECUTE privilege granted or revoked for all routines in a schema with granting of privileges allowed.
0x000000000000000000000000000000002000000000000	<b>EXECUTE IN TYPE</b>	EXECUTE privilege granted or revoked for all routines in a type.
0x000000000000000000000000000000004000000000000	<b>EXECUTE IN TYPE with GRANT</b>	EXECUTE privilege granted or revoked for all routines in a type with granting of privileges allowed.
0x000000000000000000000000000000008000000000000	<b>CREATE EXTERNAL ROUTINE</b>	CREATE EXTERNAL ROUTINE privilege granted or revoked.
0x0000000000000000000000000000000010000000000000	<b>QUIESCE_CONNECT</b>	QUIESCE_CONNECT privilege granted or revoked.
0x0000000000000000000000000000000040000000000000	<b>SECADM Authority</b>	SECADM authority granted or revoked
0x0000000000000000000000000000000080000000000000	<b>USAGE Authority</b>	USAGE privilege granted or revoked on or from a sequence



```

event status=0;
userid=boss;authid=BOSS;
application id=*LOCAL.boss.980624155404;
application name=db2audit;

```

*Table 48. Audit Record Layout for SYSADMIN Events*

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are:  SYSADMIN
Audit Event	VARCHAR(32)	Specific Audit Event.  For a list of possible values, refer to the section for the SYSADMIN category in “Audit events” on page 354.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where  Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Member number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Member number of the coordinator member.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.

Table 48. Audit Record Layout for SYSADMIN Events (continued)

NAME	FORMAT	DESCRIPTION
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(255)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	CHAR(1)	Possible values are:  " - NONE '1' - IMPLICIT_TRUSTED_CONNECTION '2' - EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.
Event Details	VARCHAR(2048)	Information that is specific to the audit event.

## Audit record layout for VALIDATE events

The format of the audit record for VALIDATE events is shown in the following table.

Sample audit record:

```
timestamp=2007-05-07-10.30.51.585626;
category=VALIDATE;
audit event=AUTHENTICATION;
event correlator=1;
event status=0;
userid=newton;
authid=NEWTON;
execution id=gstager;
application id=*LOCAL.gstager.070507143051;
application name=db2bp;
auth type=SERVER;
plugin name=IBMOSauthserver;
```

Table 49. Audit Record Layout for VALIDATE Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are:  VALIDATE
Audit Event	VARCHAR(32)	Specific Audit Event.  Possible values include: GET_GROUPS, GET_USERID, AUTHENTICATE_PASSWORD, VALIDATE_USER, AUTHENTICATION and GET_USERMAPPING_FROM_PLUGIN.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where  Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Execution ID	VARCHAR(1024)	Execution ID in use at the time of the audit event.
Origin Node Number	SMALLINT	Member number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Member number of the coordinator member.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Authentication Type	VARCHAR(32)	Authentication type at the time of the audit event.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Plug-in Name	VARCHAR(32)	The name of the plug-in in use at the time the audit event occurred.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.



Table 49. Audit Record Layout for VALIDATE Events (continued)

NAME	FORMAT	DESCRIPTION
Trusted Context Name	VARCHAR(255)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	CHAR(1)	Possible values are:  " - NONE '1' - IMPLICIT_TRUSTED_CONNECTION '2' - EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The name of the role inherited through the trusted context.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.

## Audit record layout for CONTEXT events

The following table shows the audit record layout for CONTEXT events.

Sample audit record:

```
timestamp=1998-06-24-08.42.41.476840;
category=CONTEXT;
audit_event=EXECUTE_IMMEDIATE;
event_correlator=3;
database=F00;
userid=boss;
authid=BOSS;
application_id=*LOCAL.newton.980624124210;
application_name=testapp;
package_schema=NULLID;
package_name=SQLC28A1;
package_section=203;
text=create table audit(c1 char(10), c2 integer);
```

Table 50. Audit Record Layout for CONTEXT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are:  CONTEXT
Audit Event	VARCHAR(32)	Specific Audit Event.  For a list of possible values, refer to the section for the CONTEXT category in "Audit events" on page 354.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.  When the audit event is SWITCH_USER, this field represents the user ID that is switched to.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.  When the audit event is SWITCH_USER, this field represents the authorization ID that is switched to.

Table 50. Audit Record Layout for CONTEXT Events (continued)

NAME	FORMAT	DESCRIPTION
Origin Node Number	SMALLINT	Member number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Member number of the coordinator member.
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Statement Text	CLOB(8M)	Text of the SQL or XQuery statement, if applicable. Null if no SQL or XQuery statement text is available.
Package Version	VARCHAR(64)	Version of the package in use at the time the audit event occurred.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred.
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred.
Trusted Context Name	VARCHAR(255)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	CHAR(1)	Possible values are:  " - NONE '1' - IMPLICIT_TRUSTED_CONNECTION '2' - EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.

## Audit record layout for EXECUTE events

The following table describes all of the fields that are audited as part of the EXECUTE category.

Sample audit record:

**Note:** Unlike other audit categories, the EXECUTE category, when the audit log is viewed in a table format, can show multiple rows describing one event. The first record describes the main event, and its event column contains the key word STATEMENT. The remaining rows describe the parameter markers or host variables, one row per parameter, and their event column contains the key word

DATA. When the audit log is viewed in report format, there is one record, but it has multiple entries for the Statement Value. The DATA key word is only be present in table format.

```
timestamp=2006-04-10-13.20.51.029203;
category=EXECUTE;
audit event=STATEMENT;
event correlator=1;
event status=0;
database=SAMPLE;
userid=smith;
authid=SMITH;
session authid=SMITH;
application id=*LOCAL.prodriq.060410172044;
application name=myapp;
package schema=NULLID;
package name=SQLC2F0A;
package section=201;
uow id=2;
activity id=3;
statement invocation id=0;
statement nesting level=0;
statement text=SELECT * FROM DEPARTMENT WHERE DEPTNO = ? AND DEPTNAME = ?;
statement isolation level=CS;
compilation environment=
 isolation level=CS
 query optimization=5
 degree=1
 sqlrules=DB2
 refresh age=+000000000000000.000000
 schema=SMITH
 maintained table type=SYSTEM
 resolution timestamp=2006-04-10-13.20.51.000000
 federated asynchrony=0;
value index=0;
value type=CHAR;
value data=C01;
value index=1;
value type=VARCHAR;
value extended indicator=-1;
value index=INFORMATION CENTER;
local_start_time=2006-04-10-13.20.51.021507
```

*Table 51. Audit Record Layout for EXECUTE Events*

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event
Category	CHAR(8)	Category of audit event. Possible values are: EXECUTE
Audit Event	VARCHAR(32)	Specific Audit Event.  For a list of possible values, refer to the section for the EXECUTE category in “Audit events” on page 354.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.

Table 51. Audit Record Layout for EXECUTE Events (continued)

NAME	FORMAT	DESCRIPTION
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	The Statement Authorization ID at time of audit event.
Session Authorization ID	VARCHAR(128)	The Session Authorization ID at the time of the audit event.
Origin Node Number	SMALLINT	Member number at which the audit event occurred
Coordinator Node Number	SMALLINT	Member number of the coordinator member
Application ID	VARCHAR(255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR(1024)	Application name in use at the time the audit event occurred.
Client User ID	VARCHAR(255)	The value of the CURRENT CLIENT USERID special register at the time the audit event occurred
Client Accounting String	VARCHAR(255)	The value of the CURRENT CLIENT_ACCTNG special register at the time the audit event occurred
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred
Client Application Name	VARCHAR(255)	The value of the CURRENT CLIENT_APPLNAME special register at the time the audit event occurred
Trusted Context Name	VARCHAR(255)	The name of the trusted context associated with the trusted connection.
Connection Trust type	CHAR(1)	Possible values are:  " - NONE '1' - IMPLICIT_TRUSTED_CONNECTION '2' - EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.

Table 51. Audit Record Layout for EXECUTE Events (continued)

NAME	FORMAT	DESCRIPTION
Package Schema	VARCHAR(128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR(128)	Name of package in use at the time the audit event occurred.
Package Section	SMALLINT	Section number in package being used at the time the audit event occurred.
Package Version	VARCHAR(164)	Version of the package in use at the time the audit event occurred.
Local Transaction ID	VARCHAR(10) FOR BIT DATA	The local transaction ID in use at the time the audit event occurred. This is the SQLU_TID structure that is part of the transaction logs.
Global Transaction ID	VARCHAR(30) FOR BIT DATA	The global transaction ID in use at the time the audit event occurred. This is the data field in the SQLP_GXID structure that is part of the transaction logs
UOW ID	BIGINT	The unit of work identifier in which an activity originates. This value is unique within an application ID for each unit of work.
Activity ID	BIGINT	The unique activity ID within the unit of work.
Statement Invocation ID	BIGINT	An identifier that distinguishes one invocation of a routine from others at the same nesting level within a unit of work. It is unique within a unit of work for a specific nesting level.
Statement Nesting Level	BIGINT	The level of nesting or recursion in effect when the statement was being run; each level of nesting corresponds to nested or recursive invocation of a stored procedure or user-defined function (UDF).

Table 51. Audit Record Layout for EXECUTE Events (continued)

NAME	FORMAT	DESCRIPTION
Activity Type	VARCHAR(32)	<p>The type of activity.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>• READ_DML</li> <li>• WRITE_DML</li> <li>• DDL</li> <li>• CALL</li> <li>• OTHER</li> </ul>
Statement Text	CLOB(8M)	Text of the SQL or XQuery statement, if applicable.
Statement Isolation Level	CHAR(8)	<p>The isolation value in effect for the statement while it was being run.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>• NONE (no isolation specified)</li> <li>• UR (uncommitted read)</li> <li>• CS (cursor stability)</li> <li>• RS (read stability)</li> <li>• RR (repeatable read)</li> </ul>
Compilation Environment Description	BLOB(8K)	The compilation environment used when compiling the SQL statement. You can provide this element as input to the COMPILATION_ENV table function, or to the SET COMPILATION ENVIRONMENT SQL statement
Rows Modified	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 inlined 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. This value is equivalent to the sqlerrd(5) field of the SQLCA.</p>

Table 51. Audit Record Layout for EXECUTE Events (continued)

NAME	FORMAT	DESCRIPTION
Rows Returned	BIGINT	Contains the total number of rows returned by the statement.
Savepoint ID	BIGINT	The Savepoint ID in effect for the statement while it is being run. If the Audit Event is SAVEPOINT, RELEASE_SAVEPOINT or ROLLBACK_SAVEPOINT, then the Savepoint ID is the save point that is being set, released, or rolled back to.
Statement Value Index	INTEGER	The position of the input parameter marker or host variable used in the SQL statement.
Statement Value Type	CHAR(16)	A string representation of the type of a data value associated with the SQL statement. INTEGER or CHAR are examples of possible values.
Statement Value Data	CLOB(128K)	A string representation of a data value to the SQL statement. LOB, LONG, XML, and structured type parameters are not present. Date, time, and timestamp fields are recorded in ISO format.
Statement Value Extended Indicator	INTEGER	The value of the extended indicator specified for this statement value. The possible values are: <ul style="list-style-type: none"> <li>• 0 if the statement value was specified as assigned by the indicator value,</li> <li>• -1 if NULL was specified by the indicator value,</li> <li>• -5 if DEFAULT was specified by the indicator value,</li> <li>• -7 if UNASSIGNED was specified by the indicator value.</li> </ul>

Table 51. Audit Record Layout for EXECUTE Events (continued)

NAME	FORMAT	DESCRIPTION
Local Start Time	CHAR(26)	The time that this activity began working on the partition. This field can be an empty string when the activity does not require a package, that is, for CONNECT, CONNECT RESET, COMMIT, and ROLLBACK, as an example. The value is logged in local time.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.

## Audit events

For each audit category, certain types of events can create audit records.

### Events for the AUDIT category

- ALTER\_AUDIT\_POLICY
- ARCHIVE
- AUDIT\_REMOVE
- AUDIT\_REPLACE
- AUDIT\_USING
- CONFIGURE
- CREATE\_AUDIT\_POLICY
- DB2AUD
- DROP\_AUDIT\_POLICY
- EXTRACT
- FLUSH
- LIST\_LOGS
- PRUNE (not generated in Version 9.5, and later).
- START
- STOP
- UPDATE\_DBM\_CFG

### Events for the CHECKING category

- CHECKING\_FUNCTION
- CHECKING\_MEMBERSHIP\_IN\_ROLES
- CHECKING\_OBJECT
- CHECKING\_TRANSFER

### Events for the CONTEXT category

- ADD\_NODE
- ATTACH



- BACKUP\_DB
- BIND
- CLOSE\_CONTAINER\_QUERY
- CLOSE\_CURSOR
- CLOSE\_HISTORY\_FILE
- CLOSE\_TABLESPACE\_QUERY
- COMMIT
- CONNECT
- CONNECT\_RESET
- CREATE\_DATABASE
- DARI\_START
- DARI\_STOP
- DBM\_CFG\_OPERATION
- DESCRIBE
- DESCRIBE\_DATABASE
- DETACH
- DISCOVER
- DROP\_DATABASE
- ENABLE\_MULTIPAGE
- ESTIMATE\_SNAPSHOT\_SIZE
- EXECUTE
- EXECUTE\_IMMEDIATE
- EXTERNAL\_CANCEL
- FETCH\_CONTAINER\_QUERY
- FETCH\_CURSOR
- FETCH\_HISTORY\_FILE
- FETCH\_TABLESPACE
- FORCE\_APPLICATION
- GET\_DB\_CFG
- GET\_DFLT\_CFG
- GET\_SNAPSHOT
- GET\_TABLESPACE\_STATISTIC
- IMPLICIT\_REBIND
- LOAD\_MSG\_FILE
- LOAD\_TABLE
- OPEN\_CONTAINER\_QUERY
- OPEN\_CURSOR
- OPEN\_HISTORY\_FILE
- OPEN\_TABLESPACE\_QUERY
- PREPARE
- PRUNE\_RECOVERY\_HISTORY
- QUIESCE\_TABLESPACE
- READ\_ASYNC\_LOG\_RECORD
- REBIND
- REDISTRIBUTE

- REORG
- REQUEST\_ROLLBACK
- RESET\_DB\_CFG
- RESET\_MONITOR
- RESTORE\_DB
- ROLLBACK
- ROLLFORWARD\_DB
- RUNSTATS
- SET\_APPL\_PRIORITY
- SET\_MONITOR
- SET\_RUNTIME\_DEGREE
- SET\_TABLESPACE\_CONTAINERS
- SINGLE\_TABLESPACE\_QUERY
- SWITCH\_USER
- UNLOAD\_TABLE
- UNQUIESCE\_TABLESPACE
- UPDATE\_AUDIT
- UPDATE\_DBM\_CFG
- UPDATE\_RECOVERY\_HISTORY

### **Events for the EXECUTE category**

- COMMIT Execution of a COMMIT statement
- CONNECT Establishment of a database connection
- CONNECT RESET Termination of a database connection
- DATA A host variable or parameter marker data values for the statement  
This event is repeated for each host variable or parameter marker that is part of the statement. It is only present in a delimited extract of an audit log.
- GLOBAL COMMIT Execution of a COMMIT within a global transaction
- GLOBAL ROLLBACK Execution of a ROLLBACK within a global transaction
- RELEASE SAVEPOINT Execution of a RELEASE SAVEPOINT statement
- ROLLBACK Execution of a ROLLBACK statement
- SAVEPOINT Execution of a SAVEPOINT statement
- STATEMENT Execution of an SQL statement
- SWITCH USER Switching of a user within a trusted connection

### **Events for the OBJMAINT category**

- ALTER\_OBJECT (generated when altering protected tables and when altering modules)
- CREATE\_OBJECT
- DROP\_OBJECT
- RENAME\_OBJECT

### **Events for the SECMAINT category**

- ADD\_DEFAULT\_ROLE
- ADD\_USER
- ALTER\_DEFAULT\_ROLE

- ALTER\_OBJECT
- ALTER\_SECURITY\_POLICY
- ALTER\_USER\_ADD\_ROLE
- ALTER\_USER\_AUTHENTICATION
- ALTER\_USER\_DROP\_ROLE
- CREATE\_OBJECT
- DROP\_DEFAULT\_ROLE
- DROP\_OBJECT
- DROP\_USER
- GRANT
- IMPLICIT\_GRANT
- IMPLICIT\_REVOKE
- RENAME\_OBJECT
- REVOKE
- SET\_SESSION\_USER
- TRANSFER\_OWNERSHIP
- UPDATE\_DBM\_CFG

### **Events for the SYSADMIN category**

- ACTIVATE\_DB
- ADD\_NODE
- ALTER\_BUFFERPOOL
- ALTER\_DATABASE
- ALTER\_NODEGROUP
- ALTER\_TABLESPACE
- ATTACH\_DEBUGGER
- BACKUP\_DB
- CATALOG\_DB
- CATALOG\_DCS\_DB
- CATALOG\_NODE
- CHANGE\_DB\_COMMENT
- CLOSE\_CONTAINER\_QUERY
- CLOSE\_TABLESPACE\_QUERY
- COMMIT\_DSF\_CFS
- COMMIT\_DSF\_CM
- COMMIT\_DSF\_INSTANCE
- CREATE\_BUFFERPOOL
- CREATE\_DATABASE
- CREATE\_DB\_AT\_NODE
- CREATE\_EVENT\_MONITOR
- CREATE\_INSTANCE
- CREATE\_NODEGROUP
- CREATE\_TABLESPACE
- DB2AUD
- DB2AUDIT

- DB2REMOT
- DB2SET
- DB2TRC
- DEACTIVATE\_DB
- DELETE\_INSTANCE
- DESCRIBE\_DATABASE
- DROP\_BUFFERPOOL
- DROP\_DATABASE
- DROP\_EVENT\_MONITOR
- DROP\_NODEGROUP
- DROP\_NODE\_VERIFY
- DROP\_TABLESPACE
- ENABLE\_MULTIPAGE
- ESTIMATE\_SNAPSHOT\_SIZE
- FETCH\_CONTAINER\_QUERY
- FETCH\_TABLESPACE
- FORCE\_APPLICATION
- GET\_SNAPSHOT
- GET\_TABLESPACE\_STATISTIC
- GRANT\_DBADM (V97:no longer generated)
- GRANT\_DB\_AUTH (V97:no longer generated)
- KILLDBM
- LIST\_DRDA\_INDOUBT\_TRANSACTIONS
- LOAD\_TABLE
- MAINTENANCE\_DSF\_MODE
- MERGE\_DBM\_CONFIG\_FILE
- MIGRATE\_DB
- MIGRATE\_DB\_DIR
- MIGRATE\_SYSTEM\_DIRECTORY
- OPEN\_CONTAINER\_QUERY
- OPEN\_TABLESPACE\_QUERY
- PRUNE\_RECOVERY\_HISTORY
- QUIESCE\_TABLESPACE
- READ\_ASYNC\_LOG\_RECORD
- REDISTRIBUTE\_NODEGROUP
- RENAME\_TABLESPACE
- RESET\_ADMIN\_CFG
- RESET\_DBM\_CFG
- RESET\_DB\_CFG
- RESET\_MONITOR
- RESTORE\_DB
- REVOKE\_DBADM (V97:no longer generated)
- REVOKE\_DB\_AUTH (V97:no longer generated)
- ROLLFORWARD\_DB
- SET\_APPL\_PRIORITY

- SET\_EVENT\_MONITOR\_STATE
- SET\_RUNTIME\_DEGREE
- SET\_TABLESPACE\_CONTAINERS
- SINGLE\_TABLESPACE\_QUERY
- START\_CF
- STOP\_CF
- START\_DB2
- STOP\_DB2
- START\_DSF\_INSTANCE
- STOP\_DSF\_INSTANCE
- UNCATALOG\_DB
- UNCATALOG\_DCS\_DB
- UNCATALOG\_NODE
- UNLOAD\_TABLE
- UPDATE\_ADMIN\_CFG
- UPDATE\_CLI\_CONFIGURATION
- UPDATE\_DSF\_MEMBER\_OR\_CF
- UPDATE\_DB\_VERSION
- UPDATE\_DBM\_CFG
- UPDATE\_DB\_CFG
- SET\_MONITOR
- UPDATE\_RECOVERY\_HISTORY

### **Events for the VALIDATE category**

- AUTHENTICATE
- CHECK\_GROUP\_MEMBERSHIP (not generated in Version 9.5, and later)
- GET\_USERMAPPING\_FROM\_PLUGIN
- GET\_GROUPS (not generated in Version 9.5, and later)
- GET\_USERID (not generated in Version 9.5, and later)



---

## Chapter 12. Working with operating system security

Operating systems provide security features that you can use to support security for your database installation.

---

### Db2 and Windows security

A Windows domain is an arrangement of client and server computers referenced by a specific and unique name; and, that share a single user accounts database called the Security Access Manager (SAM). One of the computers in the domain is the domain controller. The domain controller manages all aspects of user-domain interactions.

The domain controller uses the information in the domain user accounts database to authenticate users logging onto domain accounts. For each domain, one domain controller is the primary domain controller (PDC). Within the domain, there may also be backup domain controllers (BDC) which authenticate user accounts when there is no primary domain controller or the primary domain controller is not available. Backup domain controllers hold a copy of the Windows Security Account Manager (SAM) database which is regularly synchronized against the master copy on the PDC.

User accounts, user IDs, and passwords only need to be defined at the primary domain controller to be able to access domain resources.

**Note:** Two-part user IDs are supported by the CONNECT statement and the ATTACH command. The qualifier of the SAM-compatible user ID is a name of the style 'Domain\User' which has a maximum length of 15 characters.

During the setup procedure when a Windows server is installed, you may select to create:

- A primary domain controller in a new domain
- A backup domain controller in a known domain
- A stand-alone server in a known domain.

Selecting “controller” in a new domain makes that server the primary domain controller.

The user may log on to the local machine, or when the machine is installed in a Windows Domain, the user may log on to the Domain. To authenticate the user, Db2 checks the local machine first, then the Domain Controller for the current Domain, and finally any Trusted Domains known to the Domain Controller.

To illustrate how this works, suppose that the Db2 instance requires Server authentication. The configuration is as follows:

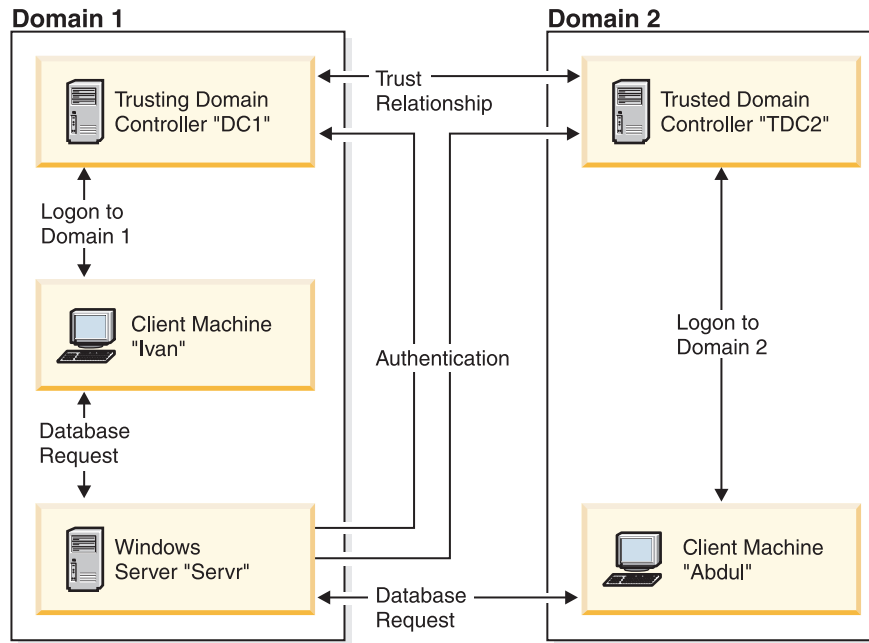


Figure 7. Authentication Using Windows Domains

Each machine has a security database, Security Access Management (SAM). DC1 is the domain controller, in which the client machine, Ivan, and the Db2 server, Servr, are enrolled. TDC2 is a trusted domain for DC1 and the client machine, Abdul, is a member of TDC2's domain.

## Authentication scenarios

### A scenario with server authentication (Windows)

The following example demonstrates authentication of a user by a server.

1. Abdul logs on to the TDC2 domain (that is, he is known in the TDC2 SAM database).
2. Abdul then connects to a Db2 database that is cataloged to reside on SRV3:  

```
db2 connect to remotedb user Abdul using fredpw
```
3. SRV3 determines where Abdul is known. The API that is used to find this information first searches the local machine (SRV3) and then the domain controller (DC1) before trying any trusted domains. Username Abdul is found on TDC2. This search order requires a single namespace for users and groups.
4. SRV3 then:
  - a. Validates the username and password with TDC2.
  - b. Finds out whether Abdul is an administrator by asking TDC2.
  - c. Enumerates all Abdul's groups by asking TDC2.

### A scenario with client authentication and a Windows client machine

The following example demonstrates authentication of a user by a client computer.

1. Dale, the administrator, logs on to SRV3 and changes the authentication for the database instance to Client:
 

```
db2 update dbm cfg using authentication client
db2stop
db2start
```



2. Ivan, at a Windows client machine, logs on to the DC1 domain (that is, he is known in the DC1 SAM database).
3. Ivan then connects to a Db2 database that is cataloged to reside on SRV3:  
DB2 CONNECT to remotedb user Ivan using johnpw
4. Ivan's machine validates the username and password. The API used to find this information first searches the local machine (Ivan) and then the domain controller (DC1) before trying any trusted domains. Username Ivan is found on DC1.
5. Ivan's machine then validates the username and password with DC1.
6. SRV3 then:
  - a. Determines where Ivan is known.
  - b. Finds out whether Ivan is an administrator by asking DC1.
  - c. Enumerates all Ivan's groups by asking DC1.

**Note:** Before attempting to connect to the Db2 database, ensure that Db2 Security Service has been started. The Security Service is installed as part of the Windows installation. Db2 is then installed and “registered” as a Windows service however, it is not started automatically. To start the Db2 Security Service, enter the NET START DB2NTSECSEVER command.

## Support for global groups (Windows)

The Db2 database system supports global groups.

To use global groups, you must include global groups inside a local group. When the Db2 database manager enumerates all the groups that a person is a member of, it also lists the local groups that the user is a member of indirectly (by the virtue of being in a global group that is itself a member of one or more local groups).

Global groups are used in two possible situations:

- Included inside a local group. Permission must be granted to this local group.
- Included on a domain controller. Permission must be granted to the global group.

## User authentication and group information with DB2 on Windows

### User name and group name restrictions (Windows)

There are a few limitations that are specific to the Windows environment. Be aware that general Db2 object naming rules also apply.

- User names under Windows are not case sensitive; however, passwords are case sensitive.
- User names and group names can be a combination of upper- and lowercase characters. However, they are usually converted to uppercase when used within the Db2 database. For example, if you connect to the database and create the table schema1.table1, this table is stored as SCHEMA1.TABLE1 within the database. (If you want to use lowercase object names, issue commands from the command line processor, enclosing the object names in quotation marks, or use third-party ODBC front-end tools.)
- The Db2 database manager supports a single namespace. That is, when running in a trusted domains environment, you should not have a user account of the same name that exists in multiple domains, or that exists in the local SAM of the server machine and in another domain.

- A user name should not be the same name as a group name.
- A local group should not have the same name as a domain level group.

## **Groups and user authentication on Windows**

Users are defined on Windows by creating user accounts using the Windows administration tool called the “User Manager”. An account containing other accounts, also called members, is a group.

Groups give Windows administrators the ability to grant rights and permissions to the users within the group at the same time, without having to maintain each user individually. Groups, like user accounts, are defined and maintained in the Security Access Manager (SAM) database.

There are two types of groups:

- Local groups. A local group can include user accounts created in the local accounts database. If the local group is on a machine that is part of a domain, the local group can also contain domain accounts and groups from the Windows domain. If the local group is created on a workstation, it is specific to that workstation.
- Global groups. A global group exists only on a domain controller and contains user accounts from the domain's SAM database. That is, a global group can only contain user accounts from the domain on which it is created; it cannot contain any other groups as members. A global group can be used in servers and workstations of its own domain, and in trusting domains.

## **Trust relationships between domains on Windows**

Trust relationships are an administration and communication link between two domains. A trust relationship between two domains enables user accounts and global groups to be used in a domain other than the domain where the accounts are defined.

Account information is shared to validate the rights and permissions of user accounts and global groups residing in the trusted domain without being authenticated. Trust relationships simplify user administration by combining two or more domains into a single administrative unit.

There are two domains in a trust relationship:

- The trusting domain. This domain trusts another domain to authenticate users for them.
- The trusted domain. This domain authenticates users on behalf of (in trust for) another domain.

Trust relationships are not transitive. This means that explicit trust relationships need to be established in each direction between domains. For example, the trusting domain may not necessarily be a trusted domain.

## **Authentication with groups and domain security (Windows)**

The Db2 database system allows you to specify either a local group or a global group when granting privileges or defining authority levels.

### **About this task**

A user is determined to be a member of a group if the user's account is defined explicitly in the local or global group, or implicitly by being a member of a global group defined to be a member of a local group.

The Db2 database manager supports the following types of groups:

- Local groups
- Global groups
- Global groups as members of local groups.

The Db2 database manager enumerates the local and global groups of which the user is a member, using the security database where the user was found. The Db2 database system provides an override that forces group enumeration to occur on the local Windows server where the Db2 database is installed, regardless of where the user account was found. This override can be achieved using the following commands:

- For global settings:

```
db2set -g DB2_GRP_LOOKUP=local
```

- For instance settings:

```
db2set -i instance_name DB2_GRP_LOOKUP=local
```

After issuing this command, you must stop and start the Db2 database instance for the change to take effect. Then create local groups and include domain accounts or global groups in the local group.

To view all Db2 profile registry variables that are set, type

```
db2set -all
```

If the **DB2\_GRP\_LOOKUP** profile registry variable is set to local, then the Db2 database manager tries to enumerate the user's groups on the local machine only. If the user is not defined as a member of a local group, or of a global group nested in a local group, then group enumeration fails. The Db2 database manager does **not** try to enumerate the user's groups on another machine in the domain or on the domain controllers.

If the Db2 database manager is running on a machine that is a primary or backup domain controller in the resource domain, it is able to locate any domain controller in any trusted domain. This occurs because the names of the domains of backup domain controllers in trusted domains are only known if you are a domain controller.

## Using an access token to acquire users' group information (Windows)

An access token is an object that describes the security context of a process or thread. The information in an access token includes the identity and privileges of the user account associated with the process or thread.

When you log on, the system verifies your password by comparing it with information stored in a security database. If the password is authenticated, the system produces an access token. Every process run on your behalf uses a copy of this access token.

An access token can also be acquired based on cached credentials. After you have been authenticated to the system, your credentials are cached by the operating system. The access token of the last logon can be referenced in the cache when it is not possible to contact the domain controller.

The access token includes information about all of the groups you belong to: local groups and various domain groups (global groups, domain local groups, and universal groups).

**Note:** Group lookup using client authentication is not supported using a remote connection even though access token support is enabled.

To enable access token support, you must use the **db2set** command to update the **DB2\_GRP\_LOOKUP** registry variable. **DB2\_GRP\_LOOKUP** can have up to two parameters, separated by a comma:

- The first parameter is for conventional group lookup and can take the values: " ", "LOCAL", or "DOMAIN".
- The second parameter is for token style group lookup and can take the values: "TOKEN", "TOKENDOMAIN", or "TOKENLOCAL".

If the second parameter (TOKEN, TOKENDOMAIN, or TOKENLOCAL) is specified, it takes precedence over conventional group enumeration. If token group enumeration fails, conventional group lookup occurs, if the first parameter of **DB2\_GRP\_LOOKUP** was specified.

The meaning of the values TOKEN, TOKENDOMAIN, and TOKENLOCAL are as follows:

- **TOKENLOCAL**  
The token is used to enumerate groups at the local machine (this is equivalent to conventional "LOCAL" group lookup).
- **TOKENDOMAIN**  
The token is used to enumerate groups at the location where the user is defined (at local machine for a local user and at the domain for a domain user). This is equivalent to conventional " ", or "DOMAIN" group lookup.
- **TOKEN**  
The token is used to enumerate groups at both the domain and on the local machine. For a local user, the groups returned will contain local groups. For a domain user, the groups returned will contain both domain and local groups. There is no equivalent in conventional group lookup.

For example, the following setting of **DB2\_GRP\_LOOKUP** enables access token support for enumerating local groups:

```
db2set DB2_GRP_LOOKUP=LOCAL,TOKENLOCAL
```

The next example enables access token support for enumerating groups at both the local machine as well as the location where the user ID is defined (if the account is defined at the domain):

```
db2set DB2_GRP_LOOKUP=,TOKEN
```

This final example enables access token support for enumerating domain groups at the location where the user ID is defined:

```
db2set DB2_GRP_LOOKUP=DOMAIN,TOKENDOMAIN
```

**Note:** Access token support can be enabled with all authentications types except CLIENT authentication.

### **The DB2\_GRP\_LOOKUP environment variable and Db2 group enumeration (Windows)**

On Windows, a user can belong to groups defined at the domain level, groups defined on the local machine, or to both.

The **DB2\_GRP\_LOOKUP** environment variable controls whether groups are enumerated on the local machine, or where the users are defined (on the local machine if they are a local user, or at the domain level if they are a domain user). Therefore, when

the security administrator grants authorities and privileges, care must be taken that **DB2\_GRP\_LOOKUP** is set as intended and the correct users receive the intended authorization.

If the **DB2\_GRP\_LOOKUP** profile registry variable is not set:

1. The Db2 database system first tries to find the user on the same machine.
2. If the user name is defined locally, the user is authenticated locally.
3. If the user is not found locally, the Db2 database system attempts to find the user name on it's domain, and then on trusted domains.

For example, consider the following situation where **DB2\_GRP\_LOOKUP** is not set:

1. The domain user DUSER1 is a member of the local group, GROUP1.
2. The security administrator (who holds SECADM authority) grants DBADM authority to group GROUP1.

```
GRANT DBADM ON database TO GROUP GROUP1
```

3. Because **DB2\_GRP\_LOOKUP** is not set, groups are enumerated where users are defined. So, groups for DUSER1 are enumerated at the domain level. Since DUSER1 does not belong to group GROUP1 at the domain level, DUSER1 does not receive DBADM authority.

Further, consider this more complex scenario involving the **UPGRADE DATABASE** command where **DB2\_GRP\_LOOKUP** is not set:

1. The domain user DUSER2 is a member of the local Administrators group.
2. The **sysadm\_group** configuration parameter is not set, therefore members of the local Administrators group automatically hold SYSADM authority.
3. User DUSER2 is able to issue the **UPGRADE DATABASE** command (since DUSER2 holds SYSADM authority). The **UPGRADE DATABASE** command grants DBADM authority on the database being upgraded to the SYSADM group, in this case, the Administrators group.
4. Because **DB2\_GRP\_LOOKUP** is not set, groups are enumerated where users are defined. So, groups for DUSER2 are enumerated at the domain level. Since DUSER2 does not belong to the Administrators group at the domain level, DUSER2 does not receive DBADM authority.

Possible solutions for this scenario are to make one of the following changes:

- Set **DB2\_GRP\_LOOKUP** = local
- Add the users that should have DBADM authority to the Administrators or GROUP1 group at the Domain Controller.

You can use the **SYSPROC.AUTH\_LIST\_AUTHORITIES\_FOR\_AUTHID** table function to verify the authorities held by a user, as shown in the following example for DUSER1:

```
SELECT AUTHORITY, D_USER, D_GROUP, D_PUBLIC, ROLE_USER, ROLE_GROUP, ROLE_PUBLIC, D_ROLE
FROM TABLE (SYSPROC.AUTH_LIST_AUTHORITIES_FOR_AUTHID ('DUSER1', 'U')) AS T
ORDER BY AUTHORITY
```

You can use the **SYSPROC.AUTH\_LIST\_GROUPS\_FOR\_AUTHID** table function to verify the groups to which the Db2 database manager has determined a user belongs, as shown in the following example for DUSER1:

```
SELECT * FROM TABLE (SYSPROC.AUTH_LIST_GROUPS_FOR_AUTHID ('DUSER1')) AS T
```

**Note:** If you use the same group name at both the domain level and on the local machine, because the Db2 database manager does not fully qualify the groups, this can lead to confusion.

## Authentication using an ordered domain list

User IDs may be defined more than once in a trusted domain forest. A trusted domain forest is a collection of domains that are interrelated through a network.

### About this task

It is possible for a user on one domain to have the same user ID as that for another user on a different domain. This may cause difficulties when attempting to do any of the following actions:

- Authenticating multiple users having the same user ID but on different domains.
- Group lookup for the purposes of granting and revoking privileges based on groups.
- Validation of passwords.
- Control of network traffic.

To prevent difficulties arising from the possibility of multiple users with the same user ID across a domain forest, you should use an ordered domain list as defined using the **db2set** and the registry variable **DB2DOMAINLIST**. When setting the order, the domains to be included in the list are separated by a comma. You must make a conscious decision regarding the order that the domains are searched when authenticating users.

Those user IDs that are present on domains further down the domain list will have to be renamed by you if they are to be authenticated for access.

Control of access can be done through the domain list. For example, if the domain of a user is not in the list, the user will not be allowed to connect.

**Note:** The **DB2DOMAINLIST** registry variable is effective only when CLIENT authentication is set in the database manager configuration and is needed if a single signon from a Windows desktop is required in a Windows domain environment. **DB2DOMAINLIST** is supported by some versions of Db2 servers however **DB2DOMAINLIST** will not be enforced if neither the client nor the server are in a Windows environment.

## Domain security support (Windows)

The following example illustrates how the Db2 database management system can support Windows domain security. The connection works because the user name and local group are on the same domain.

The connection works in the following scenario because the user name and local or global group are on the same domain.

Note that the user name and local or global group do not need to be defined on the domain where the database server is running, but they must be on the same domain as each other.

Table 52. Successful Connection Using a Domain Controller

Domain1	Domain2
A trust relationship exists with Domain2.	<ul style="list-style-type: none"> <li>• A trust relationship exists with Domain1.</li> <li>• The local or global group grp2 is defined.</li> <li>• The user name id2 is defined.</li> <li>• The user name id2 is part of grp2.</li> </ul>
<p>The Db2 server runs in this domain. The following Db2 commands are issued from it:</p> <pre>REVOKE CONNECT ON db FROM public GRANT CONNECT ON db TO GROUP grp2 CONNECT TO db USER id2</pre>	
The local or global domain is scanned but id2 is not found. Domain security is scanned.	
	The user name id2 is found on this domain. Db2 gets additional information about this user name (that is, it is part of the group grp2).
The connection works because the user name and local or global group are on the same domain.	

## Defining which users hold SYSADM authority (Windows )

Certain users have SYSADM authority if the **sysadm\_group** database manager configuration parameter is not set (that is, it is NULL).

These users are:

- Members of the local Administrators group
- Members of the Administrators group at the Domain Controller, if the Db2 database manager is configured to enumerate groups for users at the location where the users are defined (you can use the **DB2\_GRP\_LOOKUP** environment variable to configure group enumeration)
- Members of the DB2ADMNS group, if Windows extended security is enabled. The location of the DB2ADMNS group is decided during installation.
- The LocalSystem account

There are cases where the previously mentioned default behavior is not desirable. You can use the **sysadm\_group** database manager configuration parameter to override this behavior by using one of the following methods:

- Create a local group on the Db2 server machine and add to it users (domain users or local users) that you want to have SYSADM authority. The Db2 database manager should be configured to enumerate groups for the user on the local machine.
- Create a domain group and add to it the users that you want to have SYSADM authority. The Db2 database manager should be configured to enumerate groups for users at the location where the users are defined.

Then update the **sysadm\_group** database manager configuration parameter to this group, using the following commands:

```
DB2 UPDATE DBM CFG USING SYSADM_GROUP group_name
DB2STOP
DB2START
```

## Windows LocalSystem account support

On Windows platforms, the Db2 database system supports applications running under the context of the LocalSystem account (LSA) with local implicit connection. The authorization ID for the LocalSystem account is SYSTEM.

If you are using a non-English version of a Windows operating system, you need to check that the authorization ID for the LocalSystem account does not have an invalid character. For example, if you are using a French version of a Windows operating system, the LocalSystem account is *Système*, but you cannot use this account as an authorization ID because it has an invalid character, *è*.

The LocalSystem account is considered a system administrator (holding SYSADM authority) when the **sysadm\_group** database manager configuration parameter is set to NULL.

If there is a need for applications running under the context of the LocalSystem account to perform database actions that are not within the scope of SYSADM, you must grant the LocalSystem account the required database privileges or authorities. For example, if an application requires database administrator capabilities, grant the LocalSystem account DBADM authority using the GRANT (Database Authorities) statement.

Developers writing applications to be run under this account need to be aware that the Db2 database system has restrictions on objects with schema names starting with "SYS". Therefore if your applications contain DDL statements that create Db2 database objects, they should be written such that:

- For static queries, they should be bound with a value for the QUALIFIER options other than the default one (SYSTEM).
- For dynamic queries, the objects to be created should be explicitly qualified with a schema name supported by the Db2 database manager, or the CURRENT SCHEMA register must be set to a schema name supported by the Db2 database manager.

Group information for the LocalSystem account is gathered at the first group lookup request after the Db2 database instance is started and is not refreshed until the instance is restarted.

## Extended Windows security using the DB2ADMNS and DB2USERS groups

Extended security is enabled by default in all Db2 database products on Windows operating systems except IBM Data Server Runtime Client and Db2 Drivers. IBM Data Server Runtime Client and Db2 Drivers do not support extended security on Windows platforms.

An **Enable operating system security** check box appears on the **Enable operating system security for Db2 objects** panel when you install Db2 database products. Unless you disable this option, the installer creates two new groups, DB2ADMNS and DB2USERS. DB2ADMNS is the Db2 Administrators Group and DB2USERS is the Db2 Users Group. DB2ADMNS and DB2USERS are the default group names; optionally, you can specify different names for these groups at installation time (if you select silent installation, you can change these names within the installation response file). If you choose to use groups that exist on your system, be aware that the privileges of these groups are modified. They are given the privileges, as required, listed in the table, below.



It is important to understand that these groups are used for protection at the operating-system level and are in no way associated with Db2 authority levels. However, the Db2 Administrators Group (ex. DB2ADMNS) is used as the default group for SYSADM, SYSMANT, and SYSCTRL when no values are specified for database manager configuration parameters SYSADM\_GROUP, SYSMANT\_GROUP and SYSCTRL\_GROUP. It is recommended that if you are specifying a SYSADM group, then that group should be the Db2 Administrators Group. This setting can be established after installation, by an administrator.

**Note:** You can specify your Db2 Administrators Group (DB2ADMNS or the name you chose during installation) and Db2 Users Group (DB2USERS or the name you chose during installation) either as local groups or as domain groups. Both groups must be of the same type, so either both local or both domain.

If you change the computer name, and the computer groups DB2ADMNS and DB2USERS are local computer groups, you must update the DB2\_ADMINGROUP and DB2\_USERSGROUP global registries. To update the registry variables after renaming and restarting the computer run the following command:

1. Open a command prompt.
2. Run the **db2extsec** command to update security settings:  
`db2extsec -a new computer name\DB2ADMNS -u new computer name\DB2USERS`

**Note:** If extended security is enabled in Db2 database products on Windows 7, only users that belong to the DB2ADMNS group can run the graphical Db2 administration tools. In addition, members of the DB2ADMNS group need to launch the tools with full administrator privileges. This is accomplished by right-clicking on the shortcut and then choosing "Run as administrator".

## Abilities acquired through the DB2ADMNS and DB2USERS groups

The DB2ADMNS and DB2USERS groups provide members with the following abilities:

- DB2ADMNS  
Full control over all Db2 objects (see the following list of protected objects)
- DB2USERS  
Read and Execute access for all Db2 objects located in the installation and instance directories, but no access to objects under the database system directory and limited access to IPC resources  
For certain objects, there may be additional privileges available, as required (for example, write privileges, add or update file privileges, and so on). Members of this group have no access to objects under the database system directory.

**Note:** The meaning of Execute access depends on the object; for example, for a .dll or .exe file having Execute access means you have authority to execute the file, however, for a directory it means you have authority to traverse the directory.

Ideally, all Db2 administrators should be members of the DB2ADMNS group (as well as being members of the local Administrators group), but this is not a strict requirement. Everyone else who requires access to the Db2 database system *must* be a member of the DB2USERS group. To add a user to one of these groups:

1. Launch the Users and Passwords Manager tool.
2. Select the user name to add from the list.

3. Click Properties. In the Properties window, click the Group membership tab.
4. Select the Other radio button.
5. Select the appropriate group from the drop-down list.

## Adding extended security after installation (db2extsec command)

If the Db2 database system was installed without extended security enabled, you can enable it by executing the command **db2extsec**. To execute the **db2extsec** command you must be a member of the local Administrators group so that you have the authority to modify the ACL of the protected objects.

You can run the **db2extsec** command multiple times, if necessary, however, if this is done, you cannot disable extended security unless you issue the **db2extsec -r** command immediately after *each* execution of **db2extsec**.

## Removing extended security

### CAUTION:

**Do not remove extended security after it has been enabled unless absolutely necessary.**

You can remove extended security by running the command **db2extsec -r**, however, this will only succeed if no other database operations (such as creating a database, creating a new instance, adding table spaces, and so on) have been performed after enabling extended security. The safest way to remove the extended security option is to uninstall the Db2 database system, delete all the relevant Db2 directories (including the database directories) and then reinstall the Db2 database system without extended security enabled.

## Protected objects

The *static* objects that can be protected using the DB2ADMNS and DB2USERS groups are:

- File system
  - File
  - Directory
- Services
- Registry keys

The *dynamic* objects that can be protected using the DB2ADMNS and DB2USERS groups are:

- IPC resources, including:
  - Pipes
  - Semaphores
  - Events
- Shared memory

## Privileges owned by the DB2ADMNS and DB2USERS groups

The privileges assigned to the DB2ADMNS and DB2USERS groups are listed in the following table:

Table 53. Privileges for DB2ADMNS and DB2USERS groups

Privilege	DB2ADMNS	DB2USERS	Reason
Create a token object (SeCreateTokenPrivilege)	Y	N	Token manipulation (required for certain token manipulation operations and used in authentication and authorization)
Replace a process level token (SeAssignPrimaryTokenPrivilege)	Y	N	Create process as another user
Increase quotas (SeIncreaseQuotaPrivilege)	Y	N	Create process as another user
Act as part of the operating system (SeTcbPrivilege)	Y	N	LogonUser
Generate security audits (SeSecurityPrivilege)	Y	N	Manipulate audit and security log
Take ownership of files or other objects (SeTakeOwnershipPrivilege)	Y	N	Modify object ACLs
Increase scheduling priority (SeIncreaseBasePriorityPrivilege)	Y	N	Modify the process working set
Backup files and directories (SeBackupPrivilege)	Y	N	Profile/Registry manipulation (required to perform certain user profile and registry manipulation routines: LoadUserProfile, RegSaveKey(Ex), RegRestoreKey, RegReplaceKey, RegLoadKey(Ex))
Restore files and directories (SeRestorePrivilege)	Y	N	Profile/Registry manipulation (required to perform certain user profile and registry manipulation routines: LoadUserProfile, RegSaveKey(Ex), RegRestoreKey, RegReplaceKey, RegLoadKey(Ex))
Debug programs (SeDebugPrivilege)	Y	N	Token manipulation (required for certain token manipulation operations and used in authentication and authorization)
Manage auditing and security log (SeAuditPrivilege)	Y	N	Generate auditing log entries
Log on as a service (SeServiceLogonRight)	Y	N	Run Db2 as a service
Access this computer from the network (SeNetworkLogonRight)	Y	Y	Allow network credentials (allows the Db2 database manager to use the LOGON32_LOGON_NETWORK option to authenticate, which has performance implications)
Impersonate a client after authentication (SeImpersonatePrivilege)	Y	N	Client impersonation (required for Windows to allow use of certain APIs to impersonate Db2 clients: ImpersonateLoggedOnUser, ImpersonateSelf, RevertToSelf, and so on)
Lock pages in memory (SeLockMemoryPrivilege)	Y	N	Large Page support
Create global objects (SeCreateGlobalPrivilege)	Y	Y	Terminal Server support (required on Windows)

## Considerations for Windows7: User Access Control feature

The User Access Control (UAC) feature of Windows 7 impacts the Db2 database system in the following ways.

### Starting applications with full administrative privileges

On Windows 7, by default, applications start with only standard user rights, even if the user is a local administrator. To start an application with further privileges, you need to launch the command from a command window that is running with full administrative privileges. The Db2 installation process creates a shortcut called "Command window - Administrator" specifically for Windows 7 users. It is recommended that you launch this shortcut if you want to run administrative commands.

If you do not have full administrative privileges and you attempt to perform Db2 administration tasks from a command prompt or graphical tool on Windows 7, you can encounter various error messages implying that your access is denied and the tasks will fail to complete successfully.

To determine whether the action you are performing is considered to be an administration task, check whether any of the following are true:

- It requires SYSADM, SYSCTRL or SYSMAINT authority
- It modifies registry keys under the HKLM branch in the registry
- It writes to the directories under the Program Files directory

For example, the following actions are all considered to be administration tasks:

- Creating and dropping Db2 instances
- Starting and stopping Db2 instances
- Creating databases
- Updating database manager configuration parameters or Db2 Administration Server (DAS) configuration parameters
- Updating CLI configuration parameters and configuring system data source names (DSN)
- Starting the Db2 trace facility
- Running the **db2pd** utility
- Changing Db2 profile registry variables

To resolve the problem, you must perform Db2 administration tasks from a command prompt or graphical tool that is running with full administrator privileges. To launch a command prompt or graphical tool with full administrator privileges, right-click on the shortcut and then select **Run as administrator**.

**Note:** If extended security is enabled, you also need to be a member of the DB2ADMNS group in order to launch the graphical administration tools (such as the IBM Data Studio).

### User data location

User data (for example, files under instance directories) is stored in ProgramData\IBM\DB2\copy\_name, where copy\_name is the name of the Db2 copy (by default, DB2COPY1 is the name of the first copy installed). On Windows versions other than Windows 7, user data is stored in Documents and Settings\All Users\Application Data\IBM\DB2\copy\_name.

---

## Db2 and UNIX security

There are some security considerations specific to UNIX platforms that you need to be aware of.

The Db2 database does not support root acting directly as a database administrator. You should use **su - <instance owner>** as the database administrator.

For security reasons, in general, do not use the instance name as the Fenced ID. However, if you are not planning to use fenced UDFs or stored procedures, you can set the Fenced ID to the instance name instead of creating another user ID.

The recommendation is to create a user ID that is recognized as being associated with this group. The user for fenced UDFs and stored procedures is specified as a parameter of the instance creation script (**db2icrt ... -u <FencedID>**). This is not required if you install the Db2 Clients or the Db2 Software Developer's Kit.

---

## Db2 and Linux security

There are some security considerations specific to Linux platforms that you might need to be aware of.

### Change password support (Linux)

Db2 database products provide support for changing passwords on Linux operating systems.

This support is implemented through the use of security plug-in libraries called **IBMOSchgpwdclient.so** and **IBMOSchgpwdserver.so**.

To enable password change support on Linux, set the database manager configuration parameter **clnt\_pw\_plugin** to **IBMOSchgpwdclient** and **srvcon\_pw\_plugin** to **IBMOSchgpwdserver**.

You must also create a PAM configuration file called "db2" in the **/etc/pam.d** directory.

### Deploying a change password plug-in (Linux)

To enable support for changing passwords in Db2 database products on Linux, you must configure the Db2 instance to use the security plug-ins **IBMOSchgpwdclient** and **IBMOSchgpwdserver**.

#### Before you begin

The plug-in libraries are located in the following directories:

- **INSTHOME/sqllib/securityXX/plugin/IBM/client/IBMOSchgpwdclient.so**
- **INSTHOME/sqllib/securityXX/plugin/IBM/server/IBMOSchgpwdserver.so**

where **INSTHOME** is the home directory of the instance owner and **securityXX** is either **security32** or **security64**, depending on the bit-width of the instance.

#### Procedure

To deploy the security plug-ins in a Db2 instance, perform the following steps:

1. Log in as a user with root authority.

2. Create a PAM configuration file: /etc/pam.d/db2

Ensure that the file contains the appropriate set of rules, as defined by your system administrator. For example, on SLES 9 this can be used:

```
auth required pam_unix2.so nullok
account required pam_unix2.so
password required pam_pwcheck.so nullok tries=1
password required pam_unix2.so nullok use_authtok use_first_pass
session required pam_unix2.so
```

And on RHEL, this can be used:

```
##PAM-1.0
auth required /lib/security/$ISA/pam_env.so
auth sufficient /lib/security/$ISA/pam_unix.so likeauth nullok
auth required /lib/security/$ISA/pam_deny.so

account required /lib/security/$ISA/pam_unix.so
account sufficient /lib/security/$ISA/pam_succeed_if.so uid < 100 quiet
account required /lib/security/$ISA/pam_permit.so

password requisite /lib/security/$ISA/pam_cracklib.so retry=3 dcredit=-1
ucredit=-1
password sufficient /lib/security/$ISA/pam_unix.so nullok use_authtok md5
shadow remember=3
password required /lib/security/$ISA/pam_deny.so

session required /lib/security/$ISA/pam_limits.so
session required /lib/security/$ISA/pam_unix.so
```

3. Enable the security plug-ins in the Db2 instance:

- a. Update the database manager configuration parameter **SRVCON\_PW\_PLUGIN** with the value **IBMOSchgpwdsrver**:  
db2 update dbm cfg using srvcon\_pw\_plugin IBMOSchgpwdsrver
- b. Update the database manager configuration parameter **CLNT\_PW\_PLUGIN** with the value **IBMOSchgpwdclient**:  
db2 update dbm cfg using CLNT\_PW\_PLUGIN IBMOSchgpwdclient
- c. Ensure that either the database manager configuration parameter **SRVCON\_AUTH** is set to a value of **CLIENT**, **SERVER**, **SERVER\_ENCRYPT**, **DATA\_ENCRYPT**, or **DATA\_ENCRYPT\_CMP**, or the database manager configuration parameter **SRVCON\_AUTH** is set to a value of **NOT\_SPECIFIED** and **AUTHENTICATION** is set to a value of **CLIENT**, **SERVER**, **SERVER\_ENCRYPT**, **DATA\_ENCRYPT**, or **DATA\_ENCRYPT\_CMP**.

---

# Index

## Special characters

.NET

- .Net Data Provider clients 78
- GSKit 78
- SSL 78

## A

access control

- authentication 7
- column-specific 183
- DBADM (database administration)
  - authority 60
- fine-grained row and column
  - See RCAC 161
- label-based access control 183
- row-specific 183
- tables 58
- views 58

access tokens

- Windows 365

ACCESSCTRL (access control) authority

- details 37
- overview 32

adding a master key to a local

- keystore 100

AIX

- authentication methods 242
- configuring transparent LDAP 241

AIX encrypted file system (EFS) 93

already 114

ALTER privilege 43, 46

alternate\_auth\_enc configuration

parameter

- encrypting using AES 256-bit
- algorithm 7

APIs

- communication exit library
  - db2commexitDeregister 304
  - db2commexitFreeErrormsg 308
  - db2commexitInit 300
  - db2commexitRecv 305
  - db2commexitRegister 302
  - db2commexitSend 306
  - db2commexitTerm 302
  - db2commexitUserIdentity 307
- overview 300
- group plug-in
  - db2secDoesGroupExist 265
  - db2secFreeErrormsg 266
  - db2secFreeGroupListMemory 266
  - db2secGetGroupsForUser 267
  - db2secGroupPluginInit 270
  - db2secPluginTerm 271
- group retrieval plug-in 264
- security plug-in
  - db2secClientAuthPluginInit 277
  - db2secClientAuthPluginTerm 278
  - db2secDoesAuthIDExist 279
  - db2secDoesGroupExist 265

APIs (*continued*)

security plug-in (*continued*)

- db2secFreeErrormsg 266
  - db2secFreeGroupListMemory 266
  - db2secFreeInitInfo 279
  - db2secFreeToken 280
  - db2secGenerateInitialCred 280
  - db2secGetAuthIDs 282
  - db2secGetDefaultLoginContext 284
  - db2secGetGroupsForUser 267
  - db2secGroupPluginInit 270
  - db2secPluginTerm 271
  - db2secProcessServerPrincipalName 285
  - db2secRemapUserid 286
  - db2secServerAuthPluginInit 287
  - db2secServerAuthPluginTerm 290
  - db2secValidatePassword 290
  - overview 263
  - user ID/password plug-ins 271
- archivepath parameter 122
- archiving
- audit log files 122
- AUDIT events 354
- audit facility
- actions 116
  - archive 128
  - asynchronous record writing 137
  - audit data in tables
    - creating tables 126
    - loading tables 127
  - audit events table 323
  - authorities 116
  - behavior 137
  - CHECKING access approval
    - reasons 328
  - CHECKING access attempted
    - types 329
  - checking events table 326
  - CONTEXT events table 347
  - error handling 137
  - ERRORTYPE parameter 137
  - events 116
  - EXECUTE events 130, 348
  - EXECUTE timestamp 134
  - object record types 321
  - OBJMAINT events table 332
  - overview 116
  - policies 118
  - privileges 116
  - record layouts 321
  - record object types 321
  - records for EXECUTE events 348
  - SECMAINT authorities 340
  - SECMAINT events table 335
  - SECMAINT privileges 340
  - synchronous record writing 137
  - SYSADMIN events table 343
  - techniques 139
  - tips 139
  - VALIDATE events table 345

audit logs

- archiving 122, 128
- file names 125
- location 122
- audit\_buf\_sz configuration parameter
  - determining timing of writing audit records 137

authentication

- details 2
- domain security 364
- groups 364
- GSS-API 225
- ID and password 225
- Kerberos 14, 225
- LDAP users 251

lookup

- configuring 241, 244
- methods 7
- ordered domain list 368
- overview 1
- partitioned databases 14
- plug-ins

API for checking whether authentication ID exists 279

API for cleaning up client authentication plug-in resources 278

API for cleaning up resources held by db2secGenerateInitialCred API 279

API for cleaning up server authentication plug-in resources 290

API for getting authentication IDs 282

API for initializing client authentication plug-in 277

API for initializing server authentication plug-in 287

API for validating passwords 290

APIs for user ID/ password authentication plug-ins 271

deploying 235, 238, 375

LDAP 239

library locations 229

security 225

remote clients 13

security plug-ins 225

two-part user IDs 231

types

- CLIENT 7
- DATA\_ENCRYPT 7
- DATA\_ENCRYPT\_CMP 7
- GSS\_SERVER\_ENCRYPT 7
- GSSPLUGIN 7
- KERBEROS 7
- KRB\_SERVER\_ENCRYPT 7
- SERVER 7
- SERVER\_ENCRYPT 7
- AUTHID\_ATTRIBUTE 245

- authorities
  - access control (ACCESSCTRL) 37
  - audit policy 118
  - data access (DATAACCESS) 38
  - database administration (DBADM) 35, 41
  - explain administration (EXPLAIN) 40
  - implicit schema (IMPLICIT\_SCHEMA) 41
  - LOAD 40
  - overview 20, 26
  - removing DBADM from SYSCtrl 30
  - security administrator (SECADM) 34
  - SQL administration (SQLADM) 38
  - system administration (SYSADM) 29
  - system control (SYSCtrl) 30
  - system maintenance (SYSMAINT) 31
  - system monitor (SYSMON) 32
  - workload administration (WLMADM) 39
- authorization IDs
  - details 3
  - implicit authorizations 56
  - LDAP 248
  - security model overview 1
  - SETSESSIONUSER privilege 41
  - trusted client 7
  - types 47
- authorization names
  - creating views for privileges information 220
  - retrieving
    - names with DBADM authority 219
    - names with granted privileges 218
    - names with table access authority 219
    - privileges granted to 220

## B

- backup image 114, 115, 116
- backups
  - encrypting 90
  - security risks 61, 90
- BIND command
  - package re-creation ownership 56
- BIND privilege 44
- BINDADD authority
  - details 32
- binding
  - rebinding invalid packages 54
- built-in views
  - AUTHORIZATIONIDS
    - example 218
    - restricting access 220
  - OBJECTOWNERS
    - restricting access 220
  - PRIVILEGES
    - example 218
    - restricting access 220

## C

- centralized key manager 101, 104, 106, 107, 112
- centralized keystore 101, 107
- certificate authorities
  - digital certificates 84
- check backup image is encrypted 115
- check if database is encrypted 114
- CHECKING events 354
- cipher suites 87
- client authentication plug-ins 239
- CLIENT authentication type
  - details 7
- columns
  - LBAC protection
    - adding 201
    - removing 215
  - LBAC-protected
    - dropping 212
    - inserting 206
    - reading 203
    - updating 208
- communication buffer exit library
  - developing
    - control over connections 311
    - DATA\_ENCRYPT authentication 318
    - functions structure 308
    - information structure 310
  - overview 295
- communication exit library
  - APIs
    - db2commexitDeregister 304
    - db2commexitFreeErrorMsg 308
    - db2commexitInit 300
    - db2commexitRecv 305
    - db2commexitRegister 302
    - db2commexitSend 306
    - db2commexitTerm 302
    - db2commexitUserIdentity 307
  - deploying 295
  - developing
    - API calling sequences (connection concentrator) 316
    - API calling sequences (no connect reset) 315
    - API calling sequences (normal connect) 314
    - API calling sequences (overview) 314
    - API calling sequences (SET SESSION AUTHORIZATION) 317
    - API calling sequences (trusted context) 316
    - API versions 311
    - buffer structure 311
    - connect gateway 318
    - error handling 312
    - overview 299
    - restrictions 312
    - return codes 312
    - target logical node 318
  - enabling 297, 298
  - library loading 299
  - location 296
  - naming conventions 296

- communication exit library (*continued*)
  - permissions 296
  - problem determination 298
- configuration
  - LDAP
    - plug-ins 245
- Configuration file
  - PKCS #11 108
- configure 98
- CONNECT authority 32
- CONTEXT events 354
- CONTROL privilege
  - details 43
  - implicit 56
  - packages 44
  - views 43
- create backup image 114
- create database 113
- CREATE DATABASE command
  - RESTRICTIVE option 220
- create encrypted database 113
- create keystore 100
- create master key 100
- CREATE ROLE statement
  - creating roles 144
  - granting membership in roles 144
- CREATE TRUSTED CONTEXT statement
  - example 156
- CREATE\_EXTERNAL\_ROUTINE
  - authority 32
- CREATE\_NOT\_FENCED\_ROUTINE
  - authority 32
- CREATETAB authority 32
- Creating a stash file 110
- creating encrypted backup images 114
- cryptography 96
  - public key 85

## D

- data
  - audit
    - creating tables 126
    - loading into tables 127
  - encrypting 64
  - indirect access 61
  - inserting
    - LBAC-protected 206
  - label-based access control (LBAC)
    - adding protection 201
    - inserting 206
    - overview 201
    - reading 203
    - unprotecting 215
    - updating 208
  - security
    - overview 1
    - system catalog 220
- data at rest 90
- DATAACCESS (data access) authority
  - details 38
  - overview 32
- database authorities
  - granting
    - overview 32
  - overview 32
  - revoking 32



- database backup image 114, 115
- database directories
  - permissions 6
- database is encrypted 114
- database objects
  - roles 143
- database-level authorities
  - overview 26
- databases
  - accessing
    - default authorities 49
    - default privileges 49
    - implicit privileges through packages 56
  - label-based access control (LBAC) 183
- datapath parameter 122
- DB2 native encryption 96, 98
  - Data encryption key 96
  - enclib 96
  - encropts 96
  - Key manager 96
  - Keystore 96
  - keystore\_location 96
  - keystore\_type 96
  - Master key 96
  - Setting up 99
- DB2\_GRP\_LOOKUP environment variable 366, 369
- DB2\_GRP\_LOOKUP registry variable 365
- DB2ADMNS group
  - defining who holds SYSADM authority 369
  - details 370
- db2audit.log file 116
- db2cluster command
  - Db2 cluster services administrator 141
  - security model 141
- DB2COMM registry variable
  - configuring Secure Sockets Layer (SSL) support 64
- DB2LBACRULES LBAC rule set 195
- DB2LDAPSecurityConfig environment variable
  - overview 245
- db2p12tokmip 107
- DB2SECURITYLABEL data type
  - providing explicit values 200
  - viewing as string 200
- DB2USERS user group
  - details 370
- DBADM (database administration) authority
  - controlling access 60
  - details 35
  - overview 32
  - retrieving names 219
- debugging
  - security plug-ins 233
- decrypt 96
- default privileges 49
- DELETE privilege 43
- different location 115
- different systems 115, 116

- digital certificates
  - managing 64
  - overview 84
- distinguished name (DN) 248
- domain controller
  - overview 361
- domains
  - ordered domain list 368
  - security
    - authentication 364
    - trust relationships 364
    - Windows 368
- dynamic SQL
  - EXECUTE privilege 56

## E

- efsenable command 93
- efskeymgr command 93
- efsmgr command 93
- ENABLE\_SSL parameter 245
- enclib 114
- encropts 114
- encrypt database 114
- encrypted backup image 114, 115, 116
- encrypted database 113
- encrypted file system (EFS) 93
- encryption 98, 115
  - data 64
- enumeration of groups 366
- error messages
  - security plug-ins 259
- errors
  - switching user 159
  - trusted contexts 159
- ExampleBANK RCAC scenario
  - column masks 181
  - data queries 181
  - database tables 179
  - database users and roles 178
  - introduction 177
  - row permissions 180
  - security policy 178
- ExampleHMO RCAC scenario
  - column masks 169
  - data queries 171
  - data updates 171
  - database tables 165
  - database users and roles 164
  - inserting data 170
  - introduction 163
  - revoke authority 177
  - row permissions 168
  - secure functions 174
  - secure triggers 176
  - security administration 167
  - security policy 163
  - view creation 173
- EXECUTE category
  - audit information 134
  - audit records 348
  - overview 130
  - replaying activities 135
- EXECUTE events 354
- EXECUTE privilege
  - database access 56
  - packages 44

- EXECUTE privilege (*continued*)
  - routines 46
- existing database 114
- EXPLAIN authority
  - details 40
  - overview 32
- explicit trusted connections
  - establishing 151
  - user ID switching 151, 157
- extended Windows security 370

## F

- FGAC
  - See RCAC 161
- file names
  - audit logs 125
- fine-grained access control
  - See RCAC 161
- firewalls
  - application proxy 223
  - circuit level 224
  - details 223
  - screening router 223
  - stateful multi-layer inspection (SMLI) 224
- functions
  - privileges 46
  - scalar
    - DECRYPT\_BIN 64
    - DECRYPT\_CHAR 64
    - ENCRYPT 64
    - GETHINT 64

## G

- global group support 363
- Global Security Kit 98
- GRANT statement
  - example 54
  - implicit authorizations 56
  - overview 54
- group lookup support
  - details 239, 250
- GROUP\_BASEDN parameter 245
- GROUP\_LOOKUP\_ATTRIBUTE attribute 250
- GROUP\_LOOKUP\_METHOD parameter
  - configuring LDAP plug-in modules 245, 250
- GROUP\_OBJECTCLASS parameter 245
- GROUPNAME\_ATTRIBUTE parameter 245
- groups
  - access token 365
  - enumeration (Windows) 366
  - names 363
  - roles comparison 149
  - selecting 4
  - user authentication 364
- gsk8capicmd 100
- GSKCapiCmd tool
  - configuring Secure Sockets Layer (SSL) support 64, 76, 78
- GSKit 98, 100

- GSKit (*continued*)
  - configuring Secure Sockets Layer (SSL) support 64, 76, 78
  - library rules 89
  - process rules 89
- GSS-APIs
  - authentication plug-ins 293

## H

- handshakes
  - overview 83

## I

- IBMLDAPSecurity.ini file 245
- IKEYCMD tool 64, 76, 78
- iKeyman tool 64, 76, 78
- implicit authorization
  - managing 56
- IMPLICIT\_SCHEMA (implicit schema)
  - authority
    - details 41
    - overview 32
- import master key 100
- INDEX privilege
  - details 45
- indexes
  - INDEX privilege
    - expression-based indexes 45
  - privileges
    - expression-based indexes 45
    - overview 45
- insert master key 100
- INSERT privilege 43
- instance directories 6
- instances
  - authorities 26
  - configuring
    - SSL communications 64
- Internal system-defined routine
  - SECADM 29
- Internal system-defined routines 29
- ISKLM 101, 104

## K

- Kerberos authentication protocol
  - enabling 18
  - IBM i compatibility 19
  - mapping 16
  - naming 16
  - overview 14
  - plug-ins
    - creating 19
    - deploying 238
  - principals 16
  - server 7
  - setting up 14
  - System z compatibility 19
  - Windows compatibility 19
- key manager 98, 101
- keydb 100
- keystore 98, 101
  - configuring 112

- KRB\_SERVER\_ENCRYPT authentication type 7

## L

- label-based access control
  - See LBAC 183
- LBAC
  - credentials 183
  - dropping columns 212
  - inserting data 206
  - overview 20, 183
  - protected tables 183
  - reading data 203
  - removing protection 215
  - rule exemptions
    - details 199
    - effect on security label comparisons 194
  - rule sets
    - comparing security labels 194
    - DB2LBACRULES 195
    - overview 195
  - security administrators 183
  - security labels
    - ARRAY component type 187
    - comparisons 194
    - compatible data types 191
    - components 186
    - creating 191
    - details 191
    - dropping 191
    - granting 191
    - overview 183
    - revoking 191
    - SET component type 187
    - string format 193
    - TREE component type 188
  - security policies
    - adding to a table 201
    - details 185
    - overview 183
  - updating data 208
- LDAP
  - plug-ins 245, 248
  - security plug-ins 239
  - transparent
    - AIX 241
    - Kerberos 242
    - Linux 244
- LDAP\_HOST parameter 245
- libraries
  - security plug-ins
    - loading in DB2 251
    - restrictions 252
- Linux
  - security 375
  - transparent LDAP 244
- LOAD authority
  - details 40
  - overview 32
- local key manager 98, 115, 116
- local keystore 98, 100, 112
- local keystore file 100
- LocalSystem account
  - authorization 29
  - support 370

- LocalSystem account (*continued*)
  - SYSADM authority 369
- logs
  - audit 116

## M

- master key 100, 112
- methods
  - privileges 46
- migrate 107
- Migrating
  - Local keystore to PKCS #11 keystore 111
- migration
  - roles 150

## N

- naming conventions
  - Windows restrictions 363
- native encryption 98, 113, 114
- NESTED\_GROUPS parameter 245
- nicknames
  - privileges
    - indirect through packages 57

## O

- objects
  - ownership 20
- OBJMAINT events 354
- ordered domain lists 368
- ownership
  - database objects 20, 217

## P

- packages
  - access privileges for queries 56
  - authorization IDs
    - derivation 47
    - use 47
  - ownership 56
  - privileges
    - overview 44
    - revoking (overview) 54
- passwords
  - changing
    - Linux 375
  - maintaining on servers 20
- permissions
  - authorization overview 3
  - column-specific protection 183
  - directories 6
  - row-specific protection 183
- PKCS #11 configuration file 108
- PKCS #11 key manager 108
- PKCS #11 keystore 108
  - Migrating from local keystore 111
  - Set up 108
  - Stash file 110
- PKCS#12 100
- plug-ins
  - group retrieval 264

- plug-ins (*continued*)
  - GSS-API authentication 293
  - LDAP 239
  - security
    - APIs 259, 263
    - deploying 235, 236, 238, 375
    - error messages 259
    - naming conventions 230
    - restrictions (GSS-API authentication) 294
    - restrictions (plug-in libraries) 252
    - restrictions (summary) 254
    - return codes 256
    - versions 233
  - user ID/password authentication 271
- pluggable authentication module
  - AIX 241
  - Linux 244
- PRECOMPILE command
  - OWNER option 56
- prerequisite 98, 101, 104, 106
- Prerequisites 98
- Prerequisites for DB2 native encryption 98
- privileges
  - acquiring through trusted context
    - roles 156
  - ALTER
    - sequences 46
    - tables 43
  - CONTROL 43
  - DELETE 43
  - EXECUTE
    - routines 46
  - GRANT statement 54
  - granting
    - roles 149
  - hierarchy 20
  - INDEX 43
  - indexes
    - expression-based indexes 45
    - overview 45
  - indirect
    - packages containing
      - nicknames 57
  - individual 20
  - information about granted
    - retrieving 218, 220
  - INSERT 43
  - overview 20
  - ownership 20
  - packages
    - creating 44
    - implicit 20
  - planning 3
  - REFERENCES 43
  - revoking
    - overview 54
    - roles 146
  - roles 143
  - schemas 41
  - SELECT 43
  - SETSESSIONUSER 41
  - system catalog
    - privilege information 217
    - restricting access 220
  - table spaces 42

- privileges (*continued*)
  - tables 43
  - UPDATE 43
  - USAGE
    - sequences 46
    - workloads 47
  - views 43
- Privileges
  - Public
    - Routine 51
- problem determination
  - security plug-ins 233
- procedures
  - privileges 46
- PUBLIC
  - database authorities automatically granted 32
- Public Key Cryptography Standard #12 100
- public-key cryptography 85

## Q

- QUIESCE\_CONNECT authority 32

## R

- RCAC
  - conditions in masks 163
  - conditions in permissions 163
  - ExampleBANK scenario 177
  - ExampleHMO scenario 163
  - overview 161
  - rules
    - overview 162
    - SQL statements 162
  - scalar functions 163
  - scenarios
    - ExampleBANK 177
    - ExampleHMO 163
- records
  - audit 116
- recover database 115
- recover encrypted backup image 115
- recovering an encrypted database 114
- REFERENCES privilege 43
- registry variables
  - DB2COMM 64
- replaying activities
  - example 135
- requirement 98, 101, 104, 106
- restore 115, 116
- restore database 115, 116
- restore encrypted backup image 115
- restoring encrypted backup images 114
- RESTRICTIVE parameter of CREATE DATABASE command
  - denying privileges to PUBLIC 220
- Retrieving encrypted database 114
- REVOKE statement
  - example 54
  - implicit issuance 56
  - overview 54
- roles
  - creating 144
  - details 143

- roles (*continued*)
  - hierarchies 146
  - migrating from IBM Informix Dynamic Server 150
  - revoking privileges 146
  - versus groups 149
  - WITH ADMIN OPTION clause 148
- rotate master key 112
- routine invoker authorization IDs 47
- Routines
  - Privilege
    - Public 51
- row and column access control
  - See RCAC 161
- rows
  - deleting
    - LBAC-protected data 212
  - inserting
    - LBAC-protected data 206
  - protecting with LBAC 201
  - reading when using LBAC 203
  - removing LBAC 215
  - updating
    - LBAC-protected data 208
- rule sets (LBAC)
  - details 195
  - exemptions 199

## S

- SafeNet
  - KeySecure 106
- same location 115
- Savepoint ID field 130
- schemas
  - privileges 41
- SEARCH\_DN parameter 245
- SEARCH\_PW parameter 245
- SECADM
  - Internal system-defined routine 29
- SECADM (security administrator)
  - authority
    - details 34
    - overview 32
- SECLABEL scalar function
  - overview 200
- SECLABEL\_BY\_NAME scalar function
  - overview 200
- SECLABEL\_TO\_CHAR scalar function
  - overview 200
- SECMAINT events 354
- security
  - authentication 2
  - CLIENT level 7
  - column-specific 183
  - communication buffer exit libraries
    - control over connections 311
  - DATA\_ENCRYPT
    - authentication 318
  - functions structure 308
  - overview 295
  - communication buffer exit library
    - information structure 310
  - communication exit libraries
    - API calling sequences (connection concentrator) 316

## security (continued)

- communication exit libraries
  - (continued)
  - API calling sequences (no connect reset) 315
  - API calling sequences (normal connection) 314
  - API calling sequences (overview) 314
  - API calling sequences (SET SESSION AUTHORIZATION statement) 317
  - API calling sequences (trusted context) 316
  - API summary 300
  - API versions 311
  - buffer structure 311
  - connect gateway 318
  - deploying 295
  - developing 299, 312
  - enabling 297, 298
  - error handling 312
  - library loading 299
  - location 296
  - naming conventions 296
  - permissions 296
  - problem determination 298
  - restrictions 312
  - return codes 312
  - target logical node 318
- data 1
- db2extsec command 370
- disabling extended security 370
- enabling extended security 370
- encryption 85, 87
- explicit trusted connections 151
- extended security 370
- fine-grained access control
  - See RCAC 161
- indirect access to data 61
- label-based access control (LBAC) 183
- NIST SP 800-131A
  - instance configuration 85
  - LDAP configuration 87
- passwords on servers 20
- plug-ins
  - 32-bit considerations 233
  - 64-bit considerations 233
  - API calling sequence 259
  - APIs 263, 265, 266, 267, 270, 271, 277, 278, 279, 280, 282, 284, 285, 286, 287, 290
  - APIs (group retrieval) 264
  - APIs (GSS-API) 293
  - APIs (user ID/password) 271
  - APIs (versions) 233
  - deploying 225, 235, 236, 238, 254, 375
  - developing 225, 252
  - enabling 225
  - error messages 259
  - group retrieval 235
  - GSS-API (deploying) 236
  - GSS-API (restrictions) 294
  - IBMOSchgpwdclient 375
  - IBMOSchgpwdsrver 375

## security (continued)

- plug-ins (continued)
  - initialization 251
  - Kerberos 238
  - LDAP (Lightweight Directory Access Protocol) 239
  - libraries 229
  - loading 225, 251
  - naming 230
  - overview 225
  - problem determination 233
  - restrictions on libraries 252
  - return codes 256
  - SQLCODE values 233
  - SQLSTATE values 233
  - two-part user ID support 231
  - user ID/password 235
- row and column access control
  - See RCAC 161
- row-specific 183
- trusted contexts 153
- UNIX 375
- Windows
  - domain security 368
  - extended 370
  - overview 361
  - users 369
- security labels (LBAC)
  - ARRAY component type 187
  - compatible data types 191
  - components 186
  - overview 191
  - policies
    - details 185
  - SET component type 187
  - string format 193
  - TREE component type 188
- SELECT privilege 43
- sequences
  - privileges 46
- server authentication plug-ins 239
- SERVER authentication type
  - overview 7
- SERVER\_ENCRYPT authentication type
  - overview 7
- session authorization IDs
  - overview 47
- SET ENCRYPTION PASSWORD statement
  - encrypting passwords 64
- SETSESSIONUSER privilege
  - details 41
- SQL statements
  - authorization IDs 47
- SQLADM (SQL administration) authority
  - details 38
  - overview 32
- SSL 104, 106
  - CATALOG TCP/IP NODE command 78
  - certificate authorities 84
  - cipher suites 87
  - CLI clients 78
  - CLP clients 78
  - configuring
    - DB2 clients 76, 78
    - DB2 instances 64

## SSL (continued)

- DB2 Connect 64
- digital certificates 84
- embedded SQL clients 78
- handshake 83
- overview 83
- SSL configuration
  - primary and secondary HADR servers
    - Linux AMD64/Intel 73
- ssl\_cipherspecs configuration parameter
  - specifying cipher suites 64, 87
- ssl\_clnt\_keydb configuration parameter
  - configuring SSL 76, 78
- ssl\_clnt\_stash configuration parameter
  - configuring SSL 78
- SSL\_KEYFILE 245
- SSL\_PW 245
- SSL\_RSA\_FIPS\_WITH\_3DES\_EDE\_CBC\_SHA cipher suite 64
- ssl\_svcname configuration parameter
  - configuring SSL 64
- ssl\_svr\_keydb configuration parameter
  - configuring SSL 64
- ssl\_svr\_stash configuration parameter
  - configuring SSL 64
- ssl\_versions configuration parameter
  - configuring SSL 64
- SSLClientKeystash configuration parameter 78
- SSLClientKeystash connection string parameter 78
- SSLClientKeystoredb connection string parameter 78
- Statement Value Data field 130
- Statement Value Index field 130
- Statement Value Type field 130
- static SQL
  - EXECUTE privilege 56
- switching
  - user IDs 151, 157
- SYSADM (system administration) authority
  - details 29
  - Windows 369
- sysadm\_group configuration parameter
  - Windows 369
- SYSADMIN events 354
- SYSSTAT views
  - security issues 217
- SYSCTRL (system control) authority
  - details 30
- SYSDEFAULTADMWORKLOAD workload 47
- SYSDEFAULTUSERWORKLOAD workload 47
- SYSMAINT (system maintenance) authority
  - details 31
- SYSMON (system monitor) authority
  - details 32
- SYSPROC.AUDIT\_ARCHIVE stored procedure 122, 128
- SYSPROC.AUDIT\_DELIM\_EXTRACT stored procedure 122, 128
- SYSPROC.AUDIT\_LIST\_LOGS stored procedure 128
- system authorization IDs 47

- system catalogs
  - listing privileges 217
  - retrieving
    - authorization names with privileges 218
    - names with DBADM authority 219
    - names with table access authority 219
    - privileges granted to names 220
- security 220

## T

- table spaces
  - privileges 42
- tables
  - access control 58
  - audit policies 118
  - inserting into LBAC-protected 206
  - LBAC effect on reading 203
  - privileges 54
  - protecting with LBAC 183, 201
  - removing LBAC protection 215
  - retrieving information
    - authorized names 219
    - revoking privileges 54
- TLS (transport layer security) 83
- TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
  - cipher suite 64, 87
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
  - cipher suite 64, 87
- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA
  - cipher suite 64, 87
- Transport Layer Security (TLS)
  - overview 83
- troubleshooting
  - security plug-ins 233
- trust relationships
  - Windows 364
- trusted clients
  - CLIENT authentication type 7
- trusted connections
  - establishing explicit trusted connections 151
  - overview 153
- trusted contexts
  - audit policies 118
  - overview 153
  - problem determination 159
  - role membership inheritance 156

## U

- UDFs
  - non-fenced 32
- UPDATE privilege 43
- updates
  - effects of LBAC on 208
- USAGE privilege
  - details 46
  - workloads 47
- user IDs
  - LDAP 248
  - selecting 4
  - switching 157

- user IDs (*continued*)
  - two-part 231
- user names
  - Windows restrictions 363
- USER\_BASEDN 245
- USER\_OBJECTCLASS 245
- USERID\_ATTRIBUTE 245

## V

- VALIDATE events 354
- verify 114
- verifying the database backup image is encrypted 114
- views
  - access privileges examples 58
  - column access 58
  - privileges information 220
  - row access 58
  - table access control 58

## W

- Windows
  - extended security 370
  - LocalSystem account (LSA)
    - support 370
  - scenarios
    - client authentication 362
    - server authentication 362
  - user accounts
    - access tokens 365
- windows 7
  - User Access Control (UAC)
    - feature 374
- WITH ADMIN OPTION clause
  - delegating role maintenance 148
- WITH DATA option
  - details 130
- WLMADM (workload administration)
  - authority
    - details 39
    - overview 32
- write-down
  - details 195
- write-up
  - details 195

## X

- XQuery
  - dynamic 56
  - static 56







Printed in USA