

IBM DB2 10.1
for Linux, UNIX, and Windows

Database Security Guide

Updated January, 2013



IBM DB2 10.1
for Linux, UNIX, and Windows

Database Security Guide

Updated January, 2013



Note

Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 359.

Edition Notice

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

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

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

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

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

© Copyright IBM Corporation 2013.

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

Contents

About this book vii

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	25
Instance level authorities	29
Database authorities	32
Privileges	40
Authorization IDs in different contexts	45
Default privileges granted on creating a database	47
Granting and revoking access	49
Controlling access for database administrators (DBAs)	55
Gaining access to data through indirect means	56
Data encryption	59
Configuring Secure Sockets Layer (SSL) support in a DB2 instance	60
IBM Database Encryption Expert for encryption of data at rest	86
Database encryption using AIX encrypted file system (EFS)	88
Auditing DB2 activities	91
Introduction to the DB2 audit facility	91
Audit facility management	112
Security model for the db2cluster command	116

Chapter 2. Roles 119

Creating and granting membership in roles	120
Role hierarchies	122
Effect of revoking privileges from roles	122
Delegating role maintenance by using the WITH ADMIN OPTION clause	124
Roles compared to groups	124
Using roles after migrating from IBM Informix Dynamic Server	126

Chapter 3. Using trusted contexts and trusted connections 127

Trusted contexts and trusted connections	129
Role membership inheritance through a trusted context	132

Rules for switching the user ID on an explicit trusted connection	133
Trusted context problem determination	135

Chapter 4. Row and column access control (RCAC) overview 137

Row and column access control (RCAC) rules	138
SQL statements for managing RCAC rules	138
Built-in functions for managing RCAC permissions and masks	138
Scenario: ExampleHMO using row and column access control	138
Scenario: ExampleHMO using row and column access control - Security policies	139
Scenario: ExampleHMO using row and column access control - Database users and roles	139
Scenario: ExampleHMO using row and column access control - Database tables	140
Scenario: ExampleHMO using row and column access control - Security administration	142
Scenario: ExampleHMO using row and column access control - Row permissions	143
Scenario: ExampleHMO using row and column access control - Column masks	144
Scenario: ExampleHMO using row and column access control - Data insertion	145
Scenario: ExampleHMO using row and column access control - Data updates	146
Scenario: ExampleHMO using row and column access control - Data queries	146
Scenario: ExampleHMO using row and column access control - View creation	148
Scenario: ExampleHMO using row and column access control - Secure functions	149
Scenario: ExampleHMO using row and column access control - Secure triggers	151
Scenario: ExampleHMO using row and column access control - Revoke authority	152
Scenario: ExampleBANK using row and column access control	152
Scenario: ExampleBANK using row and column access control - Security policies	153
Scenario: ExampleBANK using row and column access control - Database users and roles	153
Scenario: ExampleBANK using row and column access control - Database tables	154
Scenario: ExampleBANK using row and column access control - Row permissions	155
Scenario: ExampleBANK using row and column access control - Column masks	156
Scenario: ExampleBANK using row and column access control - Data queries	156

Chapter 5. Label-based access control (LBAC) 159

LBAC security policies	161
LBAC security label components overview	162
LBAC security label component type: SET	163
LBAC security label component type: ARRAY	163
LBAC security label component type: TREE	164
LBAC security labels	167
Format for security label values	169
How LBAC security labels are compared	170
LBAC rule sets overview	171
LBAC rule set: DB2LBACRULES	171
LBAC rule exemptions	175
Built-in functions for managing LBAC security labels	176
Protection of data using LBAC	177
Reading of LBAC protected data	179
Inserting of LBAC protected data	181
Updating of LBAC protected data	184
Deleting or dropping of LBAC protected data	188
Removal of LBAC protection from data	191

Chapter 6. Using the system catalog for security information 193

Retrieving authorization names with granted privileges	194
Retrieving all names with DBADM authority	195
Retrieving names authorized to access a table	195
Retrieving all privileges granted to users	195
Securing the system catalog view	196

Chapter 7. Firewall support 199

Screening router firewalls	199
Application proxy firewalls	199
Circuit level firewalls	199
Stateful multi-layer inspection (SMLI) firewalls	200

Chapter 8. Security plug-ins 201

Security plug-in library locations	205
Security plug-in naming conventions	206
Security plug-in support for two-part user IDs	207
Security plug-in API versioning	209
32-bit and 64-bit considerations for security plug-ins	209
Security plug-in problem determination	209
Enabling plug-ins	211
Deploying a group retrieval plug-in	211
Deploying a user ID/password plug-in	211
Deploying a GSS-API plug-in	212
Deploying a Kerberos plug-in	214
LDAP-based authentication and group lookup support	215
Configuring transparent LDAP for authentication and group lookup (AIX)	217
Configuring transparent LDAP for authentication and group lookup (Linux)	220
Configuring transparent LDAP for authentication and group lookup (HP-UX)	221
Configuring transparent LDAP for authentication and group lookup (Solaris)	223
Configuring the LDAP plug-in modules	225
Enabling the LDAP plug-in modules	227

Connecting with an LDAP user ID	228
Considerations for group lookup	229
Troubleshooting authenticating LDAP users or retrieving groups	230
Writing security plug-ins	230
How DB2 loads security plug-ins	230
Restrictions for developing security plug-in libraries	232
Restrictions on security plug-ins	234
Return codes for security plug-ins	235
Error message handling for security plug-ins	238
Calling sequences for the security plug-in APIs	239

Chapter 9. Security plug-in APIs 243

APIs for group retrieval plug-ins	244
db2secDoesGroupExist API - Check if group exists	245
db2secFreeErrorMsg API - Free error message memory	246
db2secFreeGroupListMemory API - Free group list memory	246
db2secGetGroupsForUser API - Get list of groups for user	247
db2secGroupPluginInit API - Initialize group plug-in	250
db2secPluginTerm - Clean up group plug-in resources	251
APIs for user ID/password authentication plug-ins	251
db2secClientAuthPluginInit API - Initialize client authentication plug-in	257
db2secClientAuthPluginTerm API - Clean up client authentication plug-in resources	258
db2secDoesAuthIDExist - Check if authentication ID exists	259
db2secFreeInitInfo API - Clean up resources held by the db2secGenerateInitialCred	259
db2secFreeToken API - Free memory held by token	260
db2secGenerateInitialCred API - Generate initial credentials	260
db2secGetAuthIDs API - Get authentication IDs	262
db2secGetDefaultLoginContext API - Get default login context	264
db2secProcessServerPrincipalName API - Process service principal name returned from server	265
db2secRemapUserid API - Remap user ID and password	266
db2secServerAuthPluginInit - Initialize server authentication plug-in	267
db2secServerAuthPluginTerm API - Clean up server authentication plug-in resources	270
db2secValidatePassword API - Validate password	270
Required APIs and definitions for GSS-API authentication plug-ins	273
Restrictions for GSS-API authentication plug-ins	274

Chapter 10. Communication buffer exit libraries 275

Communication buffer exit library deployment	276
Communication buffer exit library location	276
Communication buffer exit library naming conventions and permissions	276
Enabling communication buffer exit libraries outside of DB2 pureScale environments	277
Enabling communication buffer exit libraries in DB2 pureScale environments	278
Communication buffer exit library problem determination	278
Communication buffer exit library development	279
How a communication buffer exit library is loaded.	279
Communication buffer exit library APIs	280
Communication buffer exit library functions structure	288
Communication buffer exit library information structure	289
Communication buffer exit library buffer structure	290
Communication buffer exit library control over connections	290
Communication buffer exit library API versions	290
Communication buffer exit library error handling and return codes	291
Communication buffer exit library development restrictions	291
Communication buffer exit library API calling sequences	293

Chapter 11. Audit facility record layouts 297

Audit record object types	297
Audit record layout for AUDIT events	298
Audit record layout for CHECKING events	302
CHECKING access approval reasons	303
CHECKING access attempted types	305
Audit record layout for OBJMAINT events	308
Audit record layout for SECMAINT events	310
SECMAINT privileges or authorities	314
Audit record layout for SYSADMIN events	317
Audit record layout for VALIDATE events	319

Audit record layout for CONTEXT events	320
Audit record layout for EXECUTE events	322
Audit events	327

Chapter 12. Working with operating system security 333

DB2 and Windows security.	333
Authentication scenarios.	334
Support for global groups (Windows)	335
User authentication and group information with DB2 on Windows	335
Defining which users hold SYSADM authority (Windows)	341
Windows LocalSystem account support	342
Extended Windows security using the DB2ADMNS and DB2USERS groups	342
Considerations for Windows 2008 and Windows Vista or higher: User Access Control feature	346
DB2 and UNIX security	347
DB2 and Linux security	347
Change password support (Linux)	347
Deploying a change password plug-in (Linux)	347

Appendix A. Overview of the DB2 technical information 349

DB2 technical library in hardcopy or PDF format	349
Displaying SQL state help from the command line processor.	352
Accessing different versions of the DB2 Information Center	352
Updating the DB2 Information Center installed on your computer or intranet server.	352
Manually updating the DB2 Information Center installed on your computer or intranet server	354
DB2 tutorials	356
DB2 troubleshooting information.	356
Terms and conditions.	356

Appendix B. Notices 359

Index 363

About this book

The *Database Security Guide* describes how to use DB2® security features to implement and manage the level of security you require for your database installation.

The *Database Security Guide* provides detailed information about:

- Managing the authentication of users who can access DB2 databases
- Setting up authorization to control user access to database objects and data

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), access control (ACCESSCTRL), 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.

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 manager operations. Database-specific 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/sqlldbidir 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 sqlbdir 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 sqlbdir/
```

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
NO	CLIENT	SERVER	SERVER	CLIENT	CLIENT	CLIENT	CLIENT
NO	SERVER	SERVER	SERVER	CLIENT	SERVER	CLIENT	SERVER
DRDAONLY	CLIENT	SERVER	SERVER	SERVER	SERVER	CLIENT	CLIENT

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

		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)
trust_allclnts	trust_clntauth	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 `sqliib/security32/plugin/IBM/client` and `sqliib/security32/plugin/IBM/server` directories
- On UNIX and Linux 64-bit operating systems: the `sqliib/security64/plugin/IBM/client` and `sqliib/security64/plugin/IBM/server` directories
- On Windows operating systems: the `sqliib\security\plugin\IBM\client` and `sqliib\security\plugin\IBM\server` directories

The source code for the UNIX and Linux plug-in, `IBMkrb5.C`, is available in the `sqliib/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 214.
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 **svrcon_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 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** (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 (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 INSERT privilege on 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
- SYSMAINT -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 SYSMAINT and SYSMON authority as well.

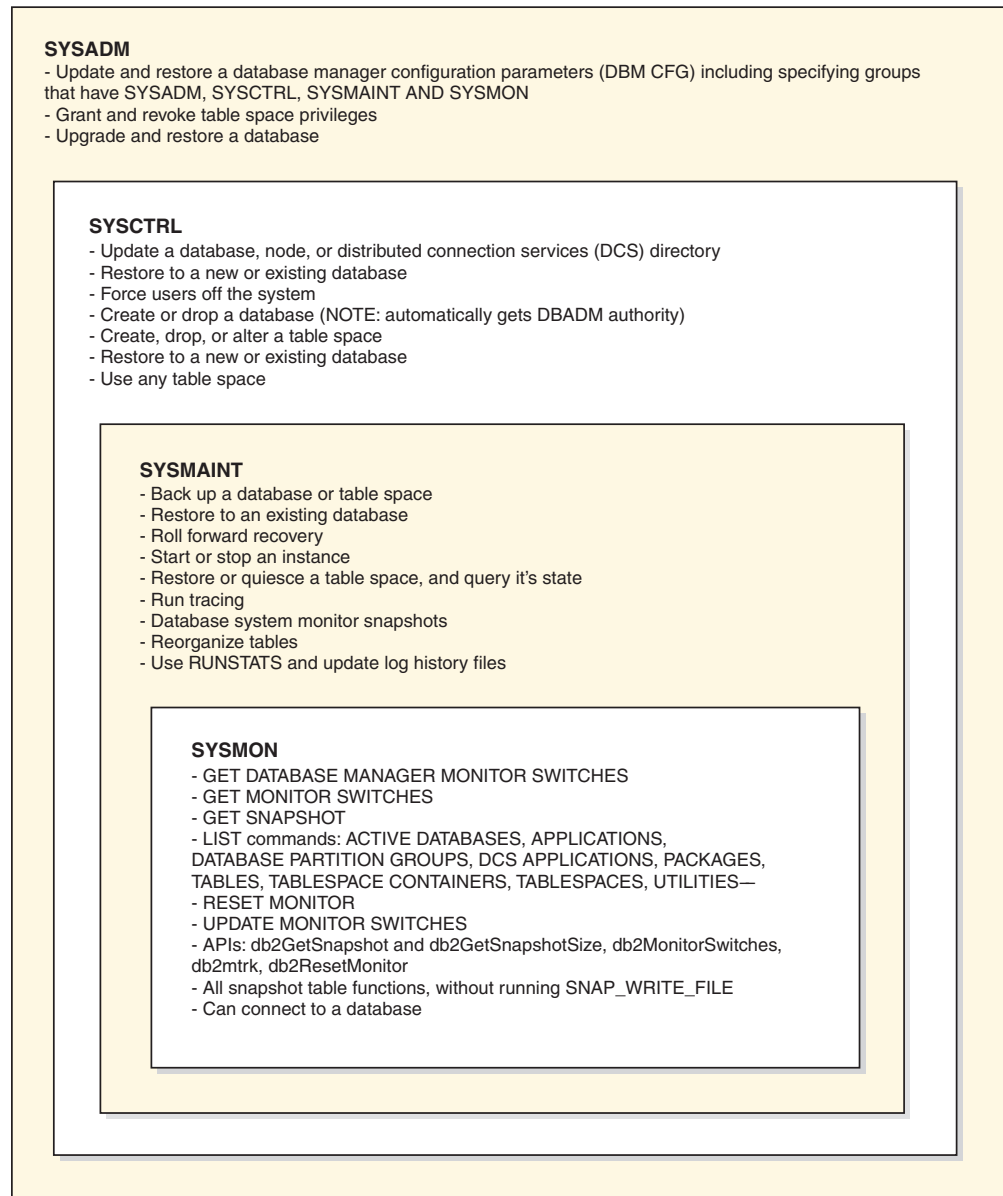


Figure 1. Instance-level authorities

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.

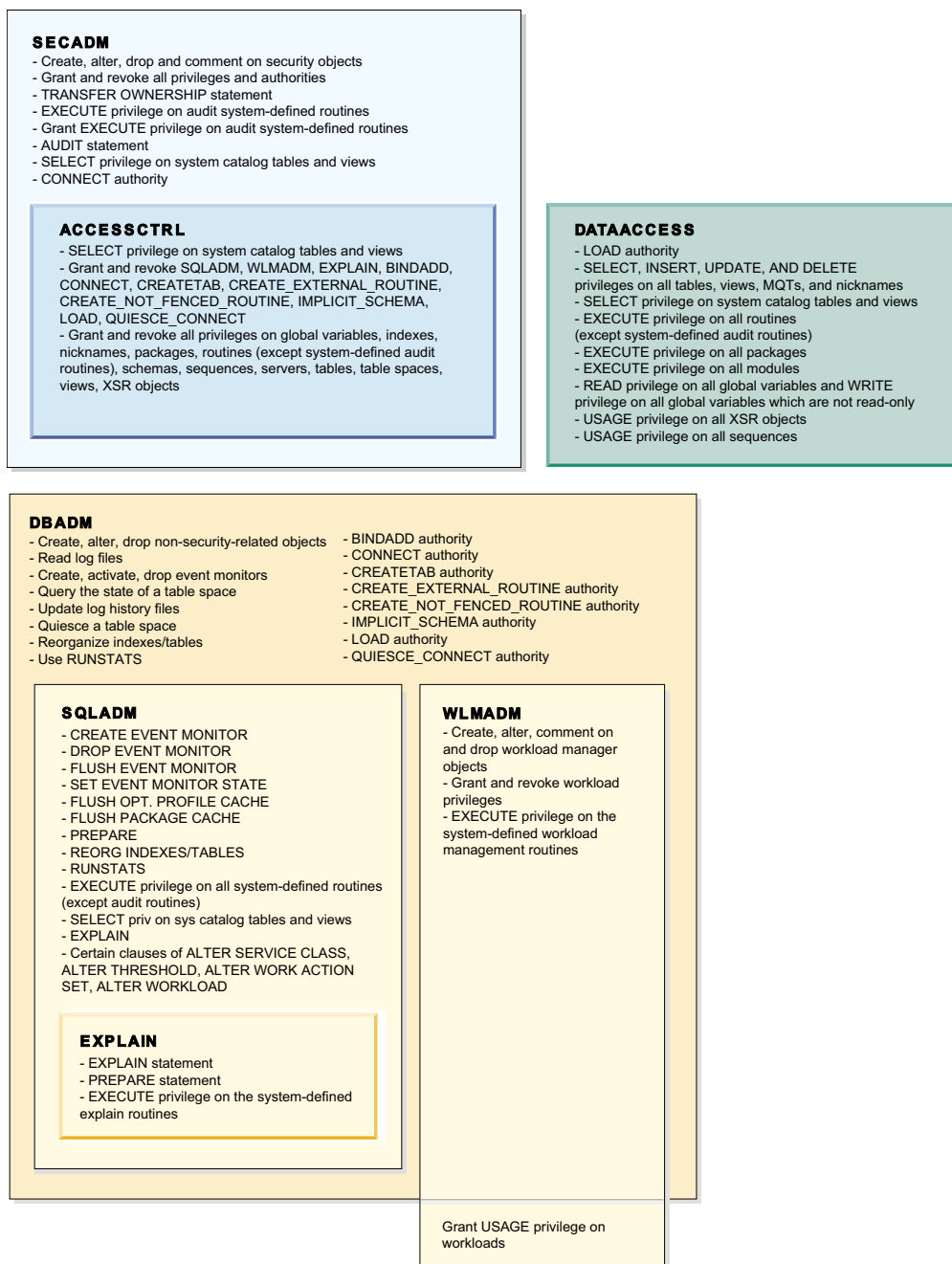


Figure 2. Database-level authorities

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
- Force users off the system
- 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
- 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.

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).

- 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 required to create objects and issue database commands. In addition, users with DBADM authority have SELECT privilege on the system catalog tables and views, and can execute all built-in DB2 routines, except audit routines.

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
- 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)
- 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)

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.

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.

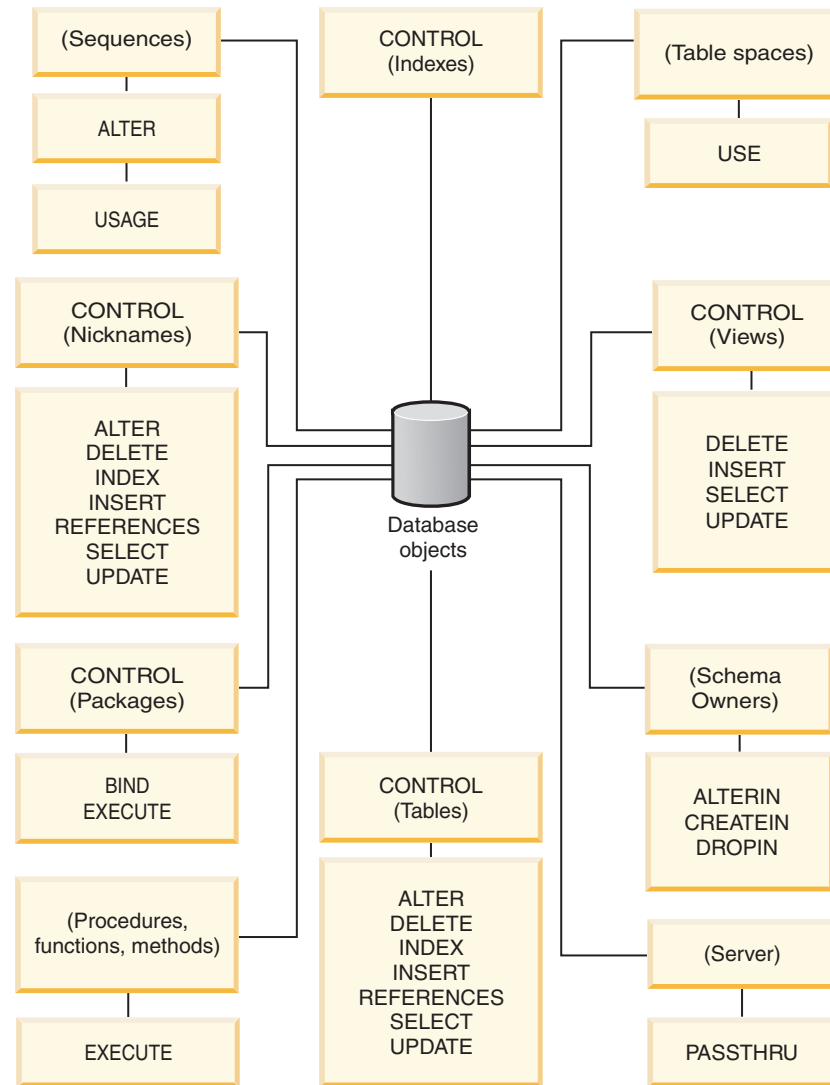


Figure 3. Object Privileges

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.

By default, at database creation time the USE privilege for table space USERSPACE1 is granted to PUBLIC, although this privilege can be revoked.

The USE privilege cannot be used with SYSCATSPACE or any 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.

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

Value of DYNAMICRULES option for issuing the package	Authorization ID used
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
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 all functions and procedures in schema SYSPROC (except audit routines)
- 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.

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 rebind the package. If this rebind attempt fails, an error occurs (SQLCODE -727). In this case, the packages must be explicitly rebound by a user with:

- Authority to rebind 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

NAME	LOCATION
Jones	New York
Hanes	Boston
Rothman	Boston
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 the encryption and decryption built-in functions **ENCRYPT**, **DECRYPT_BIN**, **DECRYPT_CHAR**, and **GETHINT** to encrypt your data within database tables.
- You can use IBM Database Encryption Expert to encrypt the underlying operating system data and backup files.
- If you are running a DB2 Enterprise Server Edition 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).

Using the **ENCRYPT**, **DECRYPT_BIN**, **DECRYPT_CHAR**, and **GETHINT** functions

The **ENCRYPT** built-in function encrypts data using a password-based encryption method. These functions also allow you to encapsulate a password hint. The password hint is embedded in the encrypted data. Once encrypted, the only way to decrypt the data is by using the correct password. Developers that choose to use these functions should plan for the management of forgotten passwords and unusable data.

The result of the **ENCRYPT** functions is **VARCHAR FOR BIT DATA** (with a limit of 32631).

Only **CHAR**, **VARCHAR**, and **FOR BIT DATA** can be encrypted.

The **DECRYPT_BIN** and **DECRYPT_CHAR** functions decrypt data using password-based decryption.

DECRYPT_BIN always returns VARCHAR FOR BIT DATA while DECRYPT_CHAR always returns VARCHAR. Since the first argument may be CHAR FOR BIT DATA or VARCHAR FOR BIT DATA, there are cases where the result is not the same as the first argument.

The length of the result depends on the bytes to the next 8 byte boundary. The length of the result could be the length of the data argument plus 40 plus the number of bytes to the next 8 byte boundary when the optional hint parameter is specified. Or, the length of the result could be the length of the data argument plus 8 plus the number of bytes to the next 8 byte boundary when the optional hint parameter is not specified.

The GETHINT function returns an encapsulated password hint. A password hint is a phrase that will help data owners remember passwords. For example, the word "Ocean" can be used as a hint to remember the password "Pacific".

The password that is used to encrypt the data is determined in one of two ways:

- Password Argument. The password is a string that is explicitly passed when the ENCRYPT function is invoked. The data is encrypted and decrypted with the given password.
- Encryption password special register. The SET ENCRYPTION PASSWORD statement encrypts the password value and sends the encrypted password to the database manager to store in a special register. ENCRYPT, DECRYPT_BIN and DECRYPT_CHAR functions invoked without a password parameter use the value in the ENCRYPTION PASSWORD special register. The ENCRYPTION PASSWORD special register is only stored in encrypted form.

The initial or default value for the special register is an empty string.

Valid lengths for passwords are between 6 and 127 inclusive. Valid lengths for hints are between 0 and 32 inclusive.

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 using an 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

Before configuring 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 High Availability Disaster Recovery (HADR) systems

SSL is supported between clients and the HADR primary server. Clients connecting to the HADR primary server using SSL are able to reroute to the HADR standby database using SSL. However, SSL is not supported between the HADR primary and standby 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 **gskcapi** command, as described in the *GSKCapiCmd User's Guide*. The path for the command is `sqllib/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 `sqllib/lib64/gskit`.

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

```
gsk8capi64 -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:

```
gsk8capi64 -cert -create -db "mydbserver.kdb" -pw "myServerPassw0rdpw0"
-label "myselfsigned" -dn "CN=myhost.mycompany.com,O=myOrganization,
OU=myOrganizationUnit,L=myLocation,ST=ON,C=CA"
```

- 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`:

```
gsk8capi64 -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/sqllib/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/sqllib/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 73 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"
```

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 : 1024
version : X509 V3
serial : 96c2db8fa769a09d
issue:CN=myhost.mycompany.com,O=myOrganization,OU=myOrganizationUnit,
L=myLocation,ST=ON,C=CA
subject:CN=myhost.mycompany.com,O=myOrganization,OU=myOrganizationUnit,
L=myLocation,ST=ON,C=CA
not before : Tuesday, 24 February 2009 17:11:50 PM
not after : Thursday, 25 February 2010 17:11:50 PM
public Key
 30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
 05 00 03 81 8D 00 30 81 89 02 81 81 00 B6 B8 DC
 79 69 62 C9 A5 C1 5C 38 31 53 AB 27 BE 63 C0 DB
 DE C6 BC 2E A4 0D 37 45 95 22 0E 83 32 FE 67 A9
 2F D7 51 FF 40 A3 76 68 B9 E3 34 CB 7D 4A D8 38
 CA B1 6B 32 66 74 8F E2 B8 DA 8F D0 F3 62 04 BE
 C4 FE 80 2A D0 FF 27 72 37 9A 36 1D DB D3 A1 33
 A1 A6 48 33 E9 64 B9 9B 6B DB 08 60 7D 5E 0E 20
 0A 26 AA 62 3A DF D3 78 56 DC 15 DE 9F 0B 91 DD
 3B 1B 2B E2 82 FA 24 FF 81 A3 F7 3F C1 02 03 01
 00 01
public key type : RSA : 1.2.840.113549.1.1.1
finger print : SHA1 :
 2D C1 93 F8 AC A0 8F E2 C2 05 D8 23 D7 5D 87 E6
 82 3C 47 EC
signature algorithm : SHA1WithRSASignature : 1.2.840.113549.1.1.5
value
 0E 80 24 98 F6 6E 89 43 76 57 76 7F 82 95 18 6A
 43 A5 81 EC F4 82 1F 1F F2 3F E5 61 67 48 C0 59
 94 17 8E 8F DE 4F 7C 35 0C 5D A7 98 73 2A 34 7D
 1E BA 53 78 A5 E4 31 45 D1 08 86 BE 5E 57 C6 9D
 B5 E7 A7 01 3F 54 01 5E 8F 8B 2F 66 19 24 1E A4
 94 58 B0 D4 40 95 AB 98 C2 EF 1C 5C 4A 29 48 EC
 8C C0 A2 B1 AC 2A E9 3C 14 E5 77 B2 A6 55 A8 21
 CB 59 81 86 79 F0 46 35 F8 FC 99 2D EC D4 B9 EB
Trusted : 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"

```

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 : 1024
subject : Common Name (CN):
  Type : 2.5.4.3
  Value: myhost.mycompany.com
Organization (O):
  Type : 2.5.4.10
  Value: myOrganization
Organizational Unit (OU):
  Type : 2.5.4.11
  Value: myOrganizationUnit
Locality (L):
  Type : 2.5.6.3
  Value: myLocation
State (ST):
  Type : ?
  Value: Ontario
Country or region (C):
  Type : 2.5.4.6
  Value: CA
public Key
30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
05 00 03 81 8D 00 30 81 89 02 81 81 00 9C B4 62
3C 89 02 4E B0 D8 EA 0B B8 CC 70 63 4A 59 1F 0F
FD 98 9A 1A 39 94 E3 43 C1 63 7A CD 21 47 57 D9
86 6F 11 B8 91 08 AC E3 E2 21 32 FE 43 1F 07 C9
F5 40 6B 3E 4D 56 35 05 62 D6 78 0B E3 97 28 F7
27 31 A4 05 BE F2 3A 44 6B D8 D1 FF 1E DA 59 63
E6 49 52 39 45 9C 1E 8E CC DA A1 D9 0F 3A 96 09
66 5C 89 23 2E EE 31 65 8D 87 8E B9 61 C6 69 BC
A5 DB EB 03 16 E6 33 85 14 68 BC DD F1 02 03 01
00 01
finger print :
e0dcde10ded3a46a53c0190e84cc994e
5d7e4bad
attributes
signature algorithm1.2.840.113549.1.1.5
value
4F 06 B4 E3 1F 00 B4 81 90 CC A2 99 4A 02 68 D0
84 B5 7F 33 0B F0 04 D5 7D 4C 5C CB 5C D3 37 77
E2 6D 10 17 50 19 D0 7F 61 C7 C8 54 7B DB CD 6F
47 9F 7E 7E 5A CC 64 20 85 95 A8 5E C7 7D FB F4
8A 7F 4B 74 6F 0A C6 EF 09 E7 0A 15 17 CC 1D D2
5D ED 02 A1 BE 1D FC F2 65 EB 0D E2 93 BC 88 4C
4C 73 76 16 9F 1B 12 3B 7A 01 CF E0 63 97 E8 38
02 FB 47 EE F2 17 54 66 4D F7 7F 9E 13 DA 76 A2

```

To display the certificate request file:

```
$ cat mycertRequestNew
```

```

-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBrjCCARcCAQAwbjELMAkGA1UEBhMCQ0ExEDA0BgNVBAgTB09udGFyaW8xEDAO
BgNVBAcTB01hcmt0YW0xDDAKBgNVBAoTA01CTTEMMAoGA1UECxMDREIyMR8wHQYD
VQQExZnaWxlcmEudG9yb2xhYi5pYm0uY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GN
ADCBiQKBgQCctGI8iQJ0sNjqC7jMcGNKWR8P/ZiaGjmU40PBY3rNIUdX2YZvEbiR
CKzj4iEy/kMfB8n1QGs+TVY1BWLWeAvjlyj3JzGkBb7y0kRr2NH/HtpZY+ZJUj1F
nB60zNqh2Q861glmXikjLu4xZY2Hjr1hxmm8pdvrAxbmM4UUAzLzd8QIDAQABoaAw
DQYJKoZIhvcNAQEFBQADgYEATwa04x8AtIGQzKKZSgJo0IS1fzML8ATVfUxcy1zT
N3fibRAXUBnQf2HHYFR7281vR59+flrMZCCF1ahex3379Ip/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 -cert -list all -db "mydbserver.kdb" -pw "mydbserverpw0"

certificates found
* default, - personal, ! trusted
-!      mycert
!      trialIntermediateCACert
!      trialRootCACert
-!      myselfsigned
db2 update dbm cfg using SSL_SVR_LABEL mycert

```

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> /sql1ib/lib32/gskit	Include <i>\$INSTHOME</i> /sql1ib/lib32/gskit in the <i>LD_LIBRARY_PATH</i> , <i>LIBPATH</i> , or <i>SHLIB_PATH</i> environment variable
64-bit	Linux and UNIX 64-bit	<i>\$INSTHOME</i> /sql1ib/lib64/gskit	Include <i>\$INSTHOME</i> /sql1ib/lib64/gskit in the <i>LD_LIBRARY_PATH</i> , <i>LIBPATH</i> , or <i>SHLIB_PATH</i> environment variable
32-bit	Windows 64-bit	C:\Program Files (x86)\IBM\GSK8\lib	Include C:\Program Files (x86)\IBM\GSK8\lib in <i>PATH</i> environment variable
64-bit	Windows 64-bit	C:\Program Files\IBM\GSK8\lib64	Include C:\Program Files\IBM\GSK8\lib64 in <i>PATH</i> 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%
```

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.

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 60 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 TCP/IP NODE** command. Issue the **CATALOG TCP/IP 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 TCP/IP 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 (`ssl_client_keystoredb` and `ssl_client_keystash`) 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; Ssl_client_keystoredb=/home/test1/keystore/clientstore.kdb; Ssl_client_keystash=/home/test1/keystore/clientstore.sth;"
```

In this case, because `Security=ssl` is specified, the `ssl_client_keystoredb` and `ssl_client_keystash` 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 `ssl_client_keystoredb` and `ssl_client_keystash` 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
SSL_client_keystoredb=/home/test1/keystore/clientstore.kdb
SSL_client_keystash=/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
SSL_client_keystoredb=/home/test1/keystore/clientstore.kdb
SSL_client_keystash=/home/test1/keystore/clientstore.sth
```

In these cases, because `Security=ssl` is specified, if the `ssl_client_keystoredb` and `ssl_client_keystash` 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.

Certificate-based authentication

Starting in DB2 Version 9.7 Fix Pack 6, a new authentication type 'CERTIFICATE' has been introduced in the `db2dsdriver.cfg` authentication parameter as shown in the syntax below:

```
<parameter name="Authentication"
value="CERTIFICATE | SERVER | SERVER_ENCRYPT | SERVER_ENCRYPT_AES | DATA_ENCRYPT | KERBEROS | GSSPLUGIN"/>
```

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 CERTIFICATE 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`.

Starting in DB2 Version 9.7 Fix Pack 6, a new keyword **SSLClientKeyStoreDBPassword** is introduced to set the password for the keystore db specified through **SSLClientKeystoredb** keyword. 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.

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 connectString = "Server=myhost:50001;Database=sampldb;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(connectString);  
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`.

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.

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

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
- 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 for Linux, UNIX, and Windows bundles vendor software that requires GSKit, or when vendor software that requires GSKit bundles DB2 for Linux, UNIX, and Windows, certain rules must be followed.

Library rule

When DB2 for Linux, UNIX, and Windows bundles vendor software that requires GSKit, the vendor software provides libraries that DB2 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows, 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 could load the GSKit libraries from the same location.

On AIX, a process can set LIBPATH or RPATH of the executable 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 executable. 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 needs to use global GSKit in server instances, or it needs to use its own local GSKit in client or server instances, it may use db2chglbpath to change the RPATH.

Symbolic link approach and restriction

When you install DB2 for Linux, UNIX, and Windows on UNIX and Linux platforms, local GSKit libraries are installed as well. Those libraries are located 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. These libraries might be either local GSKit libraries or global GSKit libraries depending on the product. When DB2 for Linux, UNIX, and Windows and another IBM product that includes GSKit libraries are both installed on the same machine, 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 in situations where more than single installation of GSKit exists on the same machine, the symbolic link approach can be used. During an initial DB2 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows and change the symbolic link from the default directory noted 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 for Linux, UNIX, and Windows, the symbolic link is preserved. If the newly installed copy has a symbolic link that points to the default location, the symbolic link 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 install 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 located in the path of the DB2 for Linux, UNIX, and Windows install copy. For example, if there are two or more instances created for any DB2 copy, the symbolic link changes affect all the instances.

The GSKit version included with DB2 for Linux, UNIX, and Windows is 8.0.14.14, 8.0.15.1 on Solaris x64.

Examples

DB2 for Linux, UNIX, and Windows bundles the LDAP client. The DB2 for Linux, UNIX, and Windows processes follow the process rule. To follow the process rule, the environment search path via RPATH is set to its local copy of GSKit. LDAP client libraries loads 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 for Linux, UNIX, and Windows. LDAP processes follows 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.

GSKit return codes

Some DB2 database manager messages might display a return code from the IBM Global Security Kit (GSKit).

General GSKit return codes

Table 2. GSKit general return codes

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x00000000	0	GSK_OK	The task completed successfully. Issued by every function call that completes successfully.
0x00000001	1	GSK_INVALID_HANDLE	The environment or SSL handle is not valid. The specified handle was not the result of a successful open function call.
0x00000002	2	GSK_API_NOT_AVAILABLE	The dynamic link library (DLL) has been unloaded and is not available. (Windows only.)
0x00000003	3	GSK_INTERNAL_ERROR	Internal error. Report this error to service.
0x00000004	4	GSK_INSUFFICIENT_STORAGE	Insufficient memory is available to perform the operation.
0x00000005	5	GSK_INVALID_STATE	The handle is in an invalid state for operation, such as performing an init operation on a handle twice.
0x00000006	6	GSK_KEY_LABEL_NOT_FOUND	Specified key label not found in key file.
0x00000007	7	GSK_CERTIFICATE_NOT_AVAILABLE	Certificate not received from partner.
0x00000008	8	GSK_ERROR_CERT_VALIDATION	Certificate validation error.
0x00000009	9	GSK_ERROR_CRYPTO	Error processing cryptography.
0x0000000a	10	GSK_ERROR_ASN	Error validating ASN fields in certificate.
0x0000000b	11	GSK_ERROR_LDAP	Error connecting to LDAP server.

Table 2. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x0000000c	12	GSK_ERROR_UNKNOWN_ERROR	Internal error. Report this error to service.
0x00000065	101	GSK_OPEN_CIPHER_ERROR	Internal error. Report this error to service.
0x00000066	102	GSK_KEYFILE_IO_ERROR	I/O error reading the key file.
0x00000067	103	GSK_KEYFILE_INVALID_FORMAT	The key file has an invalid internal format. Re-create the key file.
0x00000068	104	GSK_KEYFILE_DUPLICATE_KEY	The key file has two entries with the same key. Use the iKeyman utility to remove the duplicate key.
0x00000069	105	GSK_KEYFILE_DUPLICATE_LABEL	The key file has two entries with the same label. Use the iKeyman utility to remove the duplicate label.
0x0000006a	106	GSK_BAD_FORMAT_OR_INVALID_PASSWORD	The key file password is used as an integrity check. Either the keyfile has become corrupted or the password ID is incorrect.
0x0000006b	107	GSK_KEYFILE_CERT_EXPIRED	The default key in the key file has an expired certificate. Use the iKeyman utility to remove certificates that are expired.
0x0000006c	108	GSK_ERROR_LOAD_GSKLIB	An error occurred loading one of the GSKit dynamic link libraries. Be sure GSKit was installed correctly.
0x0000006d	109	GSK_PENDING_CLOSE_ERROR	Indicates that a connection is trying to be made in a GSKit environment after the GSK_ENVIRONMENT_CLOSE_OPTIONS has been set to GSK_DELAYED_ENVIRONMENT_CLOSE and gsk_environment_close() function has been called.
0x000000c9	201	GSK_NO_KEYFILE_PASSWORD	Neither the password nor the stash-file name was specified, so the key file could not be initialized.
0x000000ca	202	GSK_KEYRING_OPEN_ERROR	Unable to open the key file or the Microsoft Certificate Store. The path was specified incorrectly, or the file permissions did not allow the file to be opened, or the file format is incorrect.
0x000000cb	203	GSK_RSA_TEMP_KEY_PAIR	Unable to generate a temporary key pair. Report this error to service.

Table 2. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x000000cc	204	GSK_ERROR_LDAP_NO_SUCH_OBJECT	A User Name object was specified that is not found.
0x000000cd	205	GSK_ERROR_LDAP_INVALID_CREDENTIALS	A Password used for an LDAP query is not correct.
0x000000ce	206	GSK_ERROR_BAD_INDEX	An index into the Fail Over list of LDAP servers was not correct.
0x000000cd	207	GSK_ERROR_FIPS_NOT_SUPPORTED	Attempt to put GSKit into FIPS mode has failed.
0x0000012d	301	GSK_CLOSE_FAILED	Indicates that the GSKit environment close request was not properly handled. This is most likely due to a <code>gsk_secure_socket*()</code> command being attempted after a <code>gsk_close_environment()</code> call.
0x00000191	401	GSK_ERROR_BAD_DATE	The system date was set to an invalid value.
0x00000192	402	GSK_ERROR_NO_CIPHERS	Neither SSLV2 nor SSLV3 is enabled.
0x00000193	403	GSK_ERROR_NO_CERTIFICATE	The required certificate was not received from partner.
0x00000194	404	GSK_ERROR_BAD_CERTIFICATE	The received certificate was formatted incorrectly.
0x00000195	405	GSK_ERROR_UNSUPPORTED_CERTIFICATE_TYPE	The received certificate type was not supported.
0x00000196	406	GSK_ERROR_IO	An I/O error occurred on a data read or write operation.
0x00000197	407	GSK_ERROR_BAD_KEYFILE_LABEL	The specified label in the key file could not be found.
0x00000198	408	GSK_ERROR_BAD_KEYFILE_PASSWORD	The specified key file password is incorrect. The key file could not be used. The key file also might be corrupt.
0x00000199	409	GSK_ERROR_BAD_KEY_LEN_FOR_EXPORT	In a restricted cryptography environment, the key size is too long to be supported.
0x0000019a	410	GSK_ERROR_BAD_MESSAGE	An incorrectly formatted SSL message was received from the partner.
0x0000019b	411	GSK_ERROR_BAD_MAC	The message authentication code (MAC) was not successfully verified.
0x0000019c	412	GSK_ERROR_UNSUPPORTED	Unsupported SSL protocol or unsupported certificate type.
0x0000019d	413	GSK_ERROR_BAD_CERT_SIG	The received certificate contained an incorrect signature.

Table 2. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x0000019e	414	GSK_ERROR_BAD_CERT	Incorrectly formatted certificate received from partner.
0x0000019f	415	GSK_ERROR_BAD_PEER	Invalid SSL protocol received from partner.
0x000001a0	416	GSK_ERROR_PERMISSION_DENIED	Report this internal error to service.
0x000001a1	417	GSK_ERROR_SELF_SIGNED	The self-signed certificate is not valid.
0x000001a2	418	GSK_ERROR_NO_READ_FUNCTION	The read operation failed. Report this internal error to service.
0x000001a3	419	GSK_ERROR_NO_WRITE_FUNCTION	The write operation failed. Report this internal error to service.
0x000001a4	420	GSK_ERROR_SOCKET_CLOSED	The partner closed the socket before the protocol completed.
0x000001a5	421	GSK_ERROR_BAD_V2_CIPHER	The specified V2 cipher is not valid.
0x000001a6	422	GSK_ERROR_BAD_V3_CIPHER	The specified V3 cipher is not valid.
0x000001a7	423	GSK_ERROR_BAD_SEC_TYPE	Report this internal error to service.
0x000001a8	424	GSK_ERROR_BAD_SEC_TYPE_COMBINATION	Report this internal error to service.
0x000001a9	425	GSK_ERROR_HANDLE_CREATION_FAILED	The handle could not be created. Report this internal error to service.
0x000001aa	426	GSK_ERROR_INITIALIZATION_FAILED	Initialization failed. Report this internal error to service.
0x000001ab	427	GSK_ERROR_LDAP_NOT_AVAILABLE	When validating a certificate, unable to access the specified LDAP directory.
0x000001ac	428	GSK_ERROR_NO_PRIVATE_KEY	The specified key did not contain a private key.
0x000001ad	429	GSK_ERROR_PKCS11_LIBRARY_NOTLOADED	A failed attempt was made to load the specified PKCS11 shared library.
0x000001ae	430	GSK_ERROR_PKCS11_TOKEN_LABELMISMATCH	The PKCS #11 driver failed to find the token specified by the caller.
0x000001af	431	GSK_ERROR_PKCS11_TOKEN_NOTPRESENT	A PKCS #11 token is not present in the slot.
0x000001b0	432	GSK_ERROR_PKCS11_TOKEN_BADPASSWORD	The password/pin to access the PKCS #11 token is invalid.
0x000001b1	433	GSK_ERROR_INVALID_V2_HEADER	The SSL header received was not a properly SSLV2 formatted header.

Table 2. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x000001b2	434	GSK_CSP_OPEN_ERROR	Unable to access the hardware-based cryptographic service provider (CSP). Either the given CSP name is not registered in the system or the specified CSP name is registered but the certificate store failed to open.
0x000001b3	435	GSK_CONFLICTING_ATTRIBUTE_SETTING	Attribute setting conflict between PKCS11, CMS key database, and Microsoft Crypto API.
0x000001b4	436	GSK_UNSUPPORTED_PLATFORM	The requested function is not supported on the platform that the application is running. For example, the Microsoft Crypto API is not supported on platforms other than Windows 2000.
0x000001b5	437	GSK_ERROR_INCORRECT_SESSION_TYPE	Incorrect value is returned from the reset session type callback function. Only GSKit GSK_SERVER_SESSION or GSK_SERVER_SESSION_WITH_CL_AUTH is allowed.
0x000001f5	501	GSK_INVALID_BUFFER_SIZE	The buffer size is negative or zero.
0x000001f6	502	GSK_WOULD_BLOCK	Used with non-blocking I/O.
0x00000259	601	GSK_ERROR_NOT_SSLV3	SSLV3 is required for reset_cipher, and the connection uses SSLV2.
0x0000025a	602	GSK_MISC_INVALID_ID	An invalid ID was specified for the gsk_secure_soc_misc function call.
0x000002bd	701	GSK_ATTRIBUTE_INVALID_ID	The function call has an invalid ID. This also might be caused by specifying an environment handle when a handle for a SSL connection should be used.
0x000002be	702	GSK_ATTRIBUTE_INVALID_LENGTH	The attribute has a negative length, which is invalid.
0x000002bf	703	GSK_ATTRIBUTE_INVALID_ENUMERATION	The enumeration value is invalid for the specified enumeration type.
0x000002c0	704	GSK_ATTRIBUTE_INVALID_SID_CACHE	Invalid parameter list for replacing the Session ID (SID) cache routines.
0x000002c1	705	GSK_ATTRIBUTE_INVALID_NUMERIC_VALUE	When setting a numeric attribute, the specified value is invalid for the specific attribute being set.
0x000002c2	706	GSK_CONFLICTING_VALIDATION_SETTING	Conflicting parameters have been set for additional certificate validation.

Table 2. GSKit general return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant	Explanation
0x000002c3	707	GSK_AES_UNSUPPORTED	The cipher specification included an AES cipher that is not supported on the system of execution.
0x000002c4	708	GSK_PEERID_LENGTH_ERROR	The length of the peer ID is incorrect. It must be less than or equal to 16 bytes.
0x000002c5	709	GSK_CIPHER_INVALID_WHEN_FIPS_MODE_OFF	Given cipher is not allowed when FIPS mode is off.
0x000002c6	710	GSK_CIPHER_INVALID_WHEN_FIPS_MODE_ON	No FIPS approved cipher have been selected in FIPS mode.
0x00000641	1601	GSK_TRACE_STARTED	The trace started successfully.
0x00000642	1602	GSK_TRACE_STOPPED	The trace stopped successfully.
0x00000643	1603	GSK_TRACE_NOT_STARTED	No trace file was previously started so it cannot be stopped.
0x00000644	1604	GSK_TRACE_ALREADY_STARTED	Trace file already started so it cannot be started again.
0x00000645	1605	GSK_TRACE_OPEN_FAILED	Trace file can not be opened. The first parameter of gsk_start_trace() must be a valid full path file name.

Key management return codes

Table 3. Key management return codes

Return code (hexadecimal)	Return code (decimal)	Constant
0x00000000	0	GSKKM_ERR_OK
0x00000000	0	GSKKM_ERR_SUCCESS
0x00000001	1	GSKKM_ERR_UNKNOWN
0x00000002	2	GSKKM_ERR_ASN
0x00000003	3	GSKKM_ERR_ASN_INITIALIZATION
0x00000004	4	GSKKM_ERR_ASN_PARAMETER
0x00000005	5	GSKKM_ERR_DATABASE
0x00000006	6	GSKKM_ERR_DATABASE_OPEN
0x00000007	7	GSKKM_ERR_DATABASE_RE_OPEN
0x00000008	8	GSKKM_ERR_DATABASE_CREATE
0x00000009	9	GSKKM_ERR_DATABASE_ALREADY_EXISTS
0x0000000a	10	GSKKM_ERR_DATABASE_DELETE
0x0000000b	11	GSKKM_ERR_DATABASE_NOT_OPENED
0x0000000c	12	GSKKM_ERR_DATABASE_READ
0x0000000d	13	GSKKM_ERR_DATABASE_WRITE
0x0000000e	14	GSKKM_ERR_DATABASE_VALIDATION

Table 3. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x0000000f	15	GSKKM_ERR_DATABASE_INVALID_VERSION
0x00000010	16	GSKKM_ERR_DATABASE_INVALID_PASSWORD
0x00000011	17	GSKKM_ERR_DATABASE_INVALID_FILE_TYPE
0x00000012	18	GSKKM_ERR_DATABASE_CORRUPTION
0x00000013	19	GSKKM_ERR_DATABASE_PASSWORD_CORRUPTION
0x00000014	20	GSKKM_ERR_DATABASE_KEY_INTEGRITY
0x00000015	21	GSKKM_ERR_DATABASE_DUPLICATE_KEY
0x00000016	22	GSKKM_ERR_DATABASE_DUPLICATE_KEY_RECORD_ID
0x00000017	23	GSKKM_ERR_DATABASE_DUPLICATE_KEY_LABEL
0x00000018	24	GSKKM_ERR_DATABASE_DUPLICATE_KEY_SIGNATURE
0x00000019	25	GSKKM_ERR_DATABASE_DUPLICATE_KEY_UNSIGNED_CERTIFICATE
0x0000001a	26	GSKKM_ERR_DATABASE_DUPLICATE_KEY_ISSUER_AND_SERIAL_NUMBER
0x0000001b	27	GSKKM_ERR_DATABASE_DUPLICATE_KEY_SUBJECT_PUBLIC_KEY_INFO
0x0000001c	28	GSKKM_ERR_DATABASE_DUPLICATE_KEY_UNSIGNED_CRL
0x0000001d	29	GSKKM_ERR_DATABASE_DUPLICATE_LABEL
0x0000001e	30	GSKKM_ERR_DATABASE_PASSWORD_ENCRYPTION
0x0000001f	31	GSKKM_ERR_DATABASE_LDAP
0x00000020	32	GSKKM_ERR_CRYPTO
0x00000021	33	GSKKM_ERR_CRYPTO_ENGINE
0x00000022	34	GSKKM_ERR_CRYPTO_ALGORITHM
0x00000023	35	GSKKM_ERR_CRYPTO_SIGN
0x00000024	36	GSKKM_ERR_CRYPTO_VERIFY
0x00000025	37	GSKKM_ERR_CRYPTO_DIGEST
0x00000026	38	GSKKM_ERR_CRYPTO_PARAMETER
0x00000027	39	GSKKM_ERR_CRYPTO_UNSUPPORTED_ALGORITHM
0x00000028	40	GSKKM_ERR_CRYPTO_INPUT_GREATER_THAN_MODULUS
0x00000029	41	GSKKM_ERR_CRYPTO_UNSUPPORTED_MODULUS_SIZE
0x0000002a	42	GSKKM_ERR_VALIDATION
0x0000002b	43	GSKKM_ERR_VALIDATION_KEY
0x0000002c	44	GSKKM_ERR_VALIDATION_DUPLICATE_EXTENSIONS

Table 3. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x0000002d	45	GSKKM_ERR_VALIDATION_KEY_WRONG_VERSION
0x0000002e	46	GSKKM_ERR_VALIDATION_KEY_EXTENSIONS_REQUIRED
0x0000002f	47	GSKKM_ERR_VALIDATION_KEY_VALIDITY
0x00000030	48	GSKKM_ERR_VALIDATION_KEY_VALIDITY_PERIOD
0x00000031	49	GSKKM_ERR_VALIDATION_KEY_VALIDITY_PRIVATE_KEY_USAGE
0x00000032	50	GSKKM_ERR_VALIDATION_KEY_ISSUER_NOT_FOUND
0x00000033	51	GSKKM_ERR_VALIDATION_KEY_MISSING_REQUIRED_EXTENSIONS
0x00000034	52	GSKKM_ERR_VALIDATION_KEY_BASIC_CONSTRAINTS
0x00000035	53	GSKKM_ERR_VALIDATION_KEY_SIGNATURE
0x00000036	54	GSKKM_ERR_VALIDATION_KEY_ROOT_KEY_NOT_TRUSTED
0x00000037	55	GSKKM_ERR_VALIDATION_KEY_IS_REVOKED
0x00000038	56	GSKKM_ERR_VALIDATION_KEY_AUTHORITY_KEY_IDENTIFIER
0x00000039	57	GSKKM_ERR_VALIDATION_KEY_PRIVATE_KEY_USAGE_PERIOD
0x0000003a	58	GSKKM_ERR_VALIDATION_SUBJECT_ALTERNATIVE_NAME
0x0000003b	59	GSKKM_ERR_VALIDATION_ISSUER_ALTERNATIVE_NAME
0x0000003c	60	GSKKM_ERR_VALIDATION_KEY_USAGE
0x0000003d	61	GSKKM_ERR_VALIDATION_KEY_UNKNOWN_CRITICAL_EXTENSION
0x0000003e	62	GSKKM_ERR_VALIDATION_KEY_PAIR
0x0000003f	63	GSKKM_ERR_VALIDATION_CRL
0x00000040	64	GSKKM_ERR_MUTEX
0x00000041	65	GSKKM_ERR_PARAMETER
0x00000042	66	GSKKM_ERR_NULL_PARAMETER
0x00000043	67	GSKKM_ERR_NUMBER_SIZE
0x00000044	68	GSKKM_ERR_OLD_PASSWORD
0x00000045	69	GSKKM_ERR_NEW_PASSWORD
0x00000046	70	GSKKM_ERR_PASSWORD_EXPIRATION_TIME
0x00000047	71	GSKKM_ERR_THREAD
0x00000048	72	GSKKM_ERR_THREAD_CREATE
0x00000049	73	GSKKM_ERR_THREAD_WAIT_FOR_EXIT
0x0000004a	74	GSKKM_ERR_IO

Table 3. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x0000004b	75	GSKKM_ERR_LOAD
0x0000004c	76	GSKKM_ERR_PKCS11
0x0000004d	77	GSKKM_ERR_NOT_INITIALIZED
0x0000004e	78	GSKKM_ERR_DB_TABLE_CORRUPTED
0x0000004f	79	GSKKM_ERR_MEMORY_ALLOCATE
0x00000050	80	GSKKM_ERR_UNSUPPORTED_OPTION
0x00000051	81	GSKKM_ERR_GET_TIME
0x00000052	82	GSKKM_ERR_CREATE_MUTEX
0x00000053	83	GSKKM_ERR_CMDCAT_OPEN
0x00000054	84	GSKKM_ERR_ERRCAT_OPEN
0x00000055	85	GSKKM_ERR_FILENAME_NULL
0x00000056	86	GSKKM_ERR_FILE_OPEN
0x00000057	87	GSKKM_ERR_FILE_OPEN_TO_READ
0x00000058	88	GSKKM_ERR_FILE_OPEN_TO_WRITE
0x00000059	89	GSKKM_ERR_FILE_OPEN_NOT_EXIST
0x0000005a	90	GSKKM_ERR_FILE_OPEN_NOT_ALLOWED
0x0000005b	91	GSKKM_ERR_FILE_WRITE
0x0000005c	92	GSKKM_ERR_FILE_REMOVE
0x0000005d	93	GSKKM_ERR_BASE64_INVALID_DATA
0x0000005e	94	GSKKM_ERR_BASE64_INVALID_MSGTYPE
0x0000005f	95	GSKKM_ERR_BASE64_ENCODING
0x00000060	96	GSKKM_ERR_BASE64_DECODING
0x00000061	97	GSKKM_ERR_DN_TAG_NULL
0x00000062	98	GSKKM_ERR_DN_CN_NULL
0x00000063	99	GSKKM_ERR_DN_C_NULL
0x00000064	100	GSKKM_ERR_INVALID_DB_HANDLE
0x00000065	101	GSKKM_ERR_KEYDB_NOT_EXIST
0x00000066	102	GSKKM_ERR_KEYPAIRDB_NOT_EXIST
0x00000067	103	GSKKM_ERR_PWDFILE_NOT_EXIST
0x00000068	104	GSKKM_ERR_PASSWORD_CHANGE_MATCH
0x00000069	105	GSKKM_ERR_KEYDB_NULL
0x0000006a	106	GSKKM_ERR_REQKEYDB_NULL
0x0000006b	107	GSKKM_ERR_KEYDB_TRUSTCA_NULL
0x0000006c	108	GSKKM_ERR_REQKEY_FOR_CERT_NULL
0x0000006d	109	GSKKM_ERR_KEYDB_PRIVATE_KEY_NULL
0x0000006e	110	GSKKM_ERR_KEYDB_DEFAULT_KEY_NULL
0x0000006f	111	GSKKM_ERR_KEYREC_PRIVATE_KEY_NULL
0x00000070	112	GSKKM_ERR_KEYREC_CERTIFICATE_NULL
0x00000071	113	GSKKM_ERR_CRLS_NULL

Table 3. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x00000072	114	GSKKM_ERR_INVALID_KEYDB_NAME
0x00000073	115	GSKKM_ERR_UNDEFINED_KEY_TYPE
0x00000074	116	GSKKM_ERR_INVALID_DN_INPUT
0x00000075	117	GSKKM_ERR_KEY_GET_BY_LABEL
0x00000076	118	GSKKM_ERR_LABEL_LIST_CORRUPT
0x00000077	119	GSKKM_ERR_INVALID_PKCS12_DATA
0x00000078	120	GSKKM_ERR_PKCS12_PWD_CORRUPTION
0x00000079	121	GSKKM_ERR_EXPORT_TYPE
0x0000007a	122	GSKKM_ERR_PBE_ALG_UNSUPPORT
0x0000007b	123	GSKKM_ERR_KYR2KDB
0x0000007c	124	GSKKM_ERR_KDB2KYR
0x0000007d	125	GSKKM_ERR_ISSUING_CERTIFICATE
0x0000007e	126	GSKKM_ERR_FIND_ISSUER_CHAIN
0x0000007f	127	GSKKM_ERR_WEBDB_DATA_BAD_FORMAT
0x00000080	128	GSKKM_ERR_WEBDB_NOTHING_TO_WRITE
0x00000081	129	GSKKM_ERR_EXPIRE_DAYS_TOO_LARGE
0x00000082	130	GSKKM_ERR_PWD_TOO_SHORT
0x00000083	131	GSKKM_ERR_PWD_NO_NUMBER
0x00000084	132	GSKKM_ERR_PWD_NO_CONTROL_KEY
0x00000085	133	GSKKM_ERR_SIGNATURE_ALGORITHM
0x00000086	134	GSKKM_ERR_INVALID_DATABASE_TYPE
0x00000087	135	GSKKM_ERR_SECONDARY_KEYDB_TO_OTHER
0x00000088	136	GSKKM_ERR_NO_SECONDARY_KEYDB
0x00000089	137	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_LABEL_NOT_EXIST
0x0000008a	138	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_PASSWORD_REQUIRED
0x0000008b	139	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_PASSWORD_NOT_REQUIRED
0x0000008c	140	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_LIBRARY_NOT_LOADED
0x0000008d	141	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_NOT_SUPPORT
0x0000008e	142	GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_FUNCTION_FAILED
0x0000008f	143	GSKKM_ERR_LDAP_USER_NOT_FOUND
0x00000090	144	GSKKM_ERR_LDAP_INVALID_PASSWORD
0x00000091	145	GSKKM_ERR_LDAP_QUERY_ENTRY_FAILED
0x00000092	146	GSKKM_ERR_INVALID_CERT_CHAIN
0x00000093	147	GSKKM_ERR_CERT_ROOT_NOT_TRUSTED
0x00000094	148	GSKKM_ERR_CERT_REVOKED

Table 3. Key management return codes (continued)

Return code (hexadecimal)	Return code (decimal)	Constant
0x00000095	149	GSKKM_ERR_CRYPTOGRAPHIC_OBJECT_FUNCTION_FAILED
0x00000096	150	GSKKM_ERR_NO_AVAILABLE_CRL_DATASOURCE
0x00000097	151	GSKKM_ERR_NO_TOKEN_PRESENT
0x00000098	152	GSKKM_ERR_FIPS_NOT_SUPPORTED
0x00000099	153	GSKKM_ERR_FIPS_CONFLICT_SETTING
0x0000009a	154	GSKKM_ERR_PASSWORD_STRENGTH_FAILED

IBM Database Encryption Expert for encryption of data at rest

IBM Database Encryption Expert 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.

Database Encryption Expert helps organizations ensure that private and confidential data is strongly protected and in compliance with regulations and legislative acts. The key benefits of Database Encryption Expert 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

Database Encryption Expert 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.

Database Encryption Expert is transparent to users, databases, applications, and storage. No code changes or changes to existing infrastructure are required. Database Encryption Expert can protect data in any storage environment, while users continue to access data the in the same way as before.

Database Encryption Expert 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: You cannot use Database Encryption Expert in DB2 pureScale® environments.

Architecture of Database Encryption Expert

Database Encryption Expert is a set of agent and server software packages that you administer by using a Web-based user-interface and command-line utilities. The Database Encryption Expert administrator configures security policies that govern how security and encryption are implemented.

According to how these security policies are defined, the Database Encryption Expert backup agent encrypts DB2 backups, and the Database Encryption Expert file system agent encrypts DB2 data files.

The Encryption Expert 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 illustrates the architecture of Database Encryption Expert.

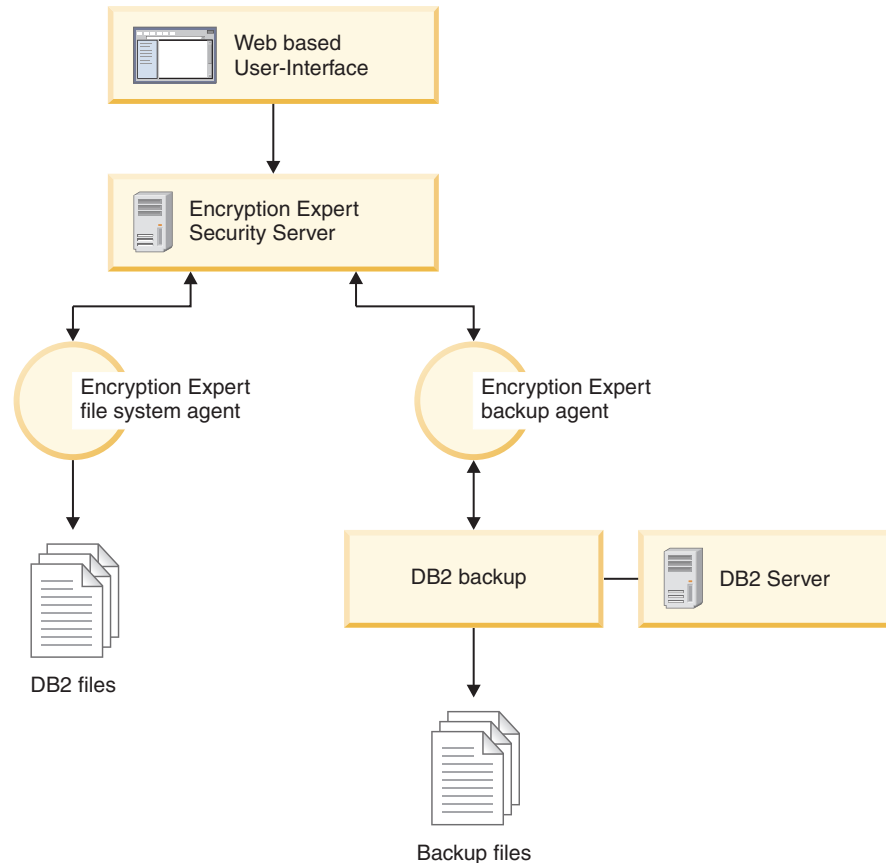


Figure 4. Architecture of Database Encryption Expert

File system agent

The Database Encryption Expert file system agent process is always running in the background. The agent intercepts any attempt to access data files, directories, or executables that you are protecting. The Database Encryption Expert 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.

Database Encryption Expert 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 Database Encryption Expert server, including native database compression. Other than an additional command-line argument, DB2 backup operators are unaware of Database Encryption Expert intervention. Database Encryption Expert 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 Encryption Expert servers in different data centers.

Audit logging

Database Encryption Expert 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 Database Encryption Expert system events, such as initialization, shut down and restart; and network connects and disconnects between different Database Encryption Expert components.

Database Encryption Expert documentation

For more information about Database Encryption Expert, go to the following web page: <http://publib.boulder.ibm.com/infocenter/mptoolic/v1r0/topic/com.ibm.db2tools.eet.doc.ug/eetwelcome.htm>.

Database encryption using AIX encrypted file system (EFS)

For DB2 Enterprise Server Edition running 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 `cllc.rte` fileset must be installed. The `cllc.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=abstgrp groups=abstgrp,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:
```

```

key          Kind ..... Group
             Id (uid / gid) ..... 212
             Type ..... Private
key          Algorithm ..... RSA_1024
             Validity ..... Key is
valid       Fingerprint .....
339dfb19:bc850f4c:5551c975:7fe4961b:2dddf3bc

```

- 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.
- 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.
- 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`
- 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 **smit** 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.

- 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

- 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:

- 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
```

- 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
```

- 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.

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, SYSMMAINT_GROUP, or SYSMON_GROUP configuration parameters.
- System Administration (SYSADMIN). Generates records when operations requiring SYSADM, SYSMMAINT, 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 4. 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 ADMINPOLICY CATEGORIES EXECUTE STATUS BOTH,
    SYSADMIN STATUS BOTH ERROR TYPE AUDIT
COMMIT
AUDIT SYSADM, DBADM USING POLICY ADMINPOLICY
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 WEBAPPOLICY CATEGORIES EXECUTE WITH DATA
    STATUS BOTH ERROR TYPE AUDIT
COMMIT
AUDIT ROLE TELLER, ROLE CLERK USING POLICY WEBAPPOLICY
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 partitioned database environment. In a partitioned 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 partitioned database environment

In a partitioned database environment, the same active audit log location (set by the **datapath** parameter) must be used on each partition. There are two ways to accomplish this:

1. Use database partition expressions when you specify the **datapath** parameter. Using database partition expressions allows the partition number to be included in the path of the audit log files and results in a different path on each database partition.
2. Use a shared drive that is the same on all members.

You can use database partition expressions anywhere within the value you specify for the **datapath** parameter. For example, on a three member system, where the database partition number is 10, the following command:

```
db2audit configure datapath '/pathForNode $N'
```

will use the following paths:

- /pathForNode10
- /pathForNode20
- /pathForNode30

Note: You cannot use database partition 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 partitioned database environment

In a partitioned 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 partition 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 partition they originate from in 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 partitioned database environment, the path for the active audit log can be a directory that is unique to each partition so that each partition writes to an individual file. In order to accurately track the origin of audit records, the partition number is included as part of the audit log file name. For example, on partition 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 non-partitioned database environment the partition 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.partition.YYYYMMDDHHMMSS.

database-level archived audit log

The file name of the database-level archived audit log is:
db2audit.dbdatabase.log.partition.YYYYMMDDHHMMSS.

In a non-partitioned database environment, the value for *partition* 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 partitioned database environment, if the server is running when archive is run, the timestamp is consistent across partitions and reflects the timestamp generated at the partition 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 `sql1ib/misc` directory (`sql1ib\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 sql1ib/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_DELM_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 5. 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 5. 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 auditlobs 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"> • audit.del • checking.del • objmaint.del • secmaint.del • sysadmin.del • validate.del • context.del • execute.del • auditlobs <p>If the files already exist, the output is appended to them. The auditlobs 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

The EXECUTE category allows you to accurately track the SQL statements and user issues. 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 6. 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_DELM_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_DELM_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.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
  min_dec_div_3=NO
  degree=1
  sqlrules=DB2
  refresh age=+00000000000000.000000
  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 charde10x3b 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 SYSMANT 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 SYSMANT 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.

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 7. 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)
```

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 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 Version 10.1 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 for Linux, UNIX, and Windows 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.

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.

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 DB2 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 DB2 row and column access control feature which addresses the challenge.

Security challenge	Row and column access control feature which addresses the security challenge
<p>Limiting column access to only privileged users.</p> <p>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.</p>	<p>Column masks can be used to filter or hide sensitive data from Jane.</p>
<p>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.</p>	<p>Row permissions can be implemented to control which user can view any particular row.</p>
<p>Restricting data on a need-to-know basis.</p>	<p>Row permissions can help with this challenge as well by restricting table level data at the user level.</p>
<p>Restricting other database objects like UDFs, triggers, views on RCAC secured data.</p>	<p>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.</p>

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. XXXXXXXXXXXX 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 XXXXXXXXXXXX 0.00      LEE
123-58-9812 MIKE     Mike    Long Strt  XXXXXXXXXXXX 0.00      JAMES
123-11-9856 SAM      Sam     Big Strt  XXXXXXXXXXXX 0.00      LEE
123-19-1454 DUG      Dug     Good Strt  XXXXXXXXXXXX 0.00      JAMES
123-45-6789 BOB      Bob     123 Some St. XXXXXXXXXXXX 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 XXXXXXXXXXXX 0.00      LEE
XXX-XX-9812 MIKE     Mike    Long Strt  XXXXXXXXXXXX 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 DB2 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 DB2 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 RFACTSPM.CUSTOMER TO USER NEWTON, USER ZURBIE, USER PLATO;

--Insert data

INSERT INTO RFACTSPM.CUSTOMER VALUES ('1111-2222-3333-4444', 'Alice', 22000, 'A');
INSERT INTO RFACTSPM.CUSTOMER VALUES ('2222-3333-4444-5555', 'Bob', 71000, 'A');
INSERT INTO RFACTSPM.CUSTOMER VALUES ('3333-4444-5555-6666', 'Carl', 123000, 'B');
INSERT INTO RFACTSPM.CUSTOMER VALUES ('4444-5555-6666-7777', 'David', 172000, 'C');

INSERT INTO RFACTSPM.INTERNAL_INFO VALUES ('A', 'NEWTON');
INSERT INTO RFACTSPM.INTERNAL_INFO VALUES ('B', 'ZURBIE');
INSERT INTO RFACTSPM.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 RFACTSPM.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 RFACTSPM.INTERNAL_INFO WHERE EMP_ID = USER)
ENFORCED FOR ALL ACCESS
ENABLE;

CREATE PERMISSION CSR_ROW_ACCESS ON RFACTSPM.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 'xxx-xxx-xxx-' || 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

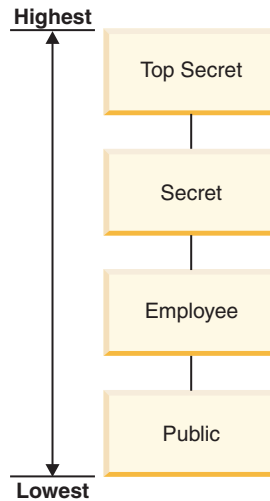
ARRAY is one type of security label component.

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.

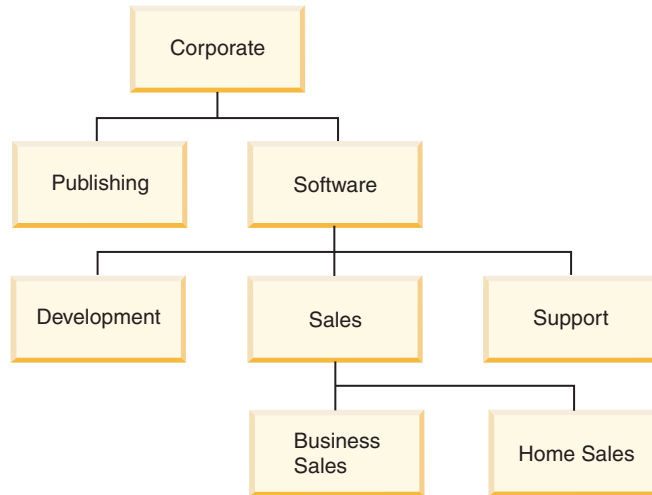
Example: If the component mycomp is defined this way:

```

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'
)

```

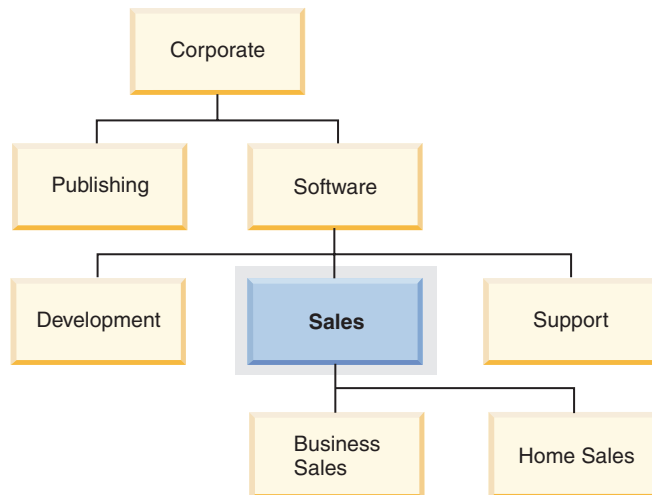
Then the elements are treated as if they are organized in a tree structure like this:



In a component of type TREE, the elements can have these types of relationships to each other:

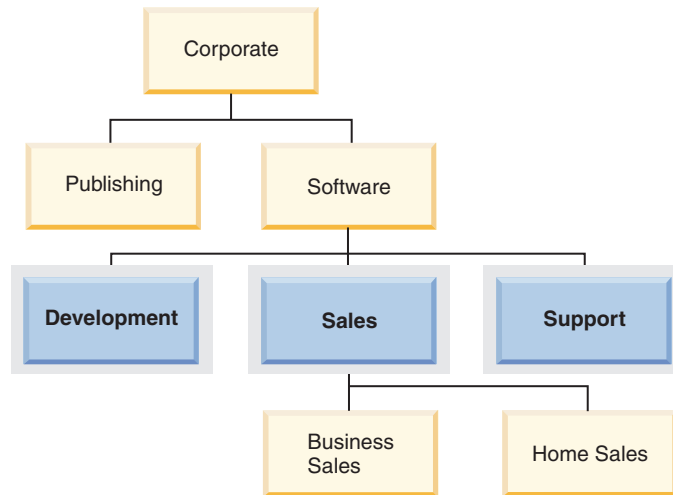
Parent Element A is a parent of element B if element B is UNDER element A.

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.

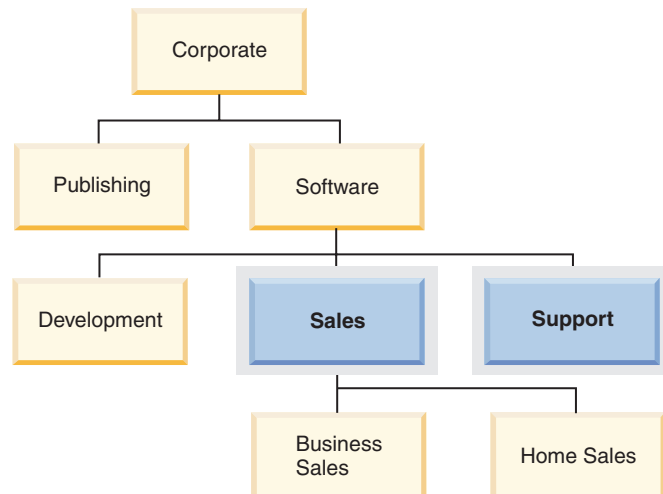
Example: This diagram shows the children of the Software element:



Sibling

Two elements are siblings of each other if they have the same parent.

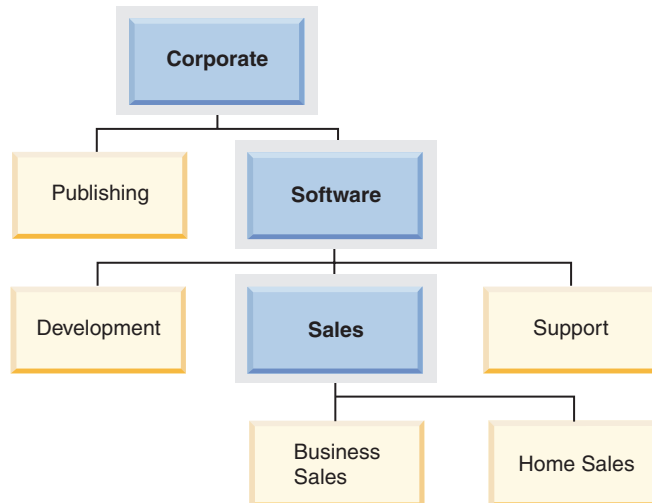
Example: This diagram shows the siblings of the Development element:



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.

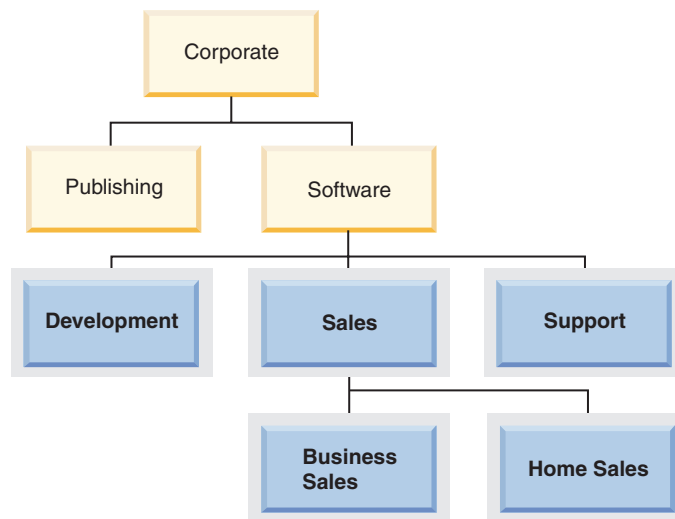
Example: This diagram shows the ancestors of the Home Sales element:



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 8. Example values for a security label

Component	Values
Level	Secret
Department	<i>Empty value</i>
Projects	<ul style="list-style-type: none">• Epsilon 37• Megaphone• Cloverleaf

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 9. 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. ¹

Table 9. 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 10. 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:

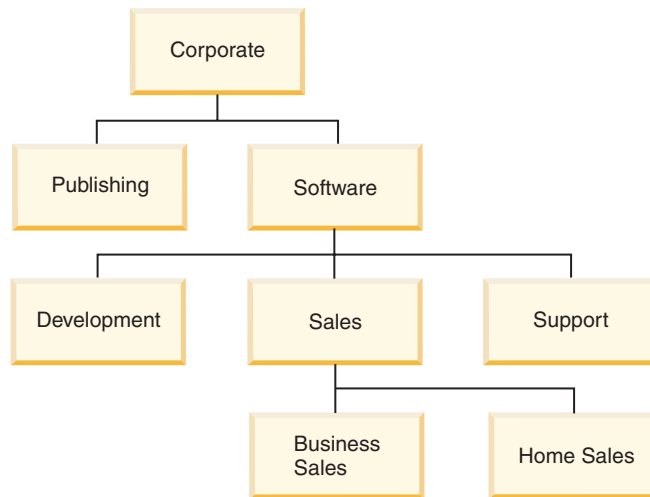


Table 11. Examples of applying the DB2LBACREADTREE and DB2LBACWRITETREE rules.

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:

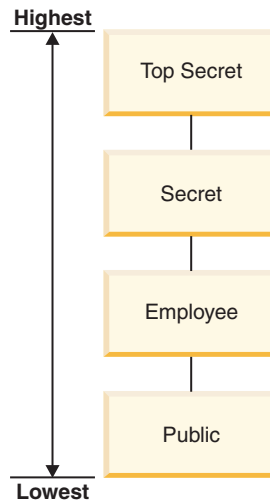


Table 12. Examples of applying the DB2LBACREADARRAY rule.

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:

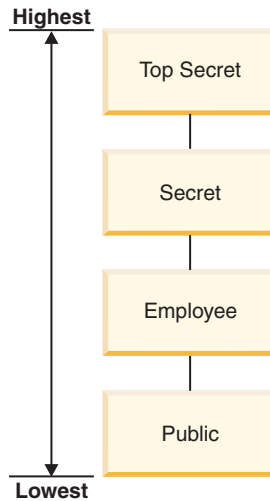


Table 13. Examples of applying the DB2LBACWRITEARRAY rule.

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.

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 14. 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 15. 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 16. 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 17. 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 18. 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 19. 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 20. 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 21. Table T1

EMPNO	LASTNAME	DEPTNO <i>Protected by</i> L2	PAYSCALE <i>Protected by</i> L3
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 22. Table T1 After Update

EMPNO	LASTNAME	DEPTNO <i>Protected by</i> L2	PAYSCALE <i>Protected by</i> L3
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 23. Table T1 After Second Update

EMPNO	LASTNAME	DEPTNO <i>Protected by L2</i>	PAYSCALE <i>Protected by L3</i>
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 24. 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 25. 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 26. Jenni's UPDATE & SELECT Query Result

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L0

The actual data in the table looks like this:

Table 27. 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 28. 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:

LASTNAME	DEPTNO <i>Protected by L2</i>	LABEL
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:

Security label	Read access?	Write access?
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:

Security label	Read access?	Write access?
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.PASSTHROUGHAUTH

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
- SYSMANT
- 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.PASSTHROUGHAUTH
- SYSCAT.ROUTINEAUTH
- SYSCAT.SCHEMAAUTH
- SYSCAT.SECURITYLABELACCESS
- SYSCAT.SECURITYPOLICYEXEMPTIONS
- SYSCAT.SEQUENCEAUTH
- SYSCAT.SURROGATEAUTHIDS
- SYSCAT.TBAUTH
- 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.

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:

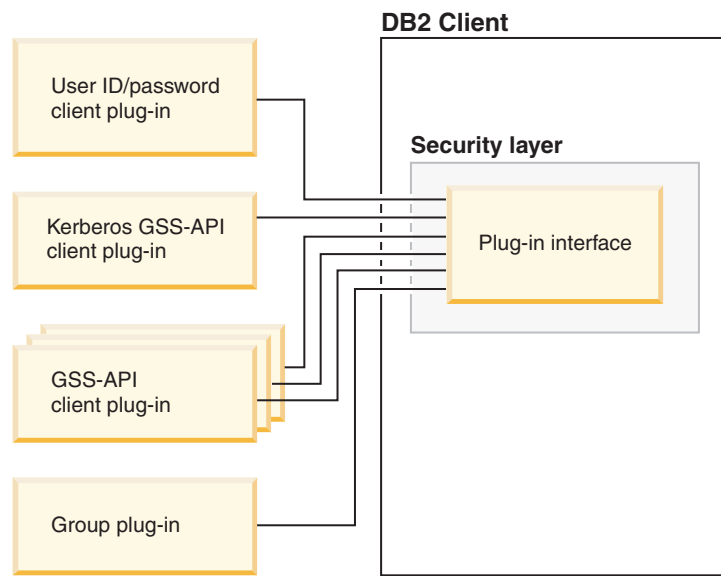


Figure 5. Deploying Security Plug-ins on DB2 Clients

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 **srvcn_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 **srvcn_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 **srvcn_auth** configuration parameter to GSSPLUGIN; and
- Set the **authentication** configuration parameter to SERVER.

The **srvcn_auth** configuration parameter is a means to override the authentication type used by incoming connections. These connections use the authentication method specified by the **srvcn_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:

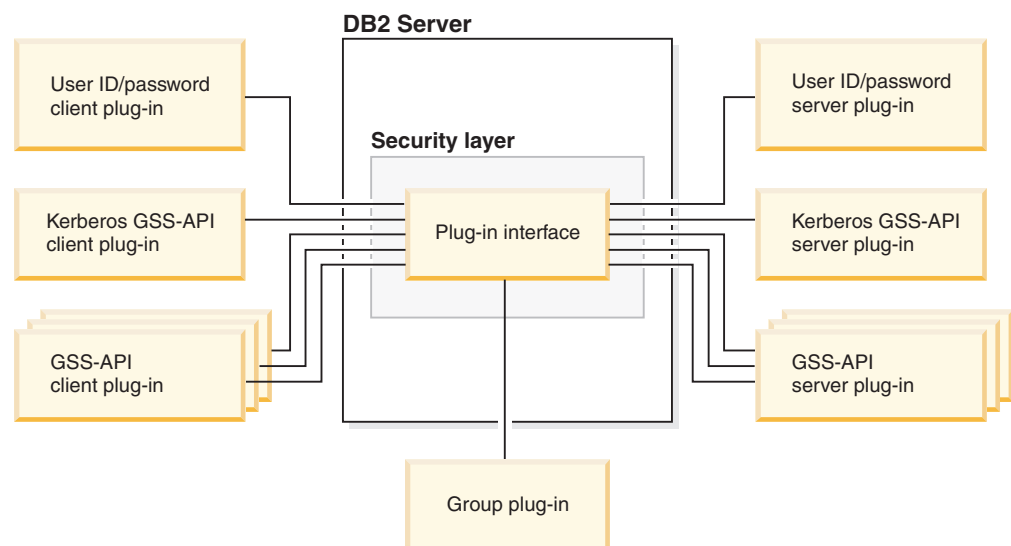


Figure 6. Deploying Security Plug-ins on DB2 Servers

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 29. 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 **clnt_krb_plugin** configuration parameter.
- Otherwise, the DB2 database manager uses the plug-in specified by the **clnt_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 30. 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 `sqlllib/db2dump/instance name.N.nfy`. To see an administration notification log on Windows operating systems, use the Event Viewer tool. The Event Viewer tool can be found by navigating from the Windows operating system "Start" button to Settings -> Control Panel -> Administrative Tools -> Event Viewer. Following are the administration notification log values related to security plug-ins:

- 13000 indicates that a call to a GSS-API security plug-in API failed with an error, and returned an optional error message.
 SQLT_ADMIN_GSS_API_ERROR (13000)
 Plug-in "*plug-in name*" received error code "*error code*" from GSS API "*gss api name*" with the error message "*error message*"
- 13001 indicates that a call to a DB2 security plug-in API failed with an error, and returned an optional error message.
 SQLT_ADMIN_PLUGIN_API_ERROR(13001)
 Plug-in "*plug-in name*" received error code "*error code*" from DB2 security plug-in API "*gss api name*" with the error message "*error message*"
- 13002 indicates that DB2 failed to unload a plug-in.
 SQLT_ADMIN_PLUGIN_UNLOAD_ERROR (13002)
 Unable to unload plug-in "*plug-in name*". No further action required.
- 13003 indicates a bad principal name.
 SQLT_ADMIN_INVALID_PRIN_NAME (13003)
 The principal name "*principal name*" used for "*plug-in name*" is invalid. Fix the principal name.
- 13004 indicates that the plug-in name is not valid. Path separators (On UNIX "/" and on Windows "\\") are not allowed in the plug-in name.
 SQLT_ADMIN_INVALID_PLGN_NAME (13004)
 The plug-in name "*plug-in name*" is invalid. Fix the plug-in name.
- 13005 indicates that the security plug-in failed to load. Ensure the plug-in is in the correct directory and that the appropriate database manager configuration parameters are updated.
 SQLT_ADMIN_PLUGIN_LOAD_ERROR (13005)
 Unable to load plug-in "*plug-in name*". Verify the plug-in existence and directory where it is located is correct.
- 13006 indicates that an unexpected error was encountered by a security plug-in. Gather all the **db2support** information, if possible capture a **db2trc**, and then call IBM support for further assistance.
 SQLT_ADMIN_PLUGIN_UNEXP_ERROR (13006)
 Plug-in encountered unexpected error. Contact IBM Support for further assistance.

Note: If you are using security plug-ins on a Windows 64-bit database server and are seeing a load error for a security plug-in, see the topics about 32-bit and 64-bit considerations and security plug-in naming conventions. The 64-bit plug-in library requires the suffix "64" on the library name, but the entry in the security plug-in database manager configuration parameters should not indicate this suffix.

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 **srvcn_gssplugin_list** with an ordered, comma-delimited list of the names of the plug-ins installed in the GSS-API plug-in directory.
 3. Either:
 - Setting the database manager configuration parameter **srvcn_auth** to GSSPLUGIN or GSS_SERVER_ENCRYPT is a way to enable the server to use GSSAPI PLUGIN authentication method. Or:
 - Setting the database manager configuration parameter **srvcn_auth** to NOT_SPECIFIED and setting **authentication** to GSSPLUGIN or GSS_SERVER_ENCRYPT is a way to enable the server to use GSSAPI PLUGIN authentication method.
- To deploy a GSS-API authentication plug-in on database clients, perform the following steps on each client:
 1. Copy the GSS-API authentication plug-in library in the client plug-in directory. You can copy numerous GSS-API plug-ins into this directory. The client selects a GSS-API plug-in for authentication during a CONNECT or ATTACH operation by picking the first GSS-API plug-in contained in the server's plug-in list that is available on the client.
 2. Optional: Catalog the databases that the client will access, indicating that the client will only accept a GSS-API authentication plug-in as the authentication mechanism. For example:

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION GSSPLUGIN
```
- For local authorization on a client, server, or gateway using a GSS-API authentication plug-in, perform the following steps:
 1. Copy the GSS-API authentication plug-in library in the client plug-in directory on the client, server, or gateway.
 2. Update the database manager configuration parameter **local_gssplugin** with the name of the plug-in.
 3. Set the **authentication** database manager configuration parameter to GSSPLUGIN, or GSS_SERVER_ENCRYPT.

Deploying a Kerberos plug-in

To customize the 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 the Linux, HP-UX and Solaris 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
- HP-UX
- Linux
- Solaris

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
- HP-UX on Itanium-based HP Integrity Series systems (IA-64)
- Linux on IA32, x64, or zSeries® hardware
- Solaris
- 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 three versions of LDAP clients: ITDS V5.2 which ships with AIX V5.3, 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 V5.3 system:

```
$ lsipp -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://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.security/doc/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://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=/com.ibm.aix.files/doc/aixfiles/user.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 **OSAUTHDB**. As a user with **SYSADM** authority run **db2set DB2AUTH=OSAUTHDB**.
 - 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_authtok 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_authtok 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. Find the **group** and **passwd** entries in `/etc/nsswitch.conf` file and ensure `ldap` is entered as a lookup method. Here is an example of the **group** and **passwd** entry:

```
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 `0SAUTHDB`. Issue the following command as a user with `SYSADM` authority:

```
db2set DB2AUTH=0SAUTHDB
```
 - 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 transparent LDAP for authentication and group lookup (HP-UX)

Starting in DB2 V9.7 Fix Pack 1 and later, to ensure your DB2 database server transparently uses LDAP-based authentication on the HP-UX operating system, you need to use Pluggable Authentication Modules (PAM). Your LDAP server should already be configured to store user and group information.

Before you begin

This procedure assumes that the LDAP server is RFC 2307 compliant.

Procedure

1. If you are using IBM Tivoli Directory Server (ITDS) Version 6.1, set up the LDAP server before the HP-UX system can connect to it. To configure your LDAP server on the HP-UX operating system, perform the following steps:
 - a. Log in on the LDAP server as a user with root authority.
 - b. Issue the **idsldapadd** command:

```
idsldapadd -D <root> -w <password> -h <hostname> -p <port> -c -i duaconfigschem.ldif
```

```
where,  
<root> - the bind dn to bind to LDAP  
<password> - the password for bind dn  
<hostname> - hostname of the LDAP server  
<port> - the port LDAP server is running. Default is 389  
<schema.ldif> - LDIF file contains DUAConfigProfile Schema
```

The object class listed in the `duaconfigschem.ldif` is added to the LDAP server automatically using the LDAP-UX setup program if either Netscape or Red Hat Directory Servers are employed. However, if ITDS is used the object class must be added manually before running the LDAP-UX setup program on the HP-UX Client.

2. To configure your operating system for LDAP and PAM, perform the following steps:

- a. Log in as a user with root authority.
- b. Install LDAP-UX Client Service and run the LDAP-UX setup program. The following screen will appear:

```
[ctrl-B]=Go Back screen 2  
Hewlett-Packard Company  
LDAP-UX Client Services Setup Program
```

```
-----  
Select which Directory Server you want to connect to:  
1. Netscape or Red Hat Directory  
2. Windows 2000/2003/2003 R2 Active Directory  
To accept the default shown in brackets, press the Return key.  
Directory Server: [1]:
```

Select option 1, as though you are connecting to the Netscape or Red Hat Directory Server and follow the instructions.

For details on installing LDAP-UX, refer to the *LDAP-UX Client Services B.04.15 Administrator's Guide*.

- c. Edit the PAM configuration file at `/etc/pam.conf`. Add the following text to the file:

```
db2 auth required          libpam_hpsec.so.1  
db2 auth sufficient       libpam_unix.so.1  
db2 auth required         libpam_ldap.so.1 use_first_pass
```

The previous configuration first checks the userid and password against the local file system. It will only conduct a LDAP lookup if the user is not found or if authentication with local file system fails.

DB2 supports PAM configurations that use `libpam_ldap.so` and `libpam_unix.so`. Configurations that use other PAM modules might work, but are not supported.

- d. Setup your HP-UX system to perform group lookup through LDAP. Find the **group** and **passwd** entries in `/etc/nsswitch.conf` file and ensure `ldap` is entered as a lookup method. Here is an example of the **group** and **passwd** entry:


```
group:      files ldap
passwd:     files ldap
```

3. 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. 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
 - CLIENT
 - c. Ensure that you are using the default empty values for Client Userid-Password Plugin (clnt_pw_plugin), Server Userid-Password Plugin (srvcon_pw_plugin) and Group Plugin (group_plugin). The default plug-ins are IBMOSauthclient, IBMOSauthserver and IBMOSgroups, and those plug-ins are implied if you leave the values for the plug-in name empty.
 - d. Restart the DB2 instance.

Note: IBMLDAPSecurity.ini is not used by transparent LDAP. This file is used only with LDAP plug-in modules.

Configuring transparent LDAP for authentication and group lookup (Solaris)

Starting in DB2 V9.7 Fix Pack 1 and later, to ensure your DB2 database server transparently uses LDAP-based authentication on the Solaris operating system, you need to use Pluggable Authentication Modules (PAM). Your LDAP server should already be configured to store user and group information.

Before you begin

This procedure assumes that the LDAP server is RFC 2307 compliant.

About this task

This task describes the steps that are applicable to Solaris 10. The instructions might vary slightly for other versions of Solaris operating systems.

Procedure

1. Configure your operating system for LDAP and PAM by performing 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 nss_ldap.so and pam_ldap.so in /usr/lib and /usr/lib/security directories.
 - c. Setup your operating system to act as a LDAP client machine. The ldapclient(1M) interface can be used to issue the **ldapclient** command. Here is the sample output:

```
ldapclient manual -a credentialLevel=proxy \  
-a authenticationMethod=simple \  
-a proxyDN=<root> \  
-
```

```
-a proxyPassword=<password> \
-a defaultSearchBase=<base> \
-a serviceSearchDescriptor=group:<group> \
-a domainName=<domain> \
-a defaultServerList=<IP>
```

where,

<root> the bind dn to bind to LDAP. This is the dn of the user entry in the LDAP server that is permitted to search the LDAP server for user accounts and groups

<password>
the password for bind dn

<base>
the dn for the search base. This should be one level above the user and group entry

<group>
the base dn for where the group information is stored

<domain>
the domain name for the LDAP server

<IP> the IP address for the LDAP server

For more information, refer to the `ldapclient(1M)` manual.

- d. Edit the PAM configuration file at `/etc/pam.conf`. Add the following text to the file:

```
db2 auth requisite      pam_authtok_get.so.1
db2 auth required      pam_unix_cred.so.1
db2 auth sufficient    pam_unix_auth.so.1
db2 auth required      pam_ldap.so.1
```

The previous configuration first checks the userid and password against the local file system. It will only conduct a LDAP lookup if the user is not found or if authentication with local file system fails.

DB2 supports PAM configurations that use `pam_ldap.so` and `pam_unix_auth.so`. Configurations that use other PAM modules might work, but are not supported.

- e. Setup your Solaris system to perform group lookup through LDAP. Find the **group** and **passwd** entries in `/etc/nsswitch.conf` file and ensure `ldap` is entered as a lookup method. Here is an example of the **group** and **passwd** entry:

```
group:      files ldap
passwd:     files ldap
```

2. Configure your DB2 instance to use transparent LDAP authentication by performing the following steps:

- a. Set the **DB2AUTH** miscellaneous registry variable to `0SAUTHDB`. Issue the following command as a user with `SYSADM` authority:

```
db2set DB2AUTH=0SAUTHDB
```

- 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.

Note: IBMLDAPSecurity.ini is not used by transparent LDAP. This file is used only with LDAP plug-in modules.

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/sqllib/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 31. 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

Table 32. 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

Table 32. User-related values (continued)

Parameter	Description
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 33. 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
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"> 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). 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). 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

Table 33. Group-related values (continued)

Parameter	Description
NESTED_GROUPS	<p>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.</p> <p>Cycles (such as A belongs to B, and B belongs to A) are handled correctly.</p> <p>This parameter is optional, and defaults to FALSE.</p>

Table 34. Miscellaneous values

Parameter	Description
SEARCH_DN, SEARCH_PW	<p>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.</p> <p>For example: SEARCH_DN = cn=root SEARCH_PW = rootpassword</p>
DEBUG	<p>Set DEBUG to TRUE to write extra information to the db2diag log files to aid in debugging LDAP related issues.</p> <p>Most of the additional information is logged at DIAGLEVEL 4 (INFO). DEBUG defaults to false.</p>

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 35. 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 36. 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 37. 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.

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_USERSPACE_LENGTH 255
#define DB2SEC_MAX_PASSWORD_LENGTH 255
#define DB2SEC_MAX_DBNAME_LENGTH 128
```

A particular plug-in implementation may require or enforce smaller maximum lengths for the authorization IDs, user IDs, and passwords. In particular, the operating system authentication plug-ins supplied with DB2 database systems are restricted to the maximum user, group and namespace length limits enforced by the operating system for cases where the operating system limits are lower than those stated 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 -qmkshrojb -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 -qmkshrojb -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 AUTHID identifier. Version 9.5, and later, of the DB2 database system allows you to have an 128-byte authorization ID, but when the authorization ID is interpreted as an operating system user ID or group name, the operating system naming restrictions apply (for example, a limitation to 8 or 30 character user IDs and 30 character group names). Therefore, while you can grant an 128-byte authorization ID, it is not possible to connect as a user that has that authorization ID. If you write your own security plugin, you should be able to take full advantage of the extended sizes for the authorization ID. For example, you can give your security plug-in a 30-byte user ID and it can return an 128-byte authorization ID during authentication that you are able to connect with.

InfoSphere® Federation Server support restrictions

DB2 II does not support the use of delegated credentials from a GSS_API plug-in to establish outbound connections to data sources. Connections to data sources must continue to use the CREATE USER MAPPING command.

Database Administration Server support restrictions

The DB2 Administration Server (DAS) does not support security plug-ins. The DAS only supports the operating system authentication mechanism.

Security plug-in problem and restriction for DB2 clients (Windows)

When developing security plug-ins that will be deployed in DB2 clients on Windows operating systems, do not unload any auxiliary libraries in the plug-in termination function. This restriction applies to all types of client security plug-ins, including group, user ID and password, Kerberos, and GSS-API plug-ins. Since these termination APIs such as db2secPluginTerm, db2secClientAuthPluginTerm

and `db2secServerAuthPluginTerm` are not called on any Windows platform, you need to do the appropriate resource cleanup.

This restriction is related to cleanup issues associated with the unloading of DLLs on Windows.

Loading plug-in libraries on AIX with extension of `.a` or `.so`

On AIX, security plug-in libraries can have a file name extension of `.a` or `.so`. The mechanism used to load the plug-in library depends on which extension is used:

- Plug-in libraries with a file name extension of `.a`

Plug-in libraries with file name extensions of `.a` are assumed to be archives containing shared object members. These members must be named `shr.o` (32-bit) or `shr64.o` (64-bit). A single archive can contain both the 32-bit and 64-bit members, allowing it to be deployed on both types of platforms.

For example, to build a 32-bit archive style plug-in library:

```
xlc_r -qmkshrobj -o shr.o MyPlugin.c -bE:MyPlugin.exp
ar rv MyPlugin.a shr.o
```

- Plug-in libraries with a file name extension of `.so`

Plug-in libraries with file name extensions of `.so` are assumed to be dynamically loadable shared objects. Such an object is either 32-bit or 64-bit, depending on the compiler and linker options used when it was built. For example, to build a 32-bit plug-in library:

```
xlc_r -qmkshrobj -o MyPlugin.so MyPlugin.c -bE:MyPlugin.exp
```

On all platforms other than AIX, security plug-in libraries are always assumed to be dynamically loadable shared objects.

GSS-API security plug-ins do not support message encryption and signing

Message encryption and signing is not available in GSS-API security plug-ins.

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 38. Security plug-in return codes

Return code	Define value	Meaning	Applicable APIs
0	DB2SEC_PLUGIN_OK	The plug-in API executed successfully.	All
-1	DB2SEC_PLUGIN_UNKNOWNERROR	The plug-in API encountered an unexpected error.	All
-2	DB2SEC_PLUGIN_BADUSER	The user ID passed in as input is not defined.	db2secGenerateInitialCred 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
-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

Table 38. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-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
-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.	db2secProcessServer PrincipalName
-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

Table 38. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-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 or 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 for Linux, UNIX, and Windows 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 **error,
    db2int32 *errorlen );
```

db2secGetGroupsForUser API parameters

authid Input. This parameter value is an SQL authid, which means that DB2 for Linux, UNIX, and Windows converts it to an uppercase character string with no trailing blanks. DB2 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows.

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.

usernamespace

Input. Length in bytes of the **usernamespace** parameter value.

usernamespace

Input. The type of namespace. Valid values for the **usernamespace** 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 **usernamespace** parameter is set to the value `DB2SEC_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 for Linux, UNIX, and Windows. 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 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows if the **authid** is not authenticated (for example, if the **authid** does not match the current connected user).

authpluginamelen

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 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows:

- 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 plug-in. 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 for Linux, UNIX, and Windows 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,
              db2int32 *errmsglen );
```

db2secPluginTerm API parameters

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secPluginTerm API execution is not successful.

errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

APIs for user ID/password authentication plug-ins

For the user ID/password plug-in module, you need to implement the following client-side APIs:

- db2secClientAuthPluginInit

Note: The db2secClientAuthPluginInit API takes as input a pointer, *logMessage_fn, to an API with the following prototype:

```
SQL_API_RC (SQL_API_FN db2secLogMessage)
(
  db2int32 level,
  void *data,
  db2int32 length
);
```

The db2secLogMessage API allows the plug-in to log messages to the **db2diag** log files for debugging or informational purposes. This API is provided by the DB2 database system, so you 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    *userspaceLen,
        db2int32    *userspacetype,
        const char *dbname,
        db2int32    dbnamelen,
        void        **token,
        char        **errorMsg,
        db2int32    *errormsglen
    );
    /* Optional */
    SQL_API_RC (SQL_API_FN * db2secRemapUserid)
    (
        char        userid[DB2SEC_MAX_USERID_LENGTH],
        db2int32    *useridlen,
        char        usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
        db2int32    *userspaceLen,
        db2int32    *userspacetype,
        char        password[DB2SEC_MAX_PASSWORD_LENGTH],
        db2int32    *passwordlen,
        char        newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
        db2int32    *newpasswordlen,
        const char *dbname,
        db2int32    dbnamelen,
        char        **errorMsg,
        db2int32    *errormsglen
    );

    SQL_API_RC (SQL_API_FN * db2secValidatePassword)
    (
        const char *userid,
        db2int32    useridlen,
        const char *userspace,
        db2int32    userspaceLen,
        db2int32    userspacetype,
        const char *password,
        db2int32    passwordlen,
        const char *newpassword,
        db2int32    newpasswordlen,
        const char *dbname,
        db2int32    dbnamelen,
        db2uint32    connection_details,
        void        **token,
        char        **errorMsg,
    );
};
```

```

db2int32 *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 * 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 *userspaceLen,
db2int32 *userspacetype,
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 *namespace,
db2int32 namespaceLen,
db2int32 namespaceType,
const char *password,
db2int32 passwordlen,
const char *newpassword,
db2int32 newpasswordlen,
const char *dbname,
db2int32 dbnameLen,

```

```

gss_cred_id_t *pGSSCredHandle,
void          **initInfo,
char          **errorMsg,
db2int32     *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeToken)
(
void      *token,
char      **errorMsg,
db2int32 *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
char *errorMsg
);

SQL_API_RC (SQL_API_FN * db2secFreeInitInfo)
(
void      *initInfo,
char      **errorMsg,
db2int32 *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char      **errorMsg,
db2int32 *errormsglen
);

/* 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 TCP/IP4 */
    db2UInt32   connect_info_bitmap;
    db2int32    dbnameLen;
    char        dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
    db2UInt32   clientIP6Address[4]; /* Set if protocol is TCP/IP6 */
} db2sec_con_details_2;
```

`db2sec_con_details_3:`

```
typedef struct db2sec_con_details_3
{
    db2int32    clientProtocol; /* See SQL_PROTOCOL_ in sqlenv.h */
    db2UInt32   clientIPAddress; /* Set if protocol is TCP/IP4 */
    db2UInt32   connect_info_bitmap;
    db2int32    dbnameLen;
    char        dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
    db2UInt32   clientIP6Address[4]; /* Set if protocol is TCP/IP6 */
    db2UInt32   clientPlatform; /* SQLM_PLATFORM_* from sqlmon.h */
    db2UInt32   reserved[16];
} db2sec_con_details_3;
```

The possible values for **conDetailsVersion** are `DB2SEC_CON_DETAILS_VERSION_1`, `DB2SEC_CON_DETAILS_VERSION_2`, and `DB2SEC_CON_DETAILS_VERSION_3` representing the version of the API.

Note: While using `db2sec_con_details_1`, `db2sec_con_details_2`, or `db2sec_con_details_3`, consider the following:

- Existing plugins that are using the `db2sec_con_details_1` structure and the `DB2SEC_CON_DETAILS_VERSION_1` value will continue to work as they did with Version 8.2 when calling the `db2GetConDetails` API. If this API is called on an IPv4 platform, the client IP address is returned in the **clientIPAddress** field of the structure. If this API is called on an IPv6 platform, a value of 0 is returned in the **clientIPAddress** field. To retrieve the client IP address on an IPv6 platform, the security plug-in code should be changed to use either the `db2sec_con_details_2` structure and the `DB2SEC_CON_DETAILS_VERSION_2` value, or the `db2sec_con_details_3` structure and the `DB2SEC_CON_DETAILS_VERSION_3` value.

- New plugins should use the `db2sec_con_details_3` structure and the `DB2SEC_CON_DETAILS_VERSION_3` value. If the `db2secGetConDetails` API is called on an IPv4 platform, the client IP address is returned in the `clientIPAddress` field of the `db2sec_con_details_3` structure and if the API is called on an IPv6 platform the client IP address is returned in the `clientIP6Address` field of the `db2sec_con_details_3` structure. The `clientProtocol` field of the connection details structure will be set to one of `SQL_PROTOCOL_TCPIP` (IPv4, with v1 of the structure), `SQL_PROTOCOL_TCPIP4` (IPv4, with v2 of the structure) or `SQL_PROTOCOL_TCPIP6` (IPv6, with v2 or v3 of the structure).
- The structure `db2sec_con_details_3` is identical to the structure `db2sec_con_details_2` except that it contains an additional field (`clientPlatform`) that identifies the client platform type (as reported by the communication layer) using platform type constants defined in `sqlmon.h`, such as `SQLM_PLATFORM_AIX`.
- `db2secServerAuthPluginTerm`
- `db2secValidatePassword`
- `db2secGetAuthIDs`
- `db2secDoesAuthIDExist`
- `db2secFreeToken`
- `db2secFreeErrorMsg`
- The only API that must be resolvable externally is `db2secServerAuthPluginInit`. This API will take a `void *` parameter, which should be cast to either:

```
typedef struct db2secUseridPasswordServerAuthFunctions_1
{
    db2int32 version;
    db2int32 pluginType;

    /* parameter lists left blank for readability
       see above for parameters */
    SQL_API_RC (SQL_API_FN * db2secValidatePassword)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secFreeToken)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();
} userid_password_server_auth_functions;
```

OR

```
typedef struct db2secGssapiServerAuthFunctions_1
{
    db2int32 version;
    db2int32 pluginType;
    gss_buffer_desc serverPrincipalName;
    gss_cred_id_t ServerCredHandle;
    SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();

    /* GSS-API specific functions
       refer to db2secPlugin.h for parameter list*/
    OM_uint32 (SQL_API_FN * gss_accept_sec_context )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_display_name )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
```

```

OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);

} gssapi_server_auth_functions;

```

You should use the `db2secUseridPasswordServerAuthFunctions_1` structure if you are writing a user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use the `db2secGssapiServerAuthFunctions_1` structure.

db2secClientAuthPluginInit API - Initialize client authentication plug-in

Initialization API, for the client authentication plug-in, that the DB2 database manager calls immediately after loading the plug-in.

API and data structure syntax

```

SQL_API_RC SQL_API_FN db2secClientAuthPluginInit
( db2int32 version,
  void *client_fns,
  db2secLogMessage *logMessage_fn,
  char **errorMsg,
  db2int32 *errorMsgLen );

```

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 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows, 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,
  db2int32 *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 *errmsglen );
```

db2secDoesAuthIDExist API parameters**authid**

Input. The authid to validate. This is upper-cased, with no trailing blanks.

authidlen

Input. Length in bytes of the **authid** parameter value.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secDoesAuthIDExist API execution is not successful.

errmsglen

Output. A pointer to an integer that indicates the length of the error message string in **errmsg** parameter.

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 *errmsglen);
```

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.

errmsglen

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 *errmsglen );
```

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.

errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

db2secGenerateInitialCred API - Generate initial credentials

The db2secGenerateInitialCred API obtains the initial GSS-API credentials based on the user ID and password that are passed in.

For Kerberos, this is the ticket-granting ticket (TGT). The credential handle that is returned in **pGSSCredHandle** parameter is the handle that is used with the gss_init_sec_context API and must be either an INITIATE or BOTH credential. The db2secGenerateInitialCred API is only called when a user ID, and possibly a password are supplied. Otherwise, the DB2 database manager specifies the value GSS_C_NO_CREDENTIAL when calling the gss_init_sec_context API to signify that the default credential obtained from the current login context is to be used.

API and data structure syntax

```
SQL_API_RC ( SQL_API_FN *db2secGenerateInitialCred)
( const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 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 *errmsglen );

```

db2secGenerateInitialCred API parameters

userid

Input. The user ID whose password is to be verified on the database server.

useridlen

Input. Length in bytes of the **userid** parameter value.

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 for Linux, UNIX, and Windows. 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 usernamespacestype,
  const char *dbname,
  db2int32 dbnamelen,
  void **token,
  char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32 *SystemAuthIDlen,
  char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32 *InitialSessionAuthIDlen,
  char username[DB2SEC_MAX_USERID_LENGTH],
  db2int32 *usernamelen,
  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.

usernamespacestype

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_USERSPACE_LENGTH],
  db2int32 *userspacelen,
  db2int32 *userspacetype,
  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.

userspace

Output. The namespace of the user ID.

userspacelen

Output. Length in bytes of the **userspace** parameter value. Under the

limitation that the **usernamespace** parameter must be set to the value `DB2SEC_NAMESPACE_SAM_COMPATIBLE` (defined in `db2secPlugin.h`), the maximum length currently supported is 15 bytes.

usernamespace

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`.

dbnameLen

Input. Length in bytes of the **dbname** parameter value.

token

Output. This is a pointer to data allocated by the plug-in that it might pass to subsequent authentication calls in the plug-in, or possibly to the group retrieval plug-in. The structure of this data is determined by the plug-in writer.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the `db2secGetDefaultLoginContext` API execution is not successful.

errmsgLen

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

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 *errmsgLen );
```

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.

errorMsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the `db2secProcessServerPrincipalName` API execution is not successful.

errormsgLen

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errorMsg** 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 **errorMsg,
  db2int32 *errormsglen);
```

db2secRemapUserid API parameters

userid

Input or output. The user ID to be remapped. If there is an input user ID value, then the API must provide an output user ID value that can be the same or different from the input user ID value. If there is no input user ID value, then the API should not return an output user ID value.

useridlen

Input or output. Length in bytes of the **userid** parameter value.

userspace

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 **userspace** will be used by the DB2 database manager only for CLIENT type authentication and is disregarded for other authentication types.

usernamepacelen

Input or output. Length in bytes of the **usernamepacelen** parameter value. Under the limitation that the **usernamepacetype** parameter must be set to the value `DB2SEC_NAMESPACE_SAM_COMPATIBLE` (defined in `db2secPlugin.h`), the maximum length currently supported is 15 bytes.

usernamepacetype

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.

errmsg1en

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 *errmsglen );
```

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 **plugintype** parameter should be set to one of `DB2SEC_PLUGIN_TYPE_USERID_PASSWORD`, `DB2SEC_PLUGIN_TYPE_GSSAPI`, or `DB2SEC_PLUGIN_TYPE_KERBEROS`. Other values can be defined in future versions of the API.

getConDetails_fn

Input. Pointer to the `db2secGetConDetails` API, which is implemented by DB2. The `db2secServerAuthPluginInit` API can call the `db2secGetConDetails` API in any one of the other authentication APIs to obtain details related to the database connection. These details include information about the

communication mechanism associated with the connection (such as the IP address, in the case of TCP/IP), which the plug-in writer might need to reference when making authentication decisions. For example, the plug-in could disallow a connection for a particular user, unless that user is connecting from a particular IP address. The use of the `db2secGetConDetails` API is optional.

If the `db2secGetConDetails` API is called in a situation not involving a database connection, it returns the value `DB2SEC_PLUGIN_NO_CON_DETAILS`, otherwise, it returns 0 on success.

The `db2secGetConDetails` API takes two input parameters; `pConDetails`, which is a pointer to the `db2sec_con_details_<version_number>` structure, and `conDetailsVersion`, which is a version number indicating which `db2sec_con_details` structure to use. Possible values are `DB2SEC_CON_DETAILS_VERSION_1` when `db2sec_con_details1` is used or `DB2SEC_CON_DETAILS_VERSION_2` when `db2sec_con_details2`. The recommended version number to use is `DB2SEC_CON_DETAILS_VERSION_2`.

Upon a successful return, the `db2sec_con_details` structure (either `db2sec_con_details1` or `db2sec_con_details2`) will contain the following information:

- The protocol used for the connection to the server. The listing of protocol definitions can be found in the file `sqlenv.h` (located in the `include` directory) (`SQL_PROTOCOL_*`). This information is filled out in the `clientProtocol` parameter.
- The TCP/IP address of the inbound connect to the server if the `clientProtocol` is `SQL_PROTOCOL_TCPIP` or `SQL_PROTOCOL_TCPIP4`. This information is filled out in the `clientIPAddress` parameter.
- The database name the client is attempting to connect to. This will not be set for instance attachments. This information is filled out in the `dbname` and `dbnameLen` parameters.
- A connection information bit-map that contains the same details as documented in the `connection_details` parameter of the `db2secValidatePassword` API. This information is filled out in the `connect_info_bitmap` parameter.
- The TCP/IP address of the inbound connect to the server if the `clientProtocol` is `SQL_PROTOCOL_TCPIP6`. This information is filled out in the `clientIP6Address` parameter and it is only available if `DB2SEC_CON_DETAILS_VERSION_2` is used for `db2secGetConDetails` API call.

`logMessage_fn`

Input. A pointer to the `db2secLogMessage` API, which is implemented by the DB2 database manager. The `db2secClientAuthPluginInit` API can call the `db2secLogMessage` API to log messages to the `db2diag` log files for debugging or informational purposes. The first parameter (`level`) of `db2secLogMessage` API specifies the type of diagnostic errors that will be recorded in the `db2diag` log files and the last two parameters 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      **errorMsg,
      db2int32 *errorMsgLen );
```

db2secValidatePassword API parameters

userid

Input. The user ID whose password is to be verified.

useridlen

Input. Length in bytes of the **userid** parameter value.

usernamespace

Input. The namespace from which the user ID was obtained.

usernamespaceLen

Input. Length in bytes of the **usernamespace** parameter value.

usernamespaceType

Input. The type of namespace. Valid values for the **usernamespaceType** parameter (defined in db2secPlugin.h) are:

- DB2SEC_NAMESPACE_SAM_COMPATIBLE Corresponds to a username style like domain\myname
- DB2SEC_NAMESPACE_USER_PRINCIPAL Corresponds to a username style like myname@domain.ibm.com

Currently, the DB2 database system only supports the value DB2SEC_NAMESPACE_SAM_COMPATIBLE. When the user ID is not available, the **usernamespaceType** parameter is set to the value DB2SEC_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. 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 39. 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.
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.

Table 39. Required APIs and Definitions for GSS-API authentication plug-ins (continued)

API type	API name	Description
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

The DB2 for Linux, UNIX, and Windows database manager provides customers and vendors the capability to review communication buffers. External trusted shared libraries are used to access the communication buffers, which flow between clients and database servers, before they are sent and received. These external libraries are known as communication buffer exit libraries.

With communication buffer exit libraries, you can examine communication buffers in order to provide solutions such as auditing or other security solutions based on the contents of the buffers. DB2 for Linux, UNIX, and Windows provides access to each buffer received from clients, as well as each buffer about to be sent to clients. Buffers are provided before they are encrypted with either `DATA_ENCRYPT` authentication or SSL. DB2 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows also makes available identity information, including the username 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 username, but rather generic tickets from which the database manager derives the username. 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, as well as 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 for Linux, UNIX, and Windows does provide samples of libraries in the `sqllib/samples/security/commexit` directory. You might choose to develop your own libraries using the samples as a guide.

Communication buffer exit library deployment

Certain considerations must be taken with the deployment of a communication buffer exit library.

In typical scenarios communication buffer exit libraries are provided by vendors. In these scenarios, the deployment of communication buffer exit libraries are handled by the vendor supplied installation scripts. The deployment steps are outlined here so you can deploy your own library if you choose to do so.

Communication buffer exit library location

Communication buffer exit libraries must exist in specific directories.

The database manager looks for communication buffer 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\instance_name\commexit

Note: On Windows platforms, the subdirectories instance_name and commexit are not created automatically. The instance owner must manually create them.

Communication buffer exit library naming conventions and permissions

Communication buffer exit libraries must adhere to platform-specific naming and permission rules.

The maximum length of a communication buffer 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 platform:

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 platform:

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 communication buffer exit library extensions on a library called `mycommexit` on all platforms:

- 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 buffer 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 updating the database manager configuration.

The following example shows the updating of the database manager configuration on a Windows 64-bit system setting the `mycommexit64.dll` library as the communication buffer 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 buffer exit libraries outside of DB2 pureScale environments

The steps outlined in this task are typically performed by third party supplied installation scripts. The steps are outlined to help you enable a communication buffer exit library you develop.

Before you begin

You must have `SYSADM` authority to perform the steps in this task.

Restrictions

The communication buffer exit library files must follow strict file permission guidelines. For more information about these guidelines, see the related concepts.

Procedure

To enable a communication buffer exit library:

1. Stop the database manager. To stop the database manager, run the `db2stop` command.
2. Copy the communication buffer exit library file to the correct directory. For more details on the required location of communication buffer 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 buffer exit libraries in DB2 pureScale environments

The steps outlined in this task are typically performed by third party supplied installation scripts. The steps are outlined to help you enable a communication buffer exit library you develop.

About this task

By making use of a communication buffer 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 buffer exit library files must follow strict file permission guidelines. For more information about these guidelines, see the related concepts.

Procedure

To enable a communication buffer exit library:

1. Copy the communication buffer exit library that contains the version number in file name to the correct directory. For more details on the required location of communication buffer 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 buffer exit library problem determination

Some options are available to help diagnose problems with a communication buffer exit library.

The communication buffer exit library is not provided as part of DB2 for Linux, UNIX, and Windows. Rather, it is a library 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 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 to determine what problem determination steps must be taken.

An interface to write to the `db2diag` log files is available to communication buffer exit libraries. The `db2diag` log files can be checked if there are concerns with how the library is functioning.

If there are concerns about the performance of the communication buffer 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 buffer exit library development

Certain considerations must be taken with the development of a communication buffer exit library.

In typical scenarios communication buffer exit libraries are provided by vendors. In these scenarios, the development of communication buffer exit libraries is handled by the vendor. You can develop your own library if you choose to do so.

How a communication buffer exit library is loaded

When the database manager is started, the communication buffer 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 buffer 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 invoking the library 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 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 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 being returned.

The function `db2commexitInit` must be declared extern "C" if the library is compiled as C++.

Communication buffer exit library APIs

APIs are implemented in the communication buffer exit library.

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.

This function must be declared extern "C" if the library is compiled as C++.

This function is not required to be thread-safe, 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 **errormsg,
    db2int32 *errormsglen
);
```

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.

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.

LogMessage_fn

Input. A pointer to the db2commexitLogMessage API, which is implemented by the DB2 database system. 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 DB2SEC_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 allocated by the plug-in that can 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.

db2commexitTerm API - Termination

This function frees resources used by the communication buffer exit library.

This API is called by the database manager just before it unloads the communication buffer exit library during **db2stop** processing. The API must be implemented in a manner so it does a proper cleanup of any resources the library holds. For instance, the API must free any memory allocated by the library, close files that are still open, and close network connections. The library is responsible for tracking these resources in order to free them.

This function is not required to be thread-safe 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 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`.

errmsgslen

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.

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 identifying 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**. Once 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 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.

db2commexitDeregister API - Deregistration

This function releases the agent from the connection with which it was associated.

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 identifying 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 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.

db2commexitRecv API - Receive

This function is called for each buffer that the database manager receives from a client.

This function is called by the database manager immediately after receiving 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 identifying 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 as well as 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 this 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 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.

This function is called by the database manager immediately before sending a communication buffer 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 identifying 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 as well as 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 have passed as 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 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.

db2commexitUserIdentity API - User identity

This function is called to provide the identity of the user for the current socket.

This function is called to inform the communication buffer exit library of the username 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 username 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 to perform 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 ** errmsg,
    db2int32 * errmsglen
);
```


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 will not access the memory pointed to by this parameter.

pCommInfo

Input. A pointer to a structure that contains information identifying 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 username 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.

errorMsg

Output. A pointer to the address of an ASCII error message string 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.

db2commexitFreeErrorMsg API - Free error message memory

This function frees the memory used to hold an error message from a previous API call.

API header file

`db2commexit.h`

API and data structure syntax

```
SQL_API_RC ( SQL_API_FN * db2commexitFreeErrorMsg )  
( char * errorMsg );
```

db2commexitFreeErrorMsg API Parameters

errorMsg

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 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      **errorMsg,  
        db2int32 *errormsglen  
    );  
  
    SQL_API_RC ( SQL_API_FN * db2commexitRegister )  
    (  
        void                ** ppConnectionContext,  
        const db2commexitCommInfo_v1 * pCommInfo,  
        db2int32             state,  
        db2int64             * pReservedFlags,  
        char                 ** errorMsg,  
        db2int32             * errormsglen  
    );  
  
    SQL_API_RC ( SQL_API_FN * db2commexitDeregister )  
    (  
        void                * pConnectionContext,  
        const db2commexitCommInfo_v1 * pCommInfo,  
        db2int32             state,  
        db2int64             * pReservedFlags,  
        char                 ** errorMsg,  
        db2int32             * errormsglen  
    );  
  
    SQL_API_RC ( SQL_API_FN * db2commexitRecv )  
    (  
        void                * pConnectionContext,  
        const db2commexitCommInfo_v1 * pCommInfo,  
        const db2commexitBuffer * pBuffer,  
        db2int64             * pReservedFlags,  
        char                 ** errorMsg,  
        db2int32             * errormsglen  
    );  
  
    SQL_API_RC ( SQL_API_FN * db2commexitSend )  
    (  
        void                * pConnectionContext,  
        const db2commexitCommInfo_v1 * pCommInfo,  
        const db2commexitBuffer * pBuffer,  
    );  
};
```

```

        db2int64          * pReservedFlags,
        char             ** errormsg,
        db2int32        * errormsglen
    );

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
);

SQL_API_RC ( SQL_API_FN * db2commexitFreeErrorMsg )
(
    char * errormsg
);

};

```

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 reserved1;
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 buffer exit library buffer structure

The buffer structure is the structure that is passed as input to the `db2commexitSend` and `db2commexitRecv` 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 buffer exit library API versions

The DB2 database system supports version numbering of the communication buffer exit library APIs. These version numbers are integers starting 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. If the

library can support a higher API version, it must return function pointers for the version that the database manager requests. If the library supports only a lower version of the API, the library must define function pointers for the version that it supports. In either situation, the library initialization function must return the version number for the API it is supporting in the version field of the functions structure.

The version numbers of the communication buffer 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 buffer exit library error handling and return codes

When an error occurs in a communication buffer 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 buffer 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 logging 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 40. Return codes that a communication buffer 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 buffer exit library development restrictions

Certain restrictions and considerations must be taken when developing a communication buffer exit library.

Restrictions

C-linkage

The communication buffer exit library must be written in C/C++ and linked with C-linkage. Header files providing the prototypes, data structures required to implement the libraries, and error code definitions are provided only for C/C++. The function `db2commexitInit` must be declared extern "C" if the library is compiled as C++.

Signal handlers

The communication buffer 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 could seriously interfere with the ability to report and recover from errors.

Exceptions

The communication buffer exit library APIs must not throw C++ exceptions. Such exceptions can interfere with database manager error handling.

Thread-safe

The communication buffer exit library functions must be thread-safe. The `db2commexitInit` and `db2commexitTerm` functions are the only APIs that do not have this requirement.

Exit handlers

The communication buffer exit library must not install exit handlers or `pthread_atfork` handlers. The use of exit handlers is not supported because the communication buffer exit library is unloaded before the database manager process exits.

Fork/threads

The communication buffer exit library must not call `fork` or create new threads because this situation can lead to undefined behavior such as traps in the database manager.

Library dependencies

On Linux and UNIX, the communication buffer 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 additional libraries, unless any dependent libraries are accessible through other methods, such as:

- they exist in `/lib` or `/usr/lib`.
- the directories they reside in are specified OS-wide (such as in the `ld.so.conf` file on Linux).
- they are specified in the `RPATH` in the library itself.

Symbol collisions

When possible, communication buffer exit library might 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 libraries 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 platform. A 32-bit communication buffer exit library must be enabled on a 32-bit database manager, and a 64-bit communication buffer 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 buffer exit library. Much of the interaction does not occur over standard communication channels and does not fit the model 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 buffer exit library.

Communication buffers must not be manipulated

It is expected that the communication buffer exit library does not manipulate or change the buffers that it is passed.

Rolling updates support

DB2 for Linux, UNIX, and Windows supports updating the fix pack level of individual members in DB2 pureScale environments without stopping other members. This is known as rolling updates. Similarly, as outlined in the deployment section for the communication buffer exit library, it is possible to update the level of the library used on individual members. It is possible that two different versions of the communication buffer exit library might be running simultaneously on two different members, with each member potentially at a different fix pack level. The communication buffer exit library must tolerate such conditions without error.

Communication buffer exit library API calling sequences

API calling sequences might be different depending on specific scenarios.

The following topics outline specific scenarios you must be aware of when developing communication buffer exit libraries. These 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 connecting to the database manager, issuing some SQL, and then disconnecting.

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.

Once the database manager receives the SQL connect statement from the client, it implicitly drives an internal connect reset before continuing with the connect. There is no change to the status of the socket, these are regular requests and replies. 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 started when parsing the buffer. 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.

API calling sequence - Trusted context and switch user

This scenario is similar to connecting without a connect reset. The difference being the client requests a trusted context switch user rather than sending a new SQL connect request.

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.

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.

Once 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. Once the idle socket has data to read, a dispatcher finds an idle agent to handle it. Over the life of an SQL connection, there may be multiple agents handling the clients request. 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, which will likely be 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.

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.

Considerations for setting the target logical node

Considerations must be taken when setting the target logical node with the *DB2NODE* variable or with the **SET CLIENT** command.

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 will reflect the connected member.

- The outbound application id at the connected member is the same as the inbound application id at the remote member.

Once the application IDs have been 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 for Linux, UNIX, and Windows acts as a connect gateway, the communication buffer exit library is called in the same manner as a standard connection. Once 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 handling of communications protected with the authentication type DATA_ENCRYPT requires special mention. Unlike SSL, the encryption and decryption necessary to support DATA_ENCRYPT is performed by the database manager after a receive from the client and before sending a reply to the client.

Receive and DATA_ENCRYPT

When receiving an encrypted DSS from the client, the buffer is decrypted as needed by the database manager. That is, the whole buffer is not decrypted all at once. 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, and not 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 encrypting a DSS reply to the client, 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 hit 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 41. 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	
METHOD_BODY	X	X	X
MODULE	X	X	X
NICKNAME	X	X	X

Table 41. Audit Record Object Types Based on Audit Events (continued)

Object type	CHECKING events	OBJMAINT events	SECMAINT events
NODEGROUP	X	X	
NONE	X	X	X
OPTIMIZATION PROFILE	X		
PACKAGE	X	X	X
PACKAGE CACHE	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 42. 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 327.
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.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.
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.
Client Workstation Name	VARCHAR(255)	The value of the CURRENT CLIENT_WRKSTNNAME special register at the time the audit event occurred.

Table 42. Audit Record Layout for AUDIT Events (continued)

NAME	FORMAT	DESCRIPTION
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(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION 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"> • N = Nickname • S = MQT • T = Table (untyped) • i = Authorization ID • g= Authority • x = Trusted context • blank = Database
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"> • U = User • G = Group • R = Role
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"> • B-Both • F-Failure • N-None • S-Success
Checking Status	CHAR(1)	The status of the CHECKING category in an audit policy. Possible values are: <ul style="list-style-type: none"> • B-Both • F-Failure • N-None • S-Success

Table 42. 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"> • B-Both • F-Failure • N-None • S-Success
Execute Status	CHAR(1)	The status of the EXECUTE category in an audit policy. Possible values are: <ul style="list-style-type: none"> • B-Both • F-Failure • N-None • S-Success
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"> • Y-WITH DATA • N-WITHOUT DATA
Objmaint Status	CHAR(1)	The status of the OBJMAINT category in an audit policy. Possible values are: <ul style="list-style-type: none"> • B-Both • F-Failure • N-None • S-Success
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"> • B-Both • F-Failure • N-None • S-Success
Validate Status	CHAR(1)	The status of the VALIDATE category in an audit policy. Possible values are: <ul style="list-style-type: none"> • B-Both • F-Failure • N-None • S-Success
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 db2audit configure command.
Archive Path	VARCHAR(1024)	The path to the archived audit logs specified on the db2audit configure command

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 43. 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 327.
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.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.
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.

Table 43. Audit record layout for CHECKING events (continued)

NAME	FORMAT	DESCRIPTION
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".
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	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.

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.

Table 44. Audit Record Layout for OBJMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.
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.
Alter Action	VARCHAR(32)	Specific Alter operation Possible values include: <ul style="list-style-type: none"> • ADD_PROTECTED_COLUMN • ADD_COLUMN_PROTECTION • DROP_COLUMN_PROTECTION • ADD_ROW_PROTECTION • ADD_SECURITY_POLICY • ADD_ELEMENT • ADD COMPONENT • USE GROUP AUTHORIZATIONS • IGNORE GROUP AUTHORIZATIONS • USE ROLE AUTHORIZATIONS • IGNORE ROLE AUTHORIZATIONS • OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL • RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
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.

Table 44. Audit Record Layout for OBJMAINT 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(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION 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.

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 45. 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

Table 45. Audit Record Layout for SECMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
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 327.
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.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.
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 45. 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"> • ADD_DEFAULT_ROLE • DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER • DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION <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"> • ADD_DEFAULT_ROLE • DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER • DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION
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"> • ADD_DEFAULT_ROLE • DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER, DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION

Table 45. 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"> • ADD_DEFAULT_ROLE • DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER • DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION
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"> • ADD_DEFAULT_ROLE, DROP_DEFAULT_ROLE • ALTER_DEFAULT_ROLE • ADD_USER • DROP_USER • ALTER_USER_ADD_ROLE • ALTER_USER_DROP_ROLE • ALTER_USER_AUTHENTICATION
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"> • READ • WRITE • ALL <p>The access type for which a security policy is altered. Possible values:</p> <ul style="list-style-type: none"> • USE GROUP AUTHORIZATIONS • IGNORE GROUP AUTHORIZATIONS • USE ROLE AUTHORIZATIONS • IGNORE ROLE AUTHORIZATIONS • OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL • RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
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 45. 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(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.

SECMAINT privileges or authorities

The following list shows the possible SECMAINT privileges or authorities.

0x00000000000000000000000000000001 Control Table

Control privilege granted or revoked on or from a table or view.

0x00000000000000000000000000000002 ALTER

Privilege granted or revoked to alter a table or sequence.

0x00000000000000000000000000000004 ALTER with GRANT

Privilege granted or revoked to alter a table or sequence with granting of privileges allowed.

0x00000000000000000000000000000008 DELETE TABLE

Privilege granted or revoked to drop a table or view.

0x00000000000000000000000000000010 DELETE TABLE with GRANT

Privilege granted or revoked to drop a table with granting of privileges allowed.

0x00000000000000000000000000000020 Table Index

Privilege granted or revoked on or from an index.

0x00000000000000000000000000000040 Table Index with GRANT

Privilege granted or revoked on or from an index with granting of privileges allowed.

0x00000000000000000000000000000080 Table INSERT

Privilege granted or revoked on or from an insert on a table or view.


```

timestamp=1998-06-24-11.54.04.129923;
category=SYSADMIN;
audit event=DB2AUDIT;
event correlator=1;
event status=0;
userid=boss;authid=BOSS;
application id=*LOCAL.boss.980624155404;
application name=db2audit;

```

Table 46. 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 327.
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.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.
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.
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.

Table 46. Audit Record Layout for SYSADMIN Events (continued)

NAME	FORMAT	DESCRIPTION
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(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.

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 47. 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.

Table 47. Audit Record Layout for VALIDATE Events (continued)

NAME	FORMAT	DESCRIPTION
Execution ID	VARCHAR(1024)	Execution ID in use at the time of the audit event.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.
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.
Trusted Context Name	VARCHAR(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The name of the role inherited through the trusted context.

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 48. 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 327.
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.
Original User ID	VARCHAR(1024)	The value of the CLIENT_ORIGUSERID global variable at the time the audit event occurred.
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.

Table 48. Audit Record Layout for CONTEXT Events (continued)

NAME	FORMAT	DESCRIPTION
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(128)	The name of the trusted context associated with the trusted connection.
Connection Trust Type	INTEGER	Possible values are: IMPLICIT_TRUSTED_CONNECTION EXPLICIT_TRUSTED_CONNECTION
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.

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.prodrig.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
  min_dec_div_3=NO
  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 49. 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 327.
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)	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.

Table 49. Audit Record Layout for EXECUTE Events (continued)

NAME	FORMAT	DESCRIPTION
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(128)	The name of the trusted context associated with the trusted connection.
Connection Trust type	INTEGER	Possible values are IMPLICIT_TRUSTED_CONNECTION and EXPLICIT_TRUSTED_CONNECTION.
Role Inherited	VARCHAR(128)	The role inherited through a trusted connection.
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

Table 49. Audit Record Layout for EXECUTE Events (continued)

NAME	FORMAT	DESCRIPTION
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).
Activity Type	VARCHAR(32)	The type of activity. Possible values are: <ul style="list-style-type: none"> • READ_DML • WRITE_DML • DDL • CALL • NONE
Statement Text	CLOB(8M)	Text of the SQL or XQuery statement, if applicable.
Statement Isolation Level	CHAR(8)	The isolation value in effect for the statement while it was being run. Possible values are: <ul style="list-style-type: none"> • NONE (no isolation specified) • UR (uncommitted read) • CS (cursor stability) • RS (read stability) • RR (repeatable read)
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

Table 49. Audit Record Layout for EXECUTE Events (continued)

NAME	FORMAT	DESCRIPTION
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"> • The enforcement of constraints after a successful delete operation • The processing of triggered SQL statements from activated inlined triggers <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>
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.

Table 49. Audit Record Layout for EXECUTE Events (continued)

NAME	FORMAT	DESCRIPTION
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"> • 0 if the statement value was specified as assigned by the indicator value, • -1 if NULL was specified by the indicator value, • -5 if DEFAULT was specified by the indicator value, • -7 if UNASSIGNED was specified by the indicator value.
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.

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 SECURITY POLICY
- ALTER_USER_ADD_ROLE
- ALTER_USER_AUTHENTICATION
- ALTER_USER_DROP_ROLE
- DROP_DEFAULT_ROLE
- DROP_USER
- GRANT
- IMPLICIT_GRANT
- IMPLICIT_REVOKE
- 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
- 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
- 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_DB2
- STOP_DB2
- UNCATALOG_DB
- UNCATALOG_DCS_DB
- UNCATALOG_NODE
- UNLOAD_TABLE
- UPDATE_ADMIN_CFG
- UPDATE_CLI_CONFIGURATION
- 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:

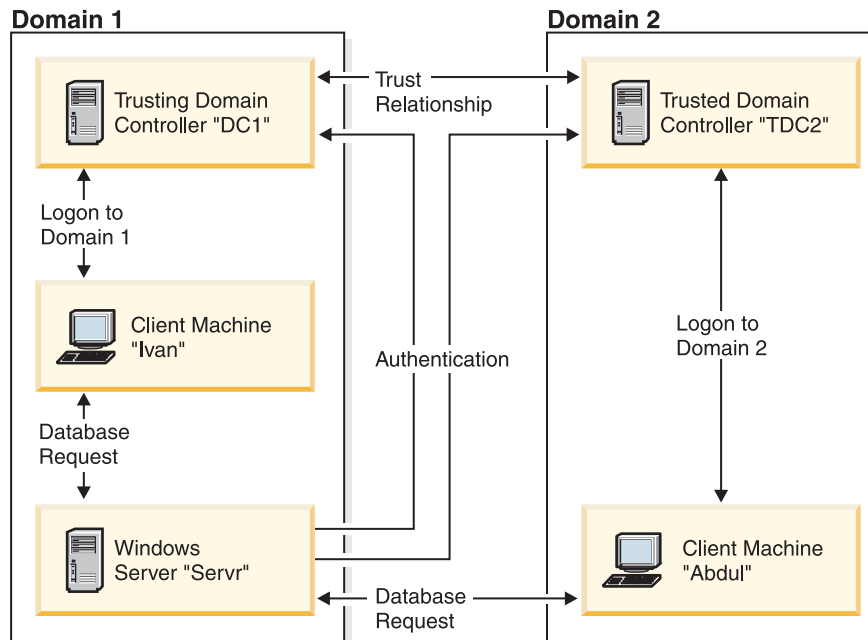


Figure 7. Authentication Using Windows Domains

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 DB2NTSECSERVER 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 50. Successful Connection Using a Domain Controller

Domain1	Domain2
A trust relationship exists with Domain2.	<ul style="list-style-type: none"> • A trust relationship exists with Domain1. • The local or global group grp2 is defined. • The user name id2 is defined. • The user name id2 is part of grp2.
The DB2 server runs in this domain. The following DB2 commands are issued from it: <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 and DB2USERS are the default group names; optionally, you can specify different names for these groups at installation time (if you select silent install, you can change these names within the install response file). If you choose to use groups that already exist on your system, be aware that the privileges of these groups will be modified. They will be given the privileges, as required, listed in the following table. 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, such as SYSADM, SYSMANT, and SYSCTRL. However, instead of using the default Administrator's group, your database administrator can use the DB2ADMNS group for one or all of the DB2 authority levels, at the discretion of the installer or administrator. It is recommended that if you are specifying a SYSADM group, then that should be the DB2ADMNS group. This can be established during installation or subsequently, 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 Vista, 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 51. 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 (required before Windows XP in order to execute the LogonUser API for authentication purposes)
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 Windows 2008 and Windows Vista or higher: User Access Control feature

The User Access Control (UAC) feature of Windows 2008, Windows Vista, and Windows 7 impacts the DB2 database system in the following ways.

Starting applications with full administrative privileges

On Windows 2008, Windows Vista, and 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 2008, Windows Vista, and 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 2008, Windows Vista, and 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 2008, Windows Vista, and 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**.

Appendix A. Overview of the DB2 technical information

DB2 technical information is available in multiple formats that can be accessed in multiple ways.

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
 - Topics (Task, concept and reference topics)
 - Sample programs
 - Tutorials
- DB2 books
 - PDF files (downloadable)
 - PDF files (from the DB2 PDF DVD)
 - printed books
- Command-line help
 - Command help
 - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at [ibm.com](http://www.ibm.com). Access the DB2 Information Management software library site at <http://www.ibm.com/software/data/sw-library/>.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this email address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss. English and translated DB2 Version 10.1 manuals in PDF format can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg27009474.

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

Table 52. DB2 technical information

Name	Form Number	Available in print	Last updated
<i>Administrative API Reference</i>	SC27-3864-00	Yes	April, 2012
<i>Administrative Routines and Views</i>	SC27-3865-01	No	January, 2013
<i>Call Level Interface Guide and Reference Volume 1</i>	SC27-3866-01	Yes	January, 2013
<i>Call Level Interface Guide and Reference Volume 2</i>	SC27-3867-01	Yes	January, 2013
<i>Command Reference</i>	SC27-3868-01	Yes	January, 2013
<i>Database Administration Concepts and Configuration Reference</i>	SC27-3871-01	Yes	January, 2013
<i>Data Movement Utilities Guide and Reference</i>	SC27-3869-01	Yes	January, 2013
<i>Database Monitoring Guide and Reference</i>	SC27-3887-01	Yes	January, 2013
<i>Data Recovery and High Availability Guide and Reference</i>	SC27-3870-01	Yes	January, 2013
<i>Database Security Guide</i>	SC27-3872-01	Yes	January, 2013
<i>DB2 Workload Management Guide and Reference</i>	SC27-3891-01	Yes	January, 2013
<i>Developing ADO.NET and OLE DB Applications</i>	SC27-3873-01	Yes	January, 2013
<i>Developing Embedded SQL Applications</i>	SC27-3874-01	Yes	January, 2013
<i>Developing Java Applications</i>	SC27-3875-01	Yes	January, 2013
<i>Developing Perl, PHP, Python, and Ruby on Rails Applications</i>	SC27-3876-00	No	April, 2012
<i>Developing RDF Applications for IBM Data Servers</i>	SC27-4462-00	Yes	January, 2013
<i>Developing User-defined Routines (SQL and External)</i>	SC27-3877-01	Yes	January, 2013
<i>Getting Started with Database Application Development</i>	GI13-2046-01	Yes	January, 2013

Table 52. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
<i>Getting Started with DB2 Installation and Administration on Linux and Windows</i>	GI13-2047-00	Yes	April, 2012
<i>Globalization Guide</i>	SC27-3878-00	Yes	April, 2012
<i>Installing DB2 Servers</i>	GC27-3884-01	Yes	January, 2013
<i>Installing IBM Data Server Clients</i>	GC27-3883-00	No	April, 2012
<i>Message Reference Volume 1</i>	SC27-3879-01	No	January, 2013
<i>Message Reference Volume 2</i>	SC27-3880-01	No	January, 2013
<i>Net Search Extender Administration and User's Guide</i>	SC27-3895-01	No	January, 2013
<i>Partitioning and Clustering Guide</i>	SC27-3882-01	Yes	January, 2013
<i>Preparation Guide for DB2 10.1 Fundamentals Exam 610</i>	SC27-4540-00	No	January, 2013
<i>Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611</i>	SC27-4541-00	No	January, 2013
<i>pureXML Guide</i>	SC27-3892-01	Yes	January, 2013
<i>Spatial Extender User's Guide and Reference</i>	SC27-3894-00	No	April, 2012
<i>SQL Procedural Languages: Application Enablement and Support</i>	SC27-3896-01	Yes	January, 2013
<i>SQL Reference Volume 1</i>	SC27-3885-01	Yes	January, 2013
<i>SQL Reference Volume 2</i>	SC27-3886-01	Yes	January, 2013
<i>Text Search Guide</i>	SC27-3888-01	Yes	January, 2013
<i>Troubleshooting and Tuning Database Performance</i>	SC27-3889-01	Yes	January, 2013
<i>Upgrading to DB2 Version 10.1</i>	SC27-3881-01	Yes	January, 2013
<i>What's New for DB2 Version 10.1</i>	SC27-3890-01	Yes	January, 2013
<i>XQuery Reference</i>	SC27-3893-01	No	January, 2013

Table 53. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
<i>DB2 Connect Installing and Configuring DB2 Connect Personal Edition</i>	SC27-3861-00	Yes	April, 2012

Table 53. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print	Last updated
DB2 Connect Installing and Configuring DB2 Connect Servers	SC27-3862-01	Yes	January, 2013
DB2 Connect User's Guide	SC27-3863-01	Yes	January, 2013

Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure

To start SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, ? 08003 displays help for the 08003 SQL state, and ? 08 displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

Documentation for other versions of DB2 products is found in separate information centers on ibm.com[®].

About this task

For DB2 Version 10.1 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v10r1>.

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/>.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/>.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/>.

For DB2 Version 9.1 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9/>.

For DB2 Version 8 topics, go to the *DB2 Information Center* URL at: <http://publib.boulder.ibm.com/infocenter/db2luw/v8/>.

Updating the DB2 Information Center installed on your computer or intranet server

A locally installed DB2 Information Center must be updated periodically.

Before you begin

A DB2 Version 10.1 Information Center must already be installed. For details, see the “Installing the DB2 Information Center using the DB2 Setup wizard” topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates update existing Information Center features and languages. One benefit of automatic updates is that the Information Center is unavailable for a shorter time compared to during a manual update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates can be used to update existing Information Center features and languages. Automatic updates reduce the downtime during the update process, however you must use the manual process when you want to add features or languages. For example, a local Information Center was originally installed with both English and French languages, and now you want to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process. In the automatic update process the Information Center incurs an outage to restart the Information Center after the update only.

This topic details the process for automatic updates. For manual update instructions, see the “Manually updating the DB2 Information Center installed on your computer or intranet server” topic.

Procedure

To automatically update the DB2 Information Center installed on your computer or intranet server:

1. On Linux operating systems,
 - a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V10.1` directory.
 - b. Navigate from the installation directory to the `doc/bin` directory.
 - c. Run the `update-ic` script:

```
update-ic
```
2. On Windows operating systems,
 - a. Open a command window.
 - b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `<Program Files>\IBM\DB2 Information Center\Version 10.1` directory, where `<Program Files>` represents the location of the Program Files directory.
 - c. Navigate from the installation directory to the `doc\bin` directory.
 - d. Run the `update-ic.bat` file:

```
update-ic.bat
```

Results

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center updates were not available, a message is added to the log. The log file is located in `doc\eclipse\configuration` directory. The log file name is a randomly generated number. For example, `1239053440785.log`.

Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

About this task

Updating your locally installed *DB2 Information Center* manually requires that you:

1. Stop the *DB2 Information Center* on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

Note: If your environment requires installing the *DB2 Information Center* updates on a machine that is not connected to the internet, mirror the update site to a local file system by using a machine that is connected to the internet and has the *DB2 Information Center* installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.

If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.

3. Stop the stand-alone Information Center, and restart the *DB2 Information Center* on your computer.

Note: On Windows 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

Procedure

To update the *DB2 Information Center* installed on your computer or intranet server:

1. Stop the *DB2 Information Center*.
 - On Windows, click **Start > Control Panel > Administrative Tools > Services**. Then right-click **DB2 Information Center** service and select **Stop**.
 - On Linux, enter the following command:
`/etc/init.d/db2icdv10 stop`
2. Start the Information Center in stand-alone mode.
 - On Windows:
 - a. Open a command window.

- b. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the *Program_Files\IBM\DB2 Information Center\Version 10.1* directory, where *Program_Files* represents the location of the Program Files directory.
- c. Navigate from the installation directory to the *doc\bin* directory.
- d. Run the *help_start.bat* file:


```
help_start.bat
```
- On Linux:
 - a. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the */opt/ibm/db2ic/V10.1* directory.
 - b. Navigate from the installation directory to the *doc/bin* directory.
 - c. Run the *help_start* script:


```
help_start
```

The systems default Web browser opens to display the stand-alone Information Center.

3. Click the **Update** button (🔄). (JavaScript must be enabled in your browser.) On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check that the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
 - On Windows, navigate to the *doc\bin* directory within the installation directory, and run the *help_end.bat* file:


```
help_end.bat
```

Note: The *help_end* batch file contains the commands required to safely stop the processes that were started with the *help_start* batch file. Do not use **Ctrl-C** or any other method to stop *help_start.bat*.
 - On Linux, navigate to the *doc/bin* directory within the installation directory, and run the *help_end* script:


```
help_end
```

Note: The *help_end* script contains the commands required to safely stop the processes that were started with the *help_start* script. Do not use any other method to stop the *help_start* script.
7. Restart the *DB2 Information Center*.
 - On Windows, click **Start > Control Panel > Administrative Tools > Services**. Then right-click **DB2 Information Center** service and select **Start**.
 - On Linux, enter the following command:


```
/etc/init.d/db2icdv10 start
```

Results

The updated *DB2 Information Center* displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 database products. Lessons provide step-by-step instructions.

Before you begin

You can view the XHTML version of the tutorial from the Information Center at <http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/>.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials

To view the tutorial, click the title.

"pureXML®" in pureXML Guide

Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 database products.

DB2 documentation

Troubleshooting information can be found in the *Troubleshooting and Tuning Database Performance* or the Database fundamentals section of the *DB2 Information Center*, which contains:

- Information about how to isolate and identify problems with DB2 diagnostic tools and utilities.
- Solutions to some of the most common problem.
- Advice to help solve other problems you might encounter with your DB2 database products.

IBM Support Portal

See the IBM Support Portal if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the IBM Support Portal at http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

Terms and conditions

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Trademarks: IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements, changes, or both in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to websites not owned by IBM are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licenses of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

Index

Special characters

.NET

- .Net Data Provider clients 67
- GSKit 67
- SSL 67

A

access control

- authentication 7
- column-specific 159
- DBADM (database administration) authority 55
- fine-grained row and column
 - see RCAC 137
- label-based access control 159
- row-specific 159
- tables 53
- views 53

access tokens

- Windows 337

ACCESSCTRL (access control) authority

- details 36
- overview 32

AIX

- authentication methods 218
- configuring transparent LDAP 217

AIX encrypted file system (EFS) 88

ALTER privilege 42, 44

alternate_auth_enc configuration parameter

- encrypting using AES 256-bit algorithm 7

APIs

- communication buffer exit library
 - db2commexitDeregister 283
 - db2commexitFreeErrorMsg 287
 - db2commexitInit 280
 - db2commexitRecv 284
 - db2commexitRegister 282
 - db2commexitSend 285
 - db2commexitTerm 281
 - db2commexitUserIdentity 286
- overview 280

group plug-in

- db2secDoesGroupExist 245
- db2secFreeErrorMsg 246
- db2secFreeGroupListMemory 246
- db2secGetGroupsForUser 247
- db2secGroupPluginInit 250
- db2secPluginTerm 251

group retrieval plug-in 244

security plug-in

- db2secClientAuthPluginInit 257
- db2secClientAuthPluginTerm 258
- db2secDoesAuthIDExist 259
- db2secDoesGroupExist 245
- db2secFreeErrorMsg 246
- db2secFreeGroupListMemory 246
- db2secFreeInitInfo 259
- db2secFreeToken 260
- db2secGenerateInitialCred 260
- db2secGetAuthIDs 262

APIs (continued)

security plug-in (continued)

- db2secGetDefaultLoginContext 264
- db2secGetGroupsForUser 247
- db2secGroupPluginInit 250
- db2secPluginTerm 251
- db2secProcessServerPrincipalName 265
- db2secRemapUserid 266
- db2secServerAuthPluginInit 267
- db2secServerAuthPluginTerm 270
- db2secValidatePassword 270
- overview 243
- user ID/password plug-ins 251

archivepath parameter 97

archiving

- audit log files 97

AUDIT events 327

audit facility

- actions 91
- archive 103
- asynchronous record writing 112
- audit data in tables
 - creating tables 101
 - loading tables 102
- audit events table 298
- authorities 91
- behavior 112
- CHECKING access approval reasons 303
- CHECKING access attempted types 305
- checking events table 302
- CONTEXT events table 320
- error handling 112
- ERRORTYPE parameter 112
- events 91
- EXECUTE events 105, 322
- EXECUTE timestamp 109
- object record types 297
- OBJMAINT events table 308
- overview 91
- policies 93
- privileges 91
- record layouts 297
- record object types 297
- records for EXECUTE events 322
- SECMAINT authorities 314
- SECMAINT events table 310
- SECMAINT privileges 314
- synchronous record writing 112
- SYSADMIN events table 317
- techniques 114
- tips 114
- VALIDATE events table 319

audit logs

- archiving 97, 103
- file names 100
- location 97

audit_buf_sz configuration parameter

- determining timing of writing audit records 112

authentication

- details 2
- domain security 336

- authentication (*continued*)
 - groups 336
 - GSS-API 201
 - ID and password 201
 - Kerberos 14, 201
 - LDAP users 230
 - lookup
 - configuring 217, 220, 222, 223
 - methods 7
 - ordered domain list 340
 - overview 1
 - partitioned databases 13
 - plug-ins
 - API for checking whether authentication ID exists 259
 - API for cleaning up client authentication plug-in resources 258
 - API for cleaning up resources held by db2secGenerateInitialCred API 259
 - API for cleaning up server authentication plug-in resources 270
 - API for getting authentication IDs 262
 - API for initializing client authentication plug-in 257
 - API for initializing server authentication plug-in 267
 - API for validating passwords 270
 - APIs for user ID/ password authentication plug-ins 251
 - deploying 211, 214, 347
 - LDAP 215
 - library locations 205
 - security 201
 - remote clients 13
 - security plug-ins 201
 - two-part user IDs 207
 - 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 225
- authorities
 - access control (ACCESSCTRL) 36
 - audit policy 93
 - data access (DATAACCESS) 37
 - database administration (DBADM) 34, 40
 - explain administration (EXPLAIN) 39
 - implicit schema (IMPLICIT_SCHEMA) 40
 - LOAD 39
 - overview 20, 26
 - removing DBADM from SYSCTRL 30
 - security administrator (SECADM) 33
 - SQL administration (SQLADM) 38
 - system administration (SYSADM) 29
 - system control (SYSCTRL) 30
 - system maintenance (SYSMAINT) 30
 - system monitor (SYSMON) 31
 - workload administration (WLMADM) 39
- authorization IDs
 - details 3
 - implicit authorizations 51
 - LDAP 228
 - security model overview 1
 - SETSESSIONUSER privilege 40

- authorization IDs (*continued*)
 - trusted client 7
 - types 46
- authorization names
 - creating views for privileges information 196
 - retrieving
 - names with DBADM authority 195
 - names with granted privileges 194
 - names with table access authority 195
 - privileges granted to 196

B

- backups
 - encrypting 86
 - security risks 56, 86
- BIND command
 - package re-creation ownership 51
- BIND privilege 43
- BINDADD authority
 - details 32
- binding
 - rebinding invalid packages 49
- built-in views
 - AUTHORIZATIONIDS
 - example 194
 - restricting access 196
 - OBJECTOWNERS
 - restricting access 196
 - PRIVILEGES
 - example 194
 - restricting access 196

C

- certificate authorities
 - digital certificates 72
- CHECKING events 327
- cipher suites 73
- client authentication plug-ins 215
- CLIENT authentication type
 - details 7
- columns
 - LBAC protection
 - adding 177
 - removing 191
 - LBAC-protected
 - dropping 188
 - inserting 182
 - reading 179
 - updating 184
- communication buffer exit library
 - APIs
 - db2commexitDeregister 283
 - db2commexitFreeErrorMsg 287
 - db2commexitInit 280
 - db2commexitRecv 284
 - db2commexitRegister 282
 - db2commexitSend 285
 - db2commexitTerm 281
 - db2commexitUserIdentity 286
- deploying 276
- developing
 - API calling sequences (connection concentrator) 294
 - API calling sequences (no connect reset) 293

- communication buffer exit library (*continued*)
 - developing (*continued*)
 - API calling sequences (normal connect) 293
 - API calling sequences (overview) 293
 - API calling sequences (SET SESSION AUTHORIZATION) 295
 - API calling sequences (trusted context) 294
 - API versions 290
 - buffer structure 290
 - connect gateway 296
 - control over connections 290
 - DATA_ENCRYPT authentication 296
 - error handling 291
 - functions structure 288
 - information structure 289
 - overview 279
 - restrictions 292
 - return codes 291
 - target logical node 295
 - enabling 277, 278
 - library loading 279
 - location 276
 - naming conventions 276
 - overview 275
 - permissions 276
 - problem determination 279
- configuration
 - LDAP
 - plug-ins 225
- CONNECT authority 32
- CONTEXT events 327
- CONTROL privilege
 - details 42
 - implicit 51
 - packages 43
 - views 42
- CREATE DATABASE command
 - RESTRICTIVE option 196
- CREATE ROLE statement
 - creating roles 120
 - granting membership in roles 120
- CREATE TRUSTED CONTEXT statement
 - example 132
- CREATE_EXTERNAL_ROUTINE authority 32
- CREATE_NOT_FENCED_ROUTINE authority 32
- CREATETAB authority 32
- cryptography
 - public key 73

D

- data
 - audit
 - creating tables 101
 - loading into tables 102
 - encrypting 59
 - indirect access 56
 - inserting
 - LBAC-protected 182
 - label-based access control (LBAC)
 - adding protection 177
 - inserting 182
 - overview 177
 - reading 179
 - unprotecting 191
 - updating 184

- data (*continued*)
 - security
 - overview 1
 - system catalog 196
- data at rest 86
- DATAACCESS (data access) authority
 - details 37
 - overview 32
- database authorities
 - granting
 - overview 32
 - overview 32
 - revoking 32
- database directories
 - permissions 6
- Database Encryption Expert 86
- database objects
 - roles 119
- database-level authorities
 - overview 26
- databases
 - accessing
 - default authorities 47
 - default privileges 47
 - implicit privileges through packages 51
 - label-based access control (LBAC) 159
- datapath parameter 97
- DB2 Information Center
 - updating 353, 354
 - versions 352
- DB2_GRP_LOOKUP environment variable 338, 341
- DB2_GRP_LOOKUP registry variable 337
- DB2ADMNS group
 - defining who holds SYSADM authority 341
 - details 342
- db2audit.log file 91
- db2cluster command
 - DB2 cluster services administrator 116
 - security model 116
- DB2COMM registry variable
 - configuring Secure Sockets Layer (SSL) support 60
- DB2LBACRULES LBAC rule set 171
- DB2LDAPSecurityConfig environment variable
 - overview 225
- DB2SECURITYLABEL data type
 - providing explicit values 176
 - viewing as string 176
- DB2USERS user group
 - details 342
- DBADM (database administration) authority
 - controlling access 55
 - details 34
 - overview 32
 - retrieving names 195
- debugging
 - security plug-ins 209
- default privileges 47
- DELETE privilege 42
- digital certificates
 - managing 60
 - overview 72
- distinguished name (DN) 228
- documentation
 - overview 349
 - PDF files 349
 - printed 349
 - terms and conditions of use 356

- domain controller
 - overview 333
- domains
 - ordered domain list 340
 - security
 - authentication 336
 - trust relationships 336
 - Windows 340
- dynamic SQL
 - EXECUTE privilege 51

E

- efsenable command 88
- efskeymgr command 88
- efsmgr command 88
- ENABLE_SSL parameter 225
- encrypted file system (EFS) 88
- encryption
 - data 59
- Encryption Expert 86
- enumeration of groups 338
- error messages
 - security plug-ins 238
- errors
 - switching user 135
 - trusted contexts 135
- ExampleBANK RCAC scenario
 - column masks 156
 - data queries 156
 - database tables 154
 - database users and roles 153
 - introduction 152
 - row permissions 155
 - security policy 153
- ExampleHMO RCAC scenario
 - column masks 144
 - data queries 146
 - data updates 146
 - database tables 140
 - database users and roles 139
 - inserting data 145
 - introduction 138
 - revoke authority 152
 - row permissions 143
 - secure functions 149
 - secure triggers 151
 - security administration 142
 - security policy 139
 - view creation 148
- EXECUTE category
 - audit information 109
 - audit records 322
 - overview 105
 - replaying activities 110
- EXECUTE events 327
- EXECUTE privilege
 - database access 51
 - packages 43
 - routines 45
- EXPLAIN authority
 - details 39
 - overview 32
- explicit trusted connections
 - establishing 127
 - user ID switching 127, 133
- extended Windows security 342

F

- FGAC
 - see RCAC 137
- file names
 - audit logs 100
- fine-grained access control
 - see RCAC 137
- firewalls
 - application proxy 199
 - circuit level 199
 - details 199
 - screening router 199
 - stateful multi-layer inspection (SMLI) 200
- functions
 - privileges 45
 - scalar
 - DECRYPT_BIN 59
 - DECRYPT_CHAR 59
 - ENCRYPT 59
 - GETHINT 59

G

- global group support 335
- GRANT statement
 - example 49
 - implicit authorizations 51
 - overview 49
- group lookup support
 - details 215, 229
- GROUP_BASEDN parameter 225
- GROUP_LOOKUP_ATTRIBUTE attribute 229
- GROUP_LOOKUP_METHOD parameter
 - configuring LDAP plug-in modules 225, 229
- GROUP_OBJECTCLASS parameter 225
- GROUPNAME_ATTRIBUTE parameter 225
- groups
 - access token 337
 - enumeration (Windows) 338
 - names 335
 - roles comparison 125
 - selecting 4
 - user authentication 336
- GSKCapiCmd tool
 - configuring Secure Sockets Layer (SSL) support 60, 67
- GSKit
 - configuring Secure Sockets Layer (SSL) support 60, 67
 - library rules 74
 - process rules 74
 - return codes 76
- GSS-APIs
 - authentication plug-ins 273

H

- handshakes
 - overview 71
- help
 - SQL statements 352
- HP-UX
 - transparent LDAP 222

I

- IBM Database Encryption Expert 86
- IBMLDAPSecurity.ini file 225
- IKEYCMD tool 60, 67
- iKeyman tool 60, 67
- implicit authorization
 - managing 51
- IMPLICIT_SCHEMA (implicit schema) authority
 - details 40
 - overview 32
- INDEX privilege
 - details 44
- indexes
 - privileges
 - overview 44
- INSERT privilege 42
- instance directories 6
- instances
 - authorities 26
 - configuring
 - SSL communications 60

K

- Kerberos authentication protocol
 - enabling 18
 - IBM i compatibility 19
 - mapping 16
 - naming 16
 - overview 14
 - plug-ins
 - creating 19
 - deploying 214
 - principals 16
 - server 7
 - setting up 14
 - System z compatibility 19
 - Windows compatibility 19
- KRB_SERVER_ENCRYPT authentication type 7

L

- label-based access control
 - See LBAC 159
- LBAC
 - credentials 159
 - dropping columns 188
 - inserting data 182
 - overview 20, 159
 - protected tables 159
 - reading data 179
 - removing protection 191
 - rule exemptions
 - details 175
 - effect on security label comparisons 170
 - rule sets
 - comparing security labels 170
 - DB2LBACRULES 171
 - overview 171
 - security administrators 159
 - security labels
 - ARRAY component type 163
 - comparisons 170
 - compatible data types 167
 - components 162
 - creating 167

- LBAC (*continued*)
 - security labels (*continued*)
 - details 167
 - dropping 167
 - granting 167
 - overview 159
 - revoking 167
 - SET component type 163
 - string format 169
 - TREE component type 164
 - security policies
 - adding to a table 177
 - details 161
 - overview 159
 - updating data 184
- LDAP
 - plug-ins 225, 227
 - security plug-ins 215
 - transparent
 - AIX 217
 - HP-UX 222
 - Kerberos 218
 - Linux 220
 - Solaris 223
 - LDAP_HOST parameter 225
- libraries
 - security plug-ins
 - loading in DB2 230
 - restrictions 232
- Linux
 - security 347
 - transparent LDAP 220
- LOAD authority
 - details 39
 - overview 32
- LocalSystem account
 - authorization 29
 - support 342
 - SYSADM authority 341
- logs
 - audit 91

M

- methods
 - privileges 45
- migration
 - roles 126

N

- naming conventions
 - Windows restrictions 335
- NESTED_GROUPS parameter 225
- nicknames
 - privileges
 - indirect through packages 52
- notices 359

O

- objects
 - ownership 20
- OBJMAINT events 327
- ordered domain lists 340

ownership
database objects 20, 193

P

packages
access privileges with queries 51
authorization IDs
derivation 46
use 46
ownership 51
privileges
overview 43
revoking (overview) 49
passwords
changing
Linux 347
maintaining on servers 20
permissions
authorization overview 3
column-specific protection 159
directories 6
row-specific protection 159
plug-ins
group retrieval 244
GSS-API authentication 273
ID authentication 251
LDAP 215
password authentication 251
security
APIs 239, 243
deploying 211, 212, 214, 347
error messages 238
naming conventions 206
restrictions (GSS-API authentication) 274
restrictions (plug-in libraries) 232
restrictions (summary) 234
return codes 235
versions 209
pluggable authentication module
PAM 217, 220, 222, 223
PRECOMPILE command
OWNER option 51
privileges
acquiring through trusted context roles 132
ALTER
sequences 44
tables 42
CONTROL 42
DELETE 42
EXECUTE
routines 45
GRANT statement 49
granting
roles 125
hierarchy 20
INDEX 42
indexes
overview 44
indirect
packages containing nicknames 52
individual 20
information about granted
retrieving 194, 196
INSERT 42
overview 20
ownership 20

privileges (*continued*)
packages
creating 43
implicit 20
planning 3
REFERENCES 42
revoking
overview 49
roles 122
roles 119
schemas 41
SELECT 42
SETSESSIONUSER 40
system catalog
privilege information 193
restricting access 196
table spaces 42
tables 42
UPDATE 42
USAGE
sequences 44
workloads 45
views 42
problem determination
information available 356
security plug-ins 209
tutorials 356
procedures
privileges 45
PUBLIC
database authorities automatically granted 32
public-key cryptography 73

Q

QUIESCE_CONNECT authority 32

R

RCAC
conditions in permissions
conditions in masks
rule management
scalar functions
conditions in permissions
conditions in masks 138
ExampleBANK
see ExampleBANK RCAC scenario 152
ExampleHMO
see ExampleHMO RCAC scenario 138
overview 137
rule management
SQL statements 138
rules 138
scenario
see ExampleBANK RCAC scenario 152
see ExampleHMO RCAC scenario 138
records
audit 91
REFERENCES privilege 42
registry variables
DB2COMM 60
replay
EXECUTE timestamp 110
RESTRICTIVE option of CREATE DATABASE command
denying privileges to PUBLIC 196
return codes
GSKit 76

- REVOKE statement
 - example 49
 - implicit issuance 51
 - overview 49
- roles
 - creating 120
 - details 119
 - hierarchies 122
 - migrating from IBM Informix Dynamic Server 126
 - revoking privileges 122
 - versus groups 125
 - WITH ADMIN OPTION clause 124
- routine invoker authorization IDs 46
- row and column access control
 - see RCAC 137
- rows
 - deleting
 - LBAC-protected data 188
 - inserting
 - LBAC-protected data 182
 - protecting with LBAC 177
 - reading when using LBAC 179
 - removing LBAC protection 191
 - updating
 - LBAC-protected data 184
- rule sets (LBAC)
 - details 171
 - exemptions 175

S

- Savepoint ID field 105
- schemas
 - privileges 41
- SEARCH_DN parameter 225
- SEARCH_PW parameter 225
- SECADM (security administrator) authority
 - details 33
 - overview 32
- SECLABEL scalar function
 - overview 176
- SECLABEL_BY_NAME scalar function
 - overview 176
- SECLABEL_TO_CHAR scalar function
 - overview 176
- SECMAINT events 327
- security
 - APIs (communication buffer exit library) 280
 - authentication 2
 - CLIENT level 7
 - column-specific 159
 - communication buffer exit library
 - API calling sequences 293
 - API calling sequences connection concentrator 294
 - API calling sequences no connect reset 293
 - API calling sequences normal connect 293
 - API calling sequences SET SESSION AUTHORIZATION 295
 - API calling sequences trusted context 294
 - API versions 290
 - buffer structure 290
 - connect gateway 296
 - control over connections 290
 - DATA_ENCRYPT 296
 - deployment 276
 - development 279
 - enabling 277, 278
 - data 1
 - db2extsec command 342
 - disabling extended security 342
 - enabling extended security 342
 - establishing explicit trusted connections 127
 - extended security 342
 - label-based access control (LBAC) 159
 - maintaining passwords on servers 20
 - plug-ins
 - 32-bit considerations 209
 - 64-bit considerations 209
 - API for validating passwords 270
 - APIs 243, 245, 246, 247, 250, 251, 257, 258, 259, 260, 262, 264, 265, 266, 267, 270
 - APIs (group retrieval) 244
 - APIs (GSS-API) 273
 - APIs (user ID/password) 251
 - APIs (versions) 209
 - calling sequence 239
 - deploying 201, 211, 212, 214, 234, 347
 - developing 201
 - enabling 201
 - error messages 238
 - GSS-API (deploying) 212
 - GSS-API (restrictions) 274
 - initialization 230
 - LDAP (Lightweight Directory Access Protocol) 215
 - libraries 205
 - loading 201, 230
 - naming 206
 - overview 201
 - problem determination 209
 - restrictions on libraries 232
 - return codes 235
 - SQLCODES 209
 - SQLSTATES 209
 - two-part user ID support 207
 - risks 56
 - row and column access control
 - fine-grained access control
 - see RCACsee RCAC 137
 - row-specific 159
 - trusted contexts 129
 - UNIX 347
 - Windows
 - domain security 340
 - extended 342
 - overview 333
 - users 341
- security labels (LBAC)
 - ARRAY component type 163
 - compatible data types 167
 - components 162
 - policies
 - details 161
- security (continued)
 - communication buffer exit library (continued)
 - error handlingreturn codes 291
 - functions structure 288
 - information structure 289
 - library loading 279
 - location 276
 - naming conventions 276
 - overview 275
 - permissions 276
 - problem determination 279
 - restrictions 292
 - target logical node 295

- security labels (LBAC) *(continued)*
 - SET component type 163
 - string format 169
 - TREE component type 164
 - use 167
- SELECT privilege 42
- sequences
 - privileges 44
- server authentication plug-ins 215
- SERVER authentication type
 - overview 7
- SERVER_ENCRYPT authentication type
 - overview 7
- session authorization IDs
 - overview 46
- SET ENCRYPTION PASSWORD statement
 - encrypting passwords 59
- SETSESSIONUSER privilege
 - details 40
- Solaris operating systems
 - transparent LDAP 223
- SQL statements
 - authorization IDs 46
 - help
 - displaying 352
- SQLADM (SQL administration) authority
 - details 38
 - overview 32
- SSL
 - CATALOG TCP/IP NODE command 67
 - certificate authorities 72
 - cipher suites 73
 - CLI clients 67
 - CLP clients 67
 - configuring
 - DB2 clients 67
 - DB2 instances 60
 - DB2 Connect 60
 - digital certificates 72
 - embedded SQL clients 67
 - handshake 71
 - protocol 71
 - ssl_cipherspecs configuration parameter
 - specifying cipher suites 60, 73
 - ssl_client_keystash connection parameter
 - configuring SSL 67
 - ssl_client_keystoredb connection parameter
 - configuring SSL 67
 - ssl_cnt_keydb configuration parameter
 - configuring SSL 67
 - ssl_cnt_stash configuration parameter
 - configuring SSL 67
 - SSL_KEYFILE 225
 - SSL_PW 225
 - SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA cipher suite 60
 - ssl_svcname configuration parameter
 - configuring SSL 60
 - ssl_svr_keydb configuration parameter
 - configuring SSL 60
 - ssl_svr_stash configuration parameter
 - configuring SSL 60
 - ssl_versions configuration parameter
 - configuring SSL 60
 - SSLClientKeystash connection parameter
 - configuring SSL 67
 - SSLClientKeystoredb connection parameter
 - configuring SSL 67
 - Statement Value Data field 105
 - Statement Value Index field 105
 - Statement Value Type field 105
 - static SQL
 - EXECUTE privilege 51
 - switching
 - user IDs 127, 133
 - SYSADM (system administration) authority
 - details 29
 - Windows 341
 - sysadm_group configuration parameter
 - Windows 341
 - SYSADMIN events 327
 - SYSCAT views
 - security issues 193
 - SYSCTRL (system control) authority
 - details 30
 - SYSDEFAULTADMWORKLOAD workload 45
 - SYSDEFAULTUSERWORKLOAD workload 45
 - SYSMAINT (system maintenance) authority
 - details 30
 - SYSMON (system monitor) authority
 - details 31
 - SYSPROC.AUDIT_ARCHIVE stored procedure 97, 103
 - SYSPROC.AUDIT_DELIM_EXTRACT stored procedure 97, 103
 - SYSPROC.AUDIT_LIST_LOGS stored procedure 103
 - system authorization IDs 46
 - system catalogs
 - listing privileges 193
 - retrieving
 - authorization names with privileges 194
 - names with DBADM authority 195
 - names with table access authority 195
 - privileges granted to names 196
 - security 196

T

 - table spaces
 - privileges 42
 - tables
 - access control 53
 - audit policy 93
 - inserting into LBAC-protected 182
 - LBAC effect on reading 179
 - privileges 49
 - protecting with LBAC 159, 177
 - removing LBAC protection 191
 - retrieving information
 - names with access to 195
 - revoking privileges 49
 - terms and conditions
 - publications 356
 - TLS (transport layer security) 71
 - TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher suite 60, 73
 - TLS_RSA_WITH_AES_128_CBC_SHA cipher suite 60, 73
 - TLS_RSA_WITH_AES_256_CBC_SHA cipher suite 60, 73
 - Transport Layer Security (TLS)
 - overview 71
 - troubleshooting
 - online information 356
 - security plug-ins 209
 - tutorials 356
 - trust relationships
 - Windows 336

- trusted clients
 - CLIENT level security 7
- trusted connections
 - establishing explicit trusted connections 127
 - overview 129
- trusted contexts
 - audit policies 93
 - overview 129
 - problem determination 135
 - role membership inheritance 132
- tutorials
 - list 356
 - problem determination 356
 - pureXML 356
 - troubleshooting 356

U

- UDFs
 - non-fenced 32
- UPDATE privilege 42
- updates
 - DB2 Information Center 353, 354
 - effects of LBAC on 184
- USAGE privilege
 - details 44
 - workloads 45
- user IDs
 - LDAP 228
 - selecting 4
 - switching 133
 - two-part 207
- user names
 - Windows restrictions 335
- USER_BASEDN 225
- USER_OBJECTCLASS 225
- USERID_ATTRIBUTE 225

V

- VALIDATE events 327
- views
 - access privileges examples 53
 - column access 53
 - privileges information 196
 - row access 53
 - table access control 53
- Vista
 - User Access Control (UAC) feature 346

W

- Windows
 - extended security 342
 - local system account (LSA) support 342
 - scenarios
 - client authentication 334
 - server authentication 334
 - user accounts
 - access tokens 337
- WITH ADMIN OPTION clause
 - delegating role maintenance 124
- WITH DATA option
 - details 105
- WLMADM (workload administration) authority
 - details 39

- WLMADM (workload administration) authority *(continued)*
 - overview 32
- write-down
 - details 171
- write-up
 - details 171

X

- XQuery
 - dynamic
 - EXECUTE privilege 51
 - static
 - EXECUTE privilege 51



Printed in USA

SC27-3872-01



Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

Database Security Guide

