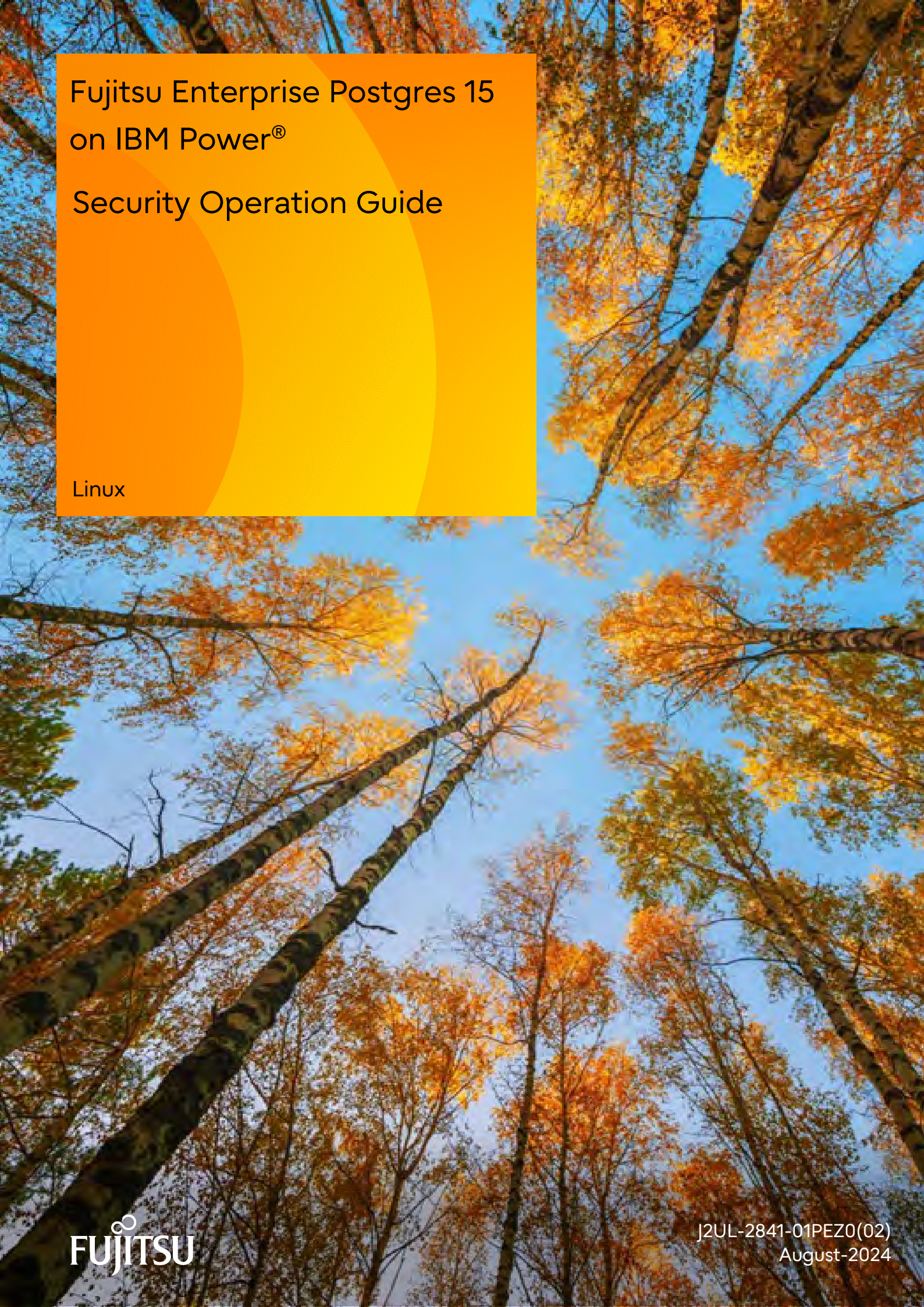
Fujitsu Enterprise Postgres 15 on IBM Power®

Security Operation Guide

Linux



FUJITSU

J2UL-2841-01PEZ0(02)
August-2024

Preface

Purpose of this document

This document describes security when building and operating a Fujitsu Software Enterprise Postgres database system.

Intended readers

This document is intended for those who are:

- Considering installing Fujitsu Enterprise Postgres
- Designing, building, and operating the security operating environment in Fujitsu Enterprise Postgres
- Accessing Fujitsu Enterprise Postgres database systems

Readers of this document are assumed to have general knowledge of:

- Business operations
- Fujitsu Enterprise Postgres
- Linux

Structure of this document

This document is structured as follows:

Chapter 1 Overview of Security

Provides an overview of the security system, and explains the security features provided by Fujitsu Enterprise Postgres.

Chapter 2 Overview of Security Operation

Provides an overview of security operation.

Chapter 3 Tasks of the Manager

Explains the tasks for security measures to be implemented by the manager.

Chapter 4 Tasks of Administrators

Explains the tasks for security measures to be implemented by administrators.

Chapter 5 Tasks of Users

Explains the tasks for security measures to be implemented by users.

Chapter 6 Audit Log Feature

Explains the audit log feature provided by Fujitsu Enterprise Postgres.

Chapter 7 Confidentiality Management

Explains the confidentiality management feature provided by Fujitsu Enterprise Postgres.

Appendix A Tables Used by Confidentiality Management Feature

Explains the tables used by the confidentiality management feature.

Appendix B System Management Functions Used by Confidentiality Management Feature

Explains the functions used by the confidentiality management feature.

References

This document contains abstracts from the following document:

- Database Security Guideline Version 2.0
  (Database Security Consortium (DBSC))

Export restrictions

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Issue date and version

```
Edition 1.2: August 2024
Edition 1.1: June 2023
Edition 1.0: April 2023
```

Copyright

Revision History

Revision	Location	Version
Corrected the article about the audit_log_disconnections parameter.	Chapter 6 Audit Log Feature	Edition 1.2
Feature names have been improved. Before modification : Confidential group After modification : Confidentiality group Before modification : Confidential level After modification : Confidentiality level Before modification : Confidential management role After modification : Confidentiality management role Before modification : Confidential management support After modification : Confidentiality management Before modification : Confidential matrix After modification : Confidentiality matrix Before modification : Confidential object After modification : Confidentiality object Before modification : Confidential privilege After modification : Confidentiality privilege Before modification : Confidential role After modification : Confidentiality role	Chapter 7 Confidentiality Management Appendix A Tables Used by Confidentiality Management Feature Appendix B System Management Functions Used by Confidentiality Management Feature	Edition 1.1

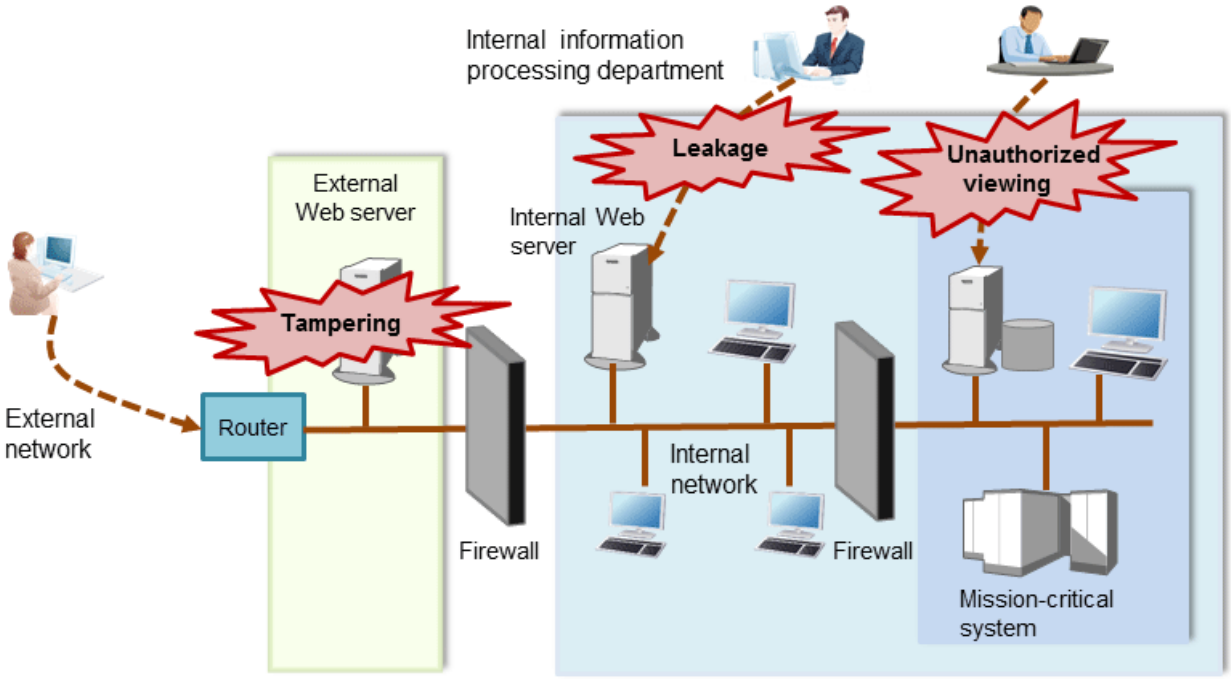
Contents

Chapter 1 Overview of Security

1.1 What is Security?

Computer security is the protection of information systems and data from risks such as leakage or tampering of information, attacks, intrusions, eavesdropping from external sources, and interference with information services. Security measures are essential for the advance prevention of security threats in order for information systems to gain trust as social infrastructure.

Figure 1.1 Security threats



The security measures in information systems can be classified as follows:

- Network
- Web
- Application
- Database
- PC

This document focuses on database security measures when using Fujitsu Enterprise Postgres.

1.2 Security Requirements

Below are the necessary security requirements for information systems.

Maintenance of security policies

A security policy clarifies the approach the company should take in relation to information assets, and the actions employees should take.

It is necessary to undertake security of information systems while maintaining security policies.

Integrated security management

Security has the aspects below. It is necessary to manage information in an integrated manner based on these aspects.

Confidentiality

Access to the information is restricted to prevent leakage of information outside of the company

Example measures: Prevention of information leakage or setup of access privileges

Integrity

Integrity is guaranteed, ensuring information does not become corrupted or tampered with

Example measures: Prevention or detection of tampering

Availability

Failure is prevented and normal operation is maintained so that information can be used when needed

Example measures: Power supply measures, system mirroring

1.3 Security Threats

A security threat is defined as something that threatens the confidentiality, integrity, and availability indicated in "1.2 Security Requirements" in respect to information assets. This includes technical threats such as accessing a database, but does not include physical destruction.

Threats are considered to be a combination of type of user who is the source of the threat, information assets that need to be protected, techniques, and unauthorized actions. For example, a threat might be a general user exploiting a database vulnerability to obtain database management information, and then tampering with that information.

When considering security measures, it is firstly necessary to clarify what kind of threats there are. A list of possible threats is shown in the table below. Refer to "Types of user" and "Information assets" for details on the definition of each type of user and information assets that should be protected.

Possible threats

Type of user	Information asset	Technique	Unauthorized action
General user Internal user System manager System developer System administrator System operator	Database management information	Eavesdropping of packets	Unauthorized acquisition (viewing) of information Unauthorized tampering or destruction (updating) of information
		Dictionary attack of passwords	
		Unauthorized acquisition of IDs/ passwords through social engineering	
		Unauthorized acquisition of information through misuse of settings	
		Unauthorized acquisition of information through exploiting a database vulnerability	
		Acquisition by an unauthorized route	
General user Internal user	General database information	Acquisition by a normal route	Misuse of information that can be acquired normally (taking data outside of the company)
		SQL issued with the aim of obstructing a job	Obstructing a job (resource depletion)
General user Internal user	General database information	Eavesdropping of packets	Unauthorized tampering or destruction (updating) of information
		Dictionary attack of passwords	
		Unauthorized acquisition of IDs/ passwords through social engineering	

Type of user	Information asset	Technique	Unauthorized action
		Unauthorized acquisition of information through exploiting configuration errors	
		Unauthorized acquisition of information through exploiting a database vulnerability	
		Acquisition by an unauthorized route	
System manager System developer System administrator System operator	General database information	Eavesdropping of packets	Unauthorized acquisition (viewing) of information Unauthorized tampering or destruction (updating) of information
		Dictionary attack of passwords	
		Unauthorized acquisition of IDs/ passwords through social engineering	
		Unauthorized acquisition of information through exploiting configuration errors	
		Unauthorized acquisition of information through exploiting a database vulnerability	
		Acquisition by an unauthorized route	
System developer	Database management information	Creation of a backdoor	Unauthorized acquisition (viewing) of information Unauthorized tampering or destruction (updating) of information
	General database information		
System manager System administrator	Database management information	Unauthorized acquisition of information by creating an unauthorized database administrator account	Unauthorized acquisition (viewing) of information Unauthorized tampering or destruction (updating) of information
	General database information		
System manager System operator	Database management information	Unauthorized acquisition of information by tampering with database-related files (definition file, physical file, and so on)	Unauthorized acquisition (viewing) of information Unauthorized tampering or destruction (updating) of information
	General database information		
Database administrator	Database management information	Misuse of information (taking information outside of the company) after obtaining it through the normal route	Misuse of information that can be acquired normally (taking information outside of the company)
		Unauthorized use of IDs/ passwords from the management information	Tampering with or destroying information that can be acquired
		Unauthorized acquisition of information by tampering with management information	

Type of user	Information asset	Technique	Unauthorized action
		SQL issued with the aim of obstructing a job	Obstructing a job (resource depletion)
	General database information	Eavesdropping of packets	Unauthorized acquisition (viewing) of information Unauthorized tampering or destruction (updating) of information
		Misuse of information (taking information outside of the company) after obtaining it through an unauthorized route	
Database operator	Database management information	Eavesdropping of packets	Unauthorized acquisition (viewing) of information Unauthorized tampering or destruction (updating) of information
		Dictionary attack of passwords	
		Unauthorized acquisition of IDs/ passwords through social engineering	
		Unauthorized acquisition of information by exploiting configuration errors	
		Unauthorized acquisition of information through exploiting a database vulnerability	
		Acquisition by an unauthorized route	
	General database information	Acquisition by a normal route	Misuse of information that can be acquired normally (taking data outside of the company)
		SQL issued with the aim of obstructing a job	Obstructing a job (resource depletion)

Types of user

In database security, the persons involved with databases and their roles are defined below.

Type of user	Role
System manager	Manages developers, administrators, and operators
System developer	Builds the network around the database server Builds the database server
System administrator	Operates devices of the surrounding database network Operates the database server
System operator	Operates the surrounding database network
Database administrator	Builds the database system Operates the database system
Database operator	Performs business operations
Internal user	End user inside the company
General user	End user outside the company

Information assets

In database security, it is necessary to protect the information assets to be stored on the database server.

Such assets are defined below.

Database management information

- Database configuration information (system catalog, user ID/password, and so on)
- Database logs (such as access logs)

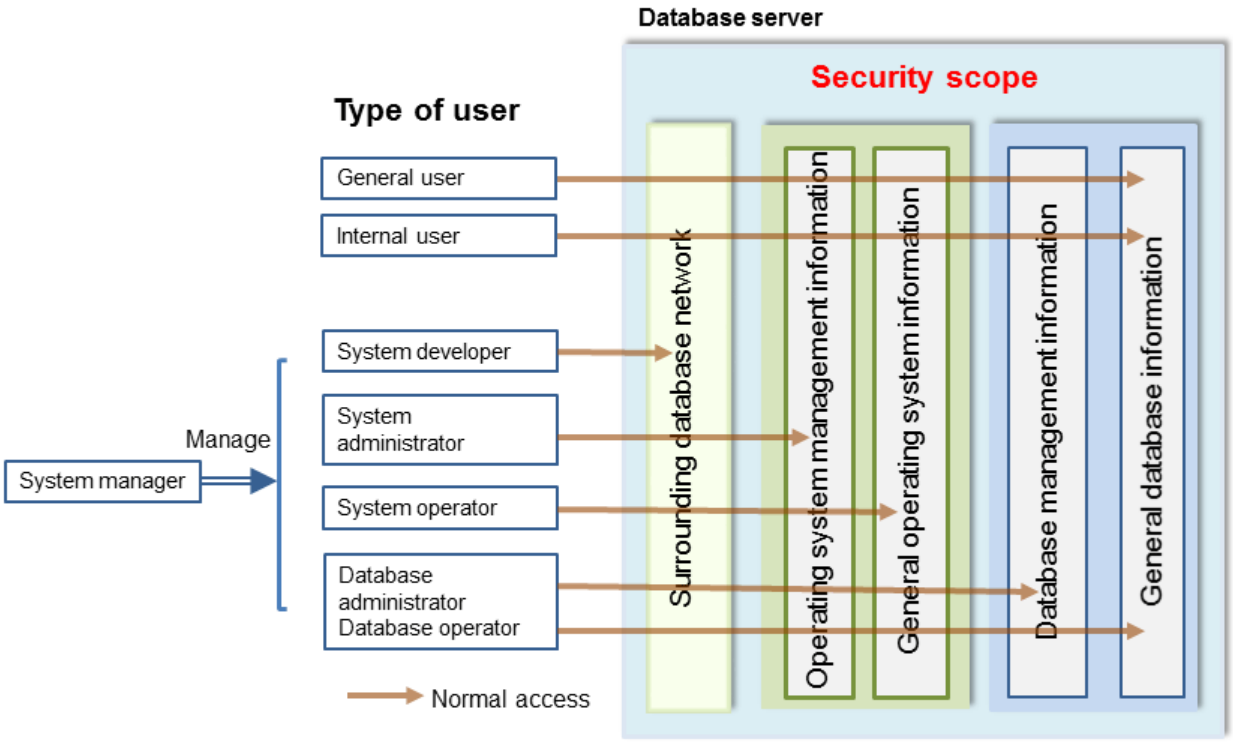
General database information

- Job data
- Applications

1.4 Security Scope

In database systems, both the database server and the surrounding database network are part of the security scope. It is necessary to clarify the extent of the security scope that each type of user is involved with, and consider security measures for the same.

The relationship of the security scope and the types of user is shown below.



1.5 Security Provided by Fujitsu Enterprise Postgres

Fujitsu Enterprise Postgres provides security features that satisfy the security requirements indicated in "1.2 Security Requirements".

This section describes security provided by Fujitsu Enterprise Postgres.

1.5.1 Roles Targeted For Security

In Fujitsu Enterprise Postgres database systems, the roles targeted in relation to security are "Manager", "Administrator", and "User". In order to build a robust security system, it is necessary to put security measures in place for each role.

The roles targeted for security and the mapping of Types of user indicated in "1.3 Security Threats" are shown in the table below.

Role targeted for security	Type of user
Manager	System manager
Administrator	System developer
	System administrator
	System operator
	Database administrator
	Database operator
User	General user
	Internal user

Manager

The manager establishes a security policy and decides on an operations policy for the organization as a whole.

Refer to "Chapter 3 Tasks of the Manager" for details.

Administrator

Administrators design, build and operate a system. While doing this, the administrators must implement the security measures in accordance with the security policy established by the manager.

Refer to "Chapter 4 Tasks of Administrators" for details.

User

A user is a person other than the manager or an administrator who accesses a database. There may be any number of users. It is necessary for users to be registered in the database system, and that access to the database is restricted according to the access privileges.

Refer to "Chapter 5 Tasks of Users" for details.

1.5.2 Security Features

Fujitsu Enterprise Postgres provides the following security features:

- Authentication
- Access control
- Encryption
- Audit log
- Data masking

This section describes each of these features.

Authentication

The databases that can be accessed can be restricted by authenticating the database users who access the database. Additionally, authentication of the server can be performed to prevent spoofing of the database server.

Refer to "Client Authentication" in "Server Administration" in the PostgreSQL Documentation for details on authentication.

Refer to "Secure TCP/IP Connections with SSL" in "Server Setup and Operation" in the PostgreSQL Documentation for details on server authentication.

Access control

Database objects can only be used by the object creator or database user who was specified as the owner when the object was created (both persons are hereinafter referred to as "owner"), or superuser, when objects are in their initial state. By having the object owner or superuser

control access privileges for database users, it is possible to control what kind of tables the database users who connect to the database can access, and what kind of operations they can perform.

Fujitsu Enterprise Postgres provides security management support features that support the design and operation of access control. For details, refer to "Chapter 7 Confidentiality Management".

Refer to "Privileges" in "The SQL Language" in the PostgreSQL Documentation for details on object access control.

Encryption

Fujitsu Enterprise Postgres provides a transparent data encryption feature that satisfies the requirements below.

- Confidential information can be changed into an unidentifiable state.
- The encryption key and data are managed separately.
- The encryption key is replaced at regular intervals.

Also, confidential data should not be operated without encryption. Fujitsu Enterprise Postgres provides security management support features to help prevent this. For details, refer to "Security Management Support".

PostgreSQL provides an encryption feature called "pgcrypto" that can also be used in Fujitsu Enterprise Postgres, however, it is recommended to use the transparent data encryption features because it will otherwise be necessary to modify the applications that consider encryption. Refer to "Protecting Storage Data Using Transparent Data Encryption" in the Operation Guide for details.

Additionally, if communication data transferred between a client and a server contains confidential information, it is necessary to encrypt the communication data to protect it against threats, such as eavesdropping on the network.

Refer to "Configuring Secure Communication Using Secure Sockets Layer" in the Operation Guide for details on encryption of communication data.

Audit log

A feature that addresses threats such as misuse of administrator privileges, unauthorized access to a database by a user, and other such threats. Information for tracing the processing of administrators and users is retrieved and stored as an audit log.

By periodically viewing and monitoring audit logs, the administrators can detect events that are impacting on the system in some way, or are depleting system resources as a result of incorrect operations by users, and can take appropriate measures to prevent information leakages or system failures in advance.

Refer to "Chapter 6 Audit Log Feature" for details.

Data masking

A feature that changes part of the data to make it available for reference in response to queries issued by an application.

For example, for a query of employee data, digits except the last four digits of an eight-digit employee number can be changed to "*" so that it can be used for reference without exposing the actual data.

Specifically, the data changed by the data masking feature can be transferred to a test database so that users who perform testing or development can reference the data. During testing, it is desirable to use the data that will be used on a production environment database. However, actual production data should not be used as is for testing because of the risk of leakage of confidential data. This feature enables data that is similar to actual production data to be safely used in test and development environments.

Refer to "Data Masking" in the Operation Guide for details on data masking.

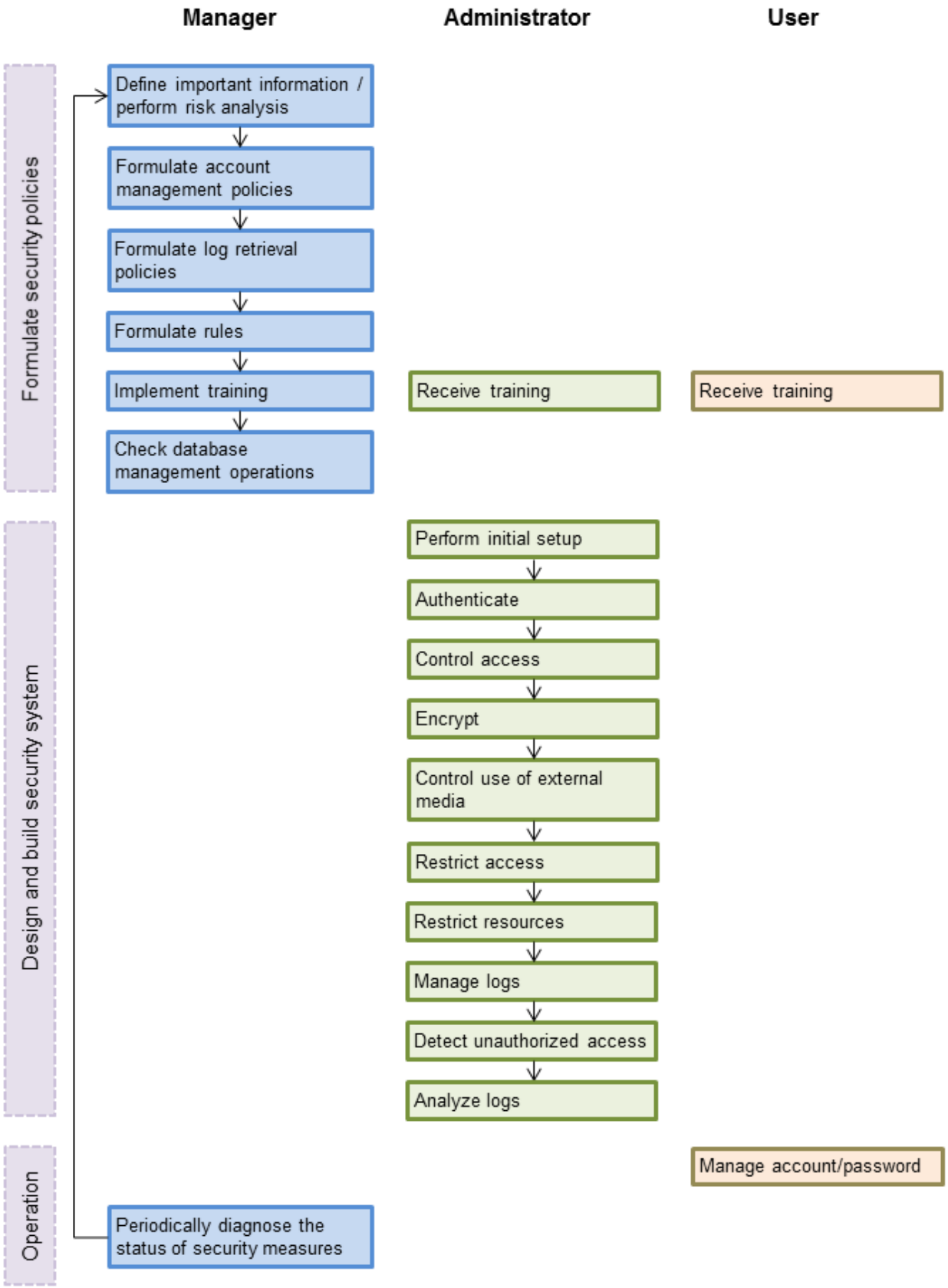
Chapter 2 Overview of Security Operation

2.1 Security Operation Flow

This section shows the flow of work when building a security environment and performing security operation in Fujitsu Enterprise Postgres.

When performing security operation, there are technical measures to be implemented to address security threats by equipping the system with security features, and manual work, such as the implementation of security guidelines, a training system, and the establishment of usage rules.

Figure 2.1 Security operation flow



Chapter 3 Tasks of the Manager

The manager formulates security policies, which become guidelines for security measures.

3.1 Defining Important Information and Risk Analysis

Before formulating security policies, define important information and perform risk analysis. Based on the importance of the information and the result of risk analysis, decide what kind of security measures to put in place.

In defining the important information, identify what should be protected and classify it by importance in order to effectively implement the security measures. Information that should be protected includes "database management information" and "general database information", as indicated in "Information assets". Examples of information classifications are "personal information" and "confidential information".

In the risk analysis, refer to "Possible threats" to identify threats that may arise, and analyze the risks in respect to such threats.

Additionally, by performing a risk analysis once annually as a guide, it is possible to identify threats that may adversely impact the business and related vulnerabilities.

3.2 Formulating Account Management Policies

In formulating an account management policy, implement the following and document the formulated policy.

Organize system users and roles

Identify the necessary roles of the relevant system based on "Types of user". Additionally, organize personnel for each role.

Organize accounts

Organize accounts with the appropriate privileges for each role, and decide on account policies.

- Database administrator account
  - Organize separate accounts for database administrators and database operators
  - Ensure that the database administrator account can only be used by specific persons
  - Perform tasks that do not require database administrator privileges using a separate account without database administrator privileges
- General account

  Create an account for general users by application usage.

Review account management policy

Review the accounts in order to effectively implement security measures.

- Regularly check the accounts mentioned above and their privileges, and determine if they are still appropriate
- If there have been system or operational changes, review the accounts and privileges
- If unsuitable accounts and privileges are discovered, modify them as required

3.3 Formulating Log Retrieval Policies

In formulating a log retrieval policy, implement the following and document the formulated policy.

Organize the purpose of log retrieval

To clarify what logs will be retrieved for, define their reason for retrieval.

Examples of the purpose might include, "To use for investigation in the event of unauthorized access", and "To submit to investigating authorities as evidence if any issues arise".

Decide on the types of logs to be retrieved

In order to retrieve appropriate logs, organize the types of logs that can be retrieved in the target system, and decide on the logs to be retrieved.

Examples of log types are "operating system logs", "application run logs", and "database audit logs".

Organize log retrieval target access

In order to decide on access for log retrieval targets, organize what kind of access will take place.

For example, the following access is possible:

- Access related to important information
  - Access to personal information, confidential information, and database management information
  - Access outside of business hours
  - Login
  - Specific SQL
- Access suspected to be unauthorized
  - Large amount of search access
  - Access from different locations
  - Access outside of business hours

Decide on the log retrieval content

In order to effectively use retrieved logs, organize the required content as a log, and decide on the retrieval content.

For example, the following output content is possible:

- When (time)
- Who (database account, application user)
- What (object ID, table name)
- Where from (machine name, IP address)
- How (SQL type, SQL statement)
- Execution result (success/fail)

Formulate log maintenance policy

In order to use the logs as purposed, formulate the log maintenance policy.

For each log, define its location, storage medium, retention period, access control, and so on.

3.4 Formulating Rules

Formulate the rules that will become the standard for security measures of the target system. Additionally, prescribe penalties for security violations. For example, formulate rules and penalties as below:

- Rules
  - Applying security patches and update programs
  - Prohibiting unauthorized acquisition of information from the database
  - Prohibiting the saving of acquired information to media that is not permitted for use
- Penalties
  - Prescribe penalties in the company's employment policies and procedures

- Set fines

3.5 Implementing Training

In order to have administrators and users recognize the importance and necessity of information security, and to prevent unauthorized access due to operational omissions and mistakes, implement and promote security-related training for administrators and users.

For example, implement promotion of security policies, formulation of training schedules, and formulation of training materials.

3.6 Checking the Database Management Operations

In order to prevent operational errors and unauthorized actions by administrators, implement the measures below:

- Always collect the latest information on security incidents and vulnerabilities related to databases
- Implement management operations only after providing advance notice
- Retain records of management operations

3.7 Periodic Diagnosis of the Status of Security Measures

In order to check if the security measures are effective, periodically diagnose if the security measures have been put in place appropriately based on the security threats.

Additionally, evaluate if the current security measures and policies are effective for the threats and vulnerabilities, and if there are any issues, review the security policies and security measures.

Chapter 4 Tasks of Administrators

Administrators perform the actions below as security measures when designing, building, and operating the system in accordance with the security policies formulated by the manager.

Preparation

- Implement training

Measures to protect against unauthorized behavior

- Perform initial setup
- Authenticate
- Control access
- Encrypt
- Control use of external media
- Restrict access
- Restrict resources

Measures to detect and trace unauthorized behavior

- Manage logs
- Detect unauthorized access
- Analyze logs

4.1 Receiving Training

Administrators receive security-related training in accordance with the training schedule formulated by the manager. Additionally, administrators instruct users to receive training.

4.2 Initial Setup

To minimize database vulnerabilities and the possibility of unauthorized access, implement the security measures below in the initial stage of system building. Additionally, configure the database server so that it primarily operates the database system only.

Making the server more robust

Configure the operating system and network to prevent intrusion into or destruction of a database server, so that the system operates on a secure server.

- Remove unnecessary features or services on the operating system
- Enable only the necessary protocols
- Implement the security features for services, protocols, and daemons considered to have a relatively low security level, such as file sharing and FTP

Installing the latest version

Always download and apply the latest patches in order to reflect the latest security measures.

Installing the minimum necessary features

Install only the necessary features in order to prevent unauthorized use of the system.

Additionally, delete or disable features and services that will not be used.

Changing the port

To prevent unauthorized use of the system, change the default port that is set during installation.

Point

Specify the port during setup of Fujitsu Enterprise Postgres. Refer to the Installation and Setup Guide for Server for details.

Access restrictions for communication features

To prevent unauthorized use of the system using the communication features, implement access restrictions for communication features.

Settings for prohibiting the access path to database configuration files

To prevent database destruction, implement the measures below:

- Restrict users who are permitted to access database configuration files, and periodically review the permissions
- Allow only administrators to access table or definition scripts

Restrictions on the access path to the database

To prevent unauthorized use or operating errors for the database, restrict the distribution range of applications used to access the database only to devices used by users who are permitted access.

Dealing with unauthorized programs

To prevent unauthorized intrusions into a system through a backdoor, such as by tampering with the program source code of an application, document the author of the program to be run and perform checking and testing so that the program will not be tampered with. Additionally, employ safe coding techniques so that issues with general coding vulnerabilities can be addressed.

System security settings

In cases where it is clear that the system security settings will impact security, set reliable security settings in the initial setup stage, such as setting appropriate security parameters.

4.3 Authentication

When accessing a database, authentication must always be performed in order to prevent tampering or information leakage from spoofing by a malicious user.

Password authentication is used when logging on to a database, and the account and password used for authentication are to be strictly managed by administrators.

Additionally, authentication must also be implemented reliably for connections to a database from clients, so that only permitted users can access the database.

4.3.1 Managing Accounts

For account management, perform the actions below.

Create the required accounts

To prevent unauthorized use of accounts, such as spoofing, implement the measures below when creating an account:

- Select the required account
- Specify the user privileges
- Create database administrator accounts and general user accounts separately according to the privileges

Point

Accounts are created using the CREATE ROLE statement. Refer to "CREATE ROLE" in the PostgreSQL Documentation for details.

Delete unnecessary accounts

Remove accounts not used on a daily basis, such as unused accounts and accounts not needed for operations that are created by default during product installation.

Point

Accounts are deleted using the DROP ROLE statement. Refer to "DROP ROLE" in the PostgreSQL Documentation for details.

Set up account lockout

The usage frequency of accounts is to be checked periodically, and if there are any accounts that have not been used for a long period, lock those accounts. Set a limit for failed login attempts, and if this limit is exceeded, lock the account. Additionally, set the period until a locked account is reenabled.

Point

Account locking can be performed by using LDAP authentication. Refer to "LDAP Authentication" in the PostgreSQL Documentation for details.

Manage database administrator accounts

Manage database administrator accounts in accordance with the account management policy formulated by the manager.

Manage development environment and production environment accounts

To prevent unauthorized use of accounts used in a development environment, delete accounts used in the development environment before operation starts in the production environment. In cases where it is unavoidable to use an account used in the development environment in the production environment, use different passwords in each environment.

Set up a temporary use account

If a temporary user will use the system, either provide a shared account with a temporary password for each use, or create a temporary account.

4.3.2 Managing Passwords

Manage passwords as below.

Make strong passwords

The use of account passwords that can easily be guessed by others, such as a password that matches the ID, or the default password provided during installation, is prohibited. Set a complex and strong password.

Change passwords regularly

Change passwords regularly to prevent others from accessing the account in case the password is obtained by unauthorized means. Additionally, configure the settings to force a password change when prompted after the first use.

Set the password expiry period

To encourage regular changing of passwords, set a password expiry period.

Point

Password setting and changing is specified using the CERATE ROLE statement or ALTER ROLE statement. Refer to "CREATE ROLE" and "ALTER ROLE" in the PostgreSQL Documentation for details.

Additionally, by using passwordcheck and LDAP authentication, the actions below can be performed:

- The default password set during installation can be changed
- The password expiry period can be set
- The number and types of characters used for the password can be checked

Refer to "passwordcheck" and "LDAP Authentication" in the PostgreSQL Documentation for details.

4.3.3 Configuring Connections and Authentication

Configure connections and authentication so that the database can only be accessed by permitted users.

Point

Client authentication is configured in pg_hba.conf. Refer to "Client Authentication" in the PostgreSQL Documentation for details.

4.4 Access Control

If appropriate access privileges are not set for administrators and users, security incidents may occur, such as information leakage resulting from access to information by an unauthorized person. To minimize such incidents, it is necessary to implement the security measures below for the access privileges and perform rule-based access control.

Point

Notes when setting access privileges

- The creation of a special account that allows granting of privileges to all users is prohibited
- The creation of a general account that allows access to general information such as operations data is prohibited

Identifying the database access requirements

To set the appropriate access privileges for each usage purpose for the database, follow the procedure below to identify the access requirements:

1. Classify the usage purpose of the account, such as "For database management", "For object management", and "For data access".
2. Classify the required privileges for each usage purpose, such as "By feature" and "By object".
3. Categorize the accounts based on each privilege.
4. Identify the minimum necessary range of data and minimum necessary access content (view, update, create, delete) to be accessed for each categorized account, and decide on the database access requirements.

Setting the access privileges

Assign the minimum necessary privileges based on the database access requirements for each categorized account. Additionally, restrict accounts when assigning administrator privileges.

Reviewing access privileges

To reflect changes in access requirements in the system, periodically review the access privileges and check if there are any access privileges that are no longer needed. If any unnecessary access privileges have been set, promptly modify the access privileges.

Point

Access privileges are set using the GRANT statement or REVOKE statement. Refer to "GRANT" and "REVOKE" in the PostgreSQL Documentation for details.

4.5 Encryption

To prevent unauthorized usage of data in the event information leakage occurs due to data theft, eavesdropping of communication, and other such activities, implement the encryption measures below.

Encrypt communication

To protect data from eavesdropping over the network between a database server and clients, use the encryption feature to encrypt communications.

Refer to "Configuring Secure Communication Using Secure Sockets Layer" in the Operation Guide for details.

Encrypt data

To protect data from theft, use the encryption feature to encrypt the data. The data below is targeted for encryption:

- Data to be stored on the database
- Backup data
- Data files

Refer to "Protecting Storage Data Using Transparent Data Encryption" in the Operation Guide for details.

Manage encryption keys

Restrict the persons who can access the encryption key to a minimum number of database administrators.

Additionally, to ensure the encrypted information will not be easily decrypted, create a mechanism for appropriately managing the encryption key for the entire life cycle (generation, distribution, saving, and disposal), and strictly manage the encryption key.

Refer to "Configuring Secure Communication Using Secure Sockets Layer" and "Protecting Storage Data Using Transparent Data Encryption" in the Operation Guide for details.

4.6 Controlling Use of External Media

Information leakage can be prevented by controlling use of external media (such as CD/DVD, USB drive, and external hard disk) and PCs that are connected to the database, and restricting the removal of data from the database.

Restricting connection of external media

Remove external media and printers that will not be used in operations, and restrict connection of external media to which information may be written.

Restricting use of external media

Restrict connections for external media and printers to control the writing of information to these devices.

Controlling use of connected PCs

Prevent leakage of information from PCs connected to the database:

- Limit connections of external media to PCs
- Implement security measures to make the PC robust
- Implement individual authentication for access from the PC
- Manage the installed software and monitor the software usage status
- Limit connections to printers

4.7 Security Measures for Servers/Applications

An even more robust security system can be achieved by strengthening security for servers and applications in addition to the security measures for databases. Implement the security measures below for servers and applications:

Restrict access

Implement the measures below and restrict access to the database server:

- Install the database server inside the firewall to prevent direct access to the database server from many unspecified PCs.
- In the local network, implement measures such as using the router to restrict IP addresses, and restrict PCs and segments that can directly access the database server.

Restrict resources

Restrict excessive use of CPU resources by general users to prevent the disruption of service and extraction of large amounts of data.

4.8 Log Management

Logs are a feature that addresses threats such as misuse of administrator privileges, and unauthorized access to a database by a user. Information for investigating/tracing processes and operations performed for the database is retrieved and managed as logs for identifying the cause in the event information leakage or unauthorized access occurs.

Fujitsu Enterprise Postgres provides the audit log feature for retrieving and managing logs.

Refer to "Chapter 6 Audit Log Feature" for details.

This section describes the information that should be obtained as logs and how to maintain logs, as a measure for managing information leakage and unauthorized access.

4.8.1 Retrieving Logs

The audit logs below are retrieved in accordance with the log retrieval policy formulated by the manager.

Login information

Retrieves logs during login and logout.

Database access information (view/update)

Retrieves all access relating to the information below:

- General database information (such as personal information and confidential information used in the business)
- Database management information (system catalog, user ID/password, and so on)

Changed information of database objects

Retrieves logs related to creating, changing, and deleting database objects such as database accounts and tables.

Operation logs for audit logs

To prevent suppression of retrieved audit logs, operations such as initialization of audit logs, and stoppage of the audit log feature are retrieved as logs.

4.8.2 Maintaining Logs

Logs are maintained in accordance with the log retrieval policy formulated by the manager.

Storing logs

Perform the actions below and store logs securely so that the retrieved logs will not be updated by others:

- Save logs to external media, and store the external media in a secure location, such as lockable storage
- Restrict the viewing of logs to administrators only, and set access restrictions for logs, such as not assigning update rights
- Decide on the log retention period, with consideration to cases where investigation tracing back to the time of discovery of an issue is required

Preventing tampering of logs

Implement measures to prevent tampering of logs, such as retaining multiple copies of logs and using storage that cannot be rewritten.

Encrypting logs

Encrypt logs so that logs are not easily viewed.

4.9 Detecting Unauthorized Access

To address unauthorized access, it is necessary to establish a mechanism for detecting unauthorized access to databases and monitor access.

Communicating unauthorized access

Create a mechanism that notifies of detected unauthorized access, such as notifying the manager and the administrator, if an account lock occurs due to the limit for failed login attempts being exceeded.

Checking access times

Create a mechanism that can check for suspicious access to the information below outside of normal access hours, together with implementing measures to address such access.

Detecting access to database management information

- Monitor logs and detect access during timeframes that have not been applied for
- In the event a request for access permission outside of normal access hours is made, the log is checked for any discrepancies in the requested content and work result

Detecting access to general database information

- Decide on the timeframes during which access to the database is permitted for each general account
- Detect access outside of normal access hours from session information logs

Checking the connection source where access is not permitted

To detect access from connection sources that are not permitted, define the sources from where access is permitted, and detect access from connection sources that are not permitted.

Define the access patterns (connection source, operating system user and account) of database administrator accounts and general accounts, and check for access outside of these patterns.

4.10 Analyzing Logs

Create a mechanism that analyzes logs to detect unauthorized behavior in cases where information leakage, unauthorized access, or other such activity, is suspected. Analyses should include those shown below.

Periodic analysis of session information

Analyze session information of logs from the perspectives below to detect unauthorized logins:

- Trend of sessions with a large number of failed login attempts
- Trend of sessions with accounts that are logged in for long periods of time
- Trend of sessions in which a large amount of resources are used

Periodic analysis of database access information

Analyze SQL statements from the perspectives below to detect unauthorized access to the database:

- Trend of SQL being executed over a long period of time
- Trend of SQL using a large amount of resources

Chapter 5 Tasks of Users

The user performs the actions below as security measures when using the system.

5.1 Receiving Training

The user must receive security-related training as instructed by the manager or the administrator to learn about security. By having users with a common awareness relating to security, an even more stable security system can be established.

5.2 Managing Accounts/Passwords

Users can use the database system by using the account and password provided by the administrator. At such times, the user is to implement the measures below so that the account and password are not misused by others:

- Be responsible for managing the ID and password in a manner that ensures the account does not become locked during login
- Change the password regularly
- Comply with the expiry period set for the password by promptly changing the password when it is about to expire

Chapter 6 Audit Log Feature

In PostgreSQL, logs output as server logs can be used as audit logs by using the log output feature. There are, however, logs that cannot be analyzed properly, such as SQL runtime logs, which do not output the schema name. Additionally, because the output conditions cannot be specified in detail, log volumes can be large, which may lead to deterioration in performance.

The audit log feature of Fujitsu Enterprise Postgres enables retrieval of details relating to database access as an audit log by extending the feature to pgaudit. Additionally, audit logs can be output to a dedicated log file or server log. This enables efficient and accurate log monitoring.

Note

The audit log feature cannot be used if PostgreSQL is running in single-user mode.

6.1 Audit Log Output Modes

In pgaudit, the two types of audit log below can be output.

Session Audit Logging

Session Audit Logging outputs information related to SQL executed in backend processes (processes generated when connection requests are received from clients), information related to starting and connecting databases, and information related to errors, as a log. In Session Audit Logging, by specifying the log output conditions and filtering the logs to be output, performance degradation due to outputting large volumes of logs can be prevented.

Refer to "6.4 Session Audit Logging" for details.

Object Audit Logging

When SELECT, INSERT, UPDATE, and DELETE are executed for specific objects (tables, columns), Object Audit Logging outputs these as a log. TRUNCATE is not supported. Object Audit Logging outputs object operations for which privileges have been assigned to specified roles, as a log. Object Audit Logging can control log output at an even finer level of granularity than Session Audit Logging.

Refer to "6.5 Object Audit Logging" for details.

Information

Depending on the application or command, Fujitsu Enterprise Postgres may execute SQL internally and the audit logs may be retrieved.

Also, the audit logs of multiple SQLs with the same statement ID may be retrieved. This is because before the user executes the SQL, another SQL is executed internally by Fujitsu Enterprise Postgres.

6.2 Setup

This section describes the setup method of pgaudit.

1. Copy the pgaudit files

   As superuser, run the following command. Note that "<*x*>" in paths indicates the product version.

   ```
   $ su -
   Password:******
   # cp -r /opt/fsepv<x>server64/OSS/pgaudit/* /opt/fsepv<x>server64
   ```

2. Create the pgaudit configuration file

   Create the pgaudit configuration file, which describes the information required for pgaudit actions. Create the file using the same encoding as used for the database.

   In addition, set write permissions for the database administrator only in the pgaudit configuration file so that policies related to the audit log are not viewed by unintended users.

Refer to "6.3 pgaudit Configuration File" for details.

Note

Do not define the rule section in the pgaudit configuration file at this point.

Example of a pgaudit configuration file

```
[output]
logger = 'auditlog'
```

3. Configure postgresql.conf

   Configure the parameters below in postgresql.conf to use audit logs:

   shared_preload_libraries

   Specify "pgaudit".

   pgaudit.config_file

   Specify the deployment destination path of the pgaudit configuration file.

   If a relative path is specified, the path will be relative to the data storage directory.

   log_replication_commands

   Specify "on".

   log_min_messages

   Check if "ERROR" or higher has been specified.

   If outputting an audit log to a server log ("serverlog" is specified in the logger parameter of the pgaudit configuration file), check the parameters below relating to server logs.

   logging_collector

   Check if "on" has been specified.

   log_destination

   Check if "stderr" has been specified.

   log_file_mode

   Check if the server log permissions are appropriate, so that only the permitted persons can access it.

   ### Information

   The default for the log_file_mode parameter is 0600, which only allows the database administrator to have access.

   For example, to permit other members of the group to which the database administrator belongs to view the audit logs, specify 0640 for log_file_mode.

   Example

   ```
   log_file_mode = 0640
   ```

   The database administrator can also be prevented from viewing audit logs by specifying 0000. However, write privileges are assigned for outputting logs.

   If outputting an audit log to a dedicated log file ("auditlog" is specified in the logger parameter of the pgaudit configuration file), check the parameter below.

max_worker_processes

If the max_worker_processes parameter has been set, add 1 to the specified value.

See

Refer to "Error Reporting and Logging" in the PostgreSQL Documentation for details on server logs.

If using database multiplexing, refer to "6.6 Database Multiplexing" for details.

Example of postgresql.conf

In the example below, only the parameters that need to be configured when using the audit log feature are described.

```
shared_preload_libraries = 'pgaudit'
pgaudit.config_file = 'pgaudit.conf'
log_replication_commands = on
log_min_messages = WARNING
```

4. Start the instance

   Start the instance and check if the message below is output.

```
LOG: pgaudit extension initialized
```

5. Create the pgaudit extension

   Execute CREATE EXTENSION to create the pgaudit extension.

```
$ psql
=# CREATE EXTENSION pgaudit;
=# \dx
                  List of installed extensions
Name    | Version |   Schema   |     Description
--------+---------+------------+---------------------------------
pgaudit | 1.0     | public     | provides auditing functionality
plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
(2 rows)
```

6. Configure the parameters in the pgaudit configuration file

   Add or change the parameters in the pgaudit configuration file as required.

   Refer to "6.3 pgaudit Configuration File" for details.

7. Restart the instance

   Restart the instance to apply the changes to the pgaudit configuration file. After restarting, check if the changes have been reflected correctly.

```
LOG:  log_catalog = 1
LOG:  log_level_string =
LOG:  log_level = 15
LOG:  log_parameter = 0
LOG:  log_statement_once = 0
LOG:  role =
LOG:  logger = auditlog
LOG:  log_directory = pgaudit_log
LOG:  log_filename = pgaudit-%Y-%m-%d_%H%M%S.log
LOG:  log_file_mode = 0600
LOG:  log_rotation_age = 1440
LOG:  log_rotation_size = 10240
LOG:  log_truncate_on_rotation = 0
LOG:  fifo_directory = /tmp
```

```
LOG:  Rule 0
LOG:  pgaudit extension initialized
```

6.3 pgaudit Configuration File

In the pgaudit configuration file, specify the information required for pgaudit actions. The pgaudit configuration file comprises three sections: "output section", "option section", and "rule section".

output section

The output section is specified using the format below:

- *paramName* = '*value*'

The valid parameters in the output section are shown in the table below.

Parameter name	Description	Remarks
logger	Dedicated log file (auditlog)/*serverLog* (serverlog) that will be the output destination of the audit log The default is "auditlog" (dedicated log file).	The dedicated log file is output using the same encoding as used for the database.
log_directory	Directory where the audit log is to be created Specify the full path or the relative path from the data storage directory. The default is "pgaudit_log".	Enabled only if "auditlog" is specified for the logger parameter
log_filename	File name of the audit log Specify a file name that varies according to the time, in the same manner as for log_filename in the postgresql.conf file. The default is "pgaudit-%Y-%m-%d_%H%M%S.log".	Enabled only if "auditlog" is specified for the logger parameter
log_file_mode	Specify the permissions of the audit log so that only permitted persons can access it. The parameter value is the numeric mode specified in the format permitted in chmod and umask system calls. The default is "0600". Refer to "log_file_mode" in "6.2 Setup" for information on audit log file permissions.	Enabled only if "auditlog" is specified for the logger parameter
log_rotation_age	Maximum age of the audit log file A new audit log file is generated when the time (minute units) specified here elapses. To disable generation of new log files based on time, specify "0". The valid units are "min" (minutes), "h" (hours), and "d" (days). If the unit is omitted, "min" will be used. The default is "1d" (1 day).	Enabled only if "auditlog" is specified for the logger parameter
log_rotation_size	Maximum size of the audit log file A new log file will be generated after logs of the size specified here are output to a log file. To disable generation of new log files based on size, specify "0". The valid units are "kB" (kilobytes), "MB" (megabytes), and "GB" (gigabytes). If the unit is omitted, "kB" will be used.	Enabled only if "auditlog" is specified for the logger parameter

Parameter name	Description	Remarks
	The default is "10MB".	
log_truncate_on_rotation	If rotating audit log files based on time, this parameter is used to specify whether to overwrite (on)/not overwrite (off) existing audit log files of the same name. For example, if "on" is specified, and "pgaudit-%H.log" is specified for log_filename, 24 separate log files will be generated based on time, and those files will be cyclically overwritten. The default is "off". If "off" is specified, the logs will be written to the existing audit log files.	Enabled only if "auditlog" is specified for the logger parameter
fifo_directory	FIFO (named pipe) directory to be used between the daemon process that outputs audit log files and the backend process FIFO named p.PGAUDIT.*nnnn* (*nnnn* is the postmaster PID) are created in the fifo_directories directory. The files cannot be deleted manually. The default is "/tmp".	Enabled only if "auditlog" is specified for the logger parameter

Information

If the logger parameter is set to "serverlog", audit logs will be output to the server log as log messages, therefore the status information and message severity level according to the log_line_prefix parameter in postgresql.conf will be output to the beginning of the audit log.

If the logger parameter is omitted or set to "auditlog", audit logs will be output to a dedicated log file as dedicated logs, therefore the status information and message severity level according to the log_line_prefix parameter in the postgresql.conf file will not be output.

Refer to "Output format" in "6.4 Session Audit Logging"or "Output format" in "6.5 Object Audit Logging" for information on the output format of audit logs.

Point

The pgaudit log_file_mode configuration parameter setting is separate from, and unaffected by, the log_file_mode GUC parameter setting and the -g/-allow-group-access initdb option.

When using a dedicated pgaudit log file, since the pgaudit log_directory location defaults to inside the data storage directory, it is possible for the pgaudit log_file_mode permissions to conflict with the intended file permissions specified by the -g/-allow-group-access initdb option. In this case, the pgaudit log_directory should be specified to be a directory located outside of the data storage directory.

option section

The option section is specified using the format below:

- *paramName* = '*value*'

The valid parameters in the option section are shown in the table below.

Parameter name	Description	Remarks
role	Name of roles used in Object Audit Logging If specifying a name containing uppercase characters, key words, multibyte characters and commas, enclose the name in double quotation marks.	Parameter used in Object Audit Logging only

Parameter name	Description	Remarks
log_catalog	Whether to enable (on)/disable (off) log output for pg_catalog Specify "off" if you do not want to retrieve audit logs that access pg_catalog. The default is "on" (enabled).	
log_parameter	Whether to enable (on)/disable (off) output of values passed by parameters in SQL execution The default is "off" (disabled).	
log_statement_once	Whether to control (on)/not control (off) output for the second and subsequent SQL statements if the same SQL statement is the log output target The default is "off" (do not control).	
log_level	Log level of audit logs The valid values are "DEBUG5", "DEBUG4", "DEBUG3", "DEBUG2", "DEBUG1", "INFO", "NOTICE", "WARNING", and "LOG". The default is "LOG".	Enabled only if "serverlog" is specified for the logger parameter
audit_log_disconnections	When using Mirroring Controller, specify whether to enable (on) or disable (off) the output of disconnection logs for connections other than those of Mirroring Controller. The default is "off" (disabled). This parameter is valid when log_disconnections in postgresql.conf is off.	Parameter used in Session Audit Logging only

rule section

The rule section is used in Session Audit Logging. Refer to "6.4 Session Audit Logging" for details.

Note

Do not specify the rule section if the role parameter has been specified in the option section. If you specify the rule section, the audit logs of Object Audit Logging and Session Audit Logging will be output intermingled. The CSV format of the role audit log and the CSV format of the rule audit log are different, you will be unable to view in CSV format.

6.4 Session Audit Logging

In Session Audit Logging, specify the rules for filtering logs to be output in the rule section in the pgaudit configuration file.

Rules are specified using the formats below. Multiple values can be specified, using a comma as the delimiter.

- *paramName* = '*value*'
- *paramName* != '*value*'

If [rule] is described on its own in the rule section with no parameters specified, all audit logs of Session Audit Logging will be output.

Example

```
[output]
logger = 'auditlog'
[rule]
```

If [rule] is not described as a section, audit logs of Session Audit Logging will not be output.

Example

```
[output]
logger = 'auditlog'
```

The valid parameters in the rule section are shown in the table below.

Parameter name	Description	Example
timestamp	Timestamp range Refer to "timestamp" for details on how to specify timestamps.	timestamp = '09:00:00 - 10:00:00, 18:00:00 - 18:30:00'
database	Database name To specify a blank value, use a pair of double quotation marks ("") instead of the *value*. When specifying a name containing uppercase characters, key words, multibyte characters and commas, enclose the name in double quotation marks.	database = 'prodcut_db'
audit_role	Role name To specify a blank value, use a pair of double quotation marks ("") instead of the *value*. When specifying a name containing uppercase characters, key words, multibyte characters and commas, enclose the name in double quotation marks.	audit_role = 'appuser1'
class	Operation class Select from the values below. Multiple values can be specified. Refer to "class" for details on the meaning of each class. - BACKUP - CONNECT - DDL - ERROR - FUNCTION - MISC - READ - ROLE - WRITE - SYSTEM	class = 'READ, WRITE'
object_type	Object type This parameter is enabled when the class parameter is "READ" and "WRITE". Select from the values below. Multiple values can be specified. - TABLE - INDEX	object_type = 'TABLE, INDEX'

Parameter name	Description	Example
	- SEQUENCE - TOAST_VALUE - VIEW - MATERIALIZED_VIEW - COMPOSITE_TYPE - FOREIGN_TABLE - FUNCTION	
object_name	Object name This parameter is enabled when the class parameter is "READ" and "WRITE". For objects that can be modified by a schema, such as a table, modify such objects by schema name. When specifying a name containing uppercase characters, key words, multibyte characters and commas, enclose the name in double quotation marks. When specifying the object name "*schemaName*.*tableName*", enclose the entire object name modified by schema name in double quotation marks. To specify a blank value, use a pair of double quotation marks ("") instead of the *value*.	object_name = 'myschema.tbl1' object_name = 'myschema.tbl1, "*mySchema*.TABLE"'
application_name	Application name To specify a blank value, use a pair of double quotation marks ("") instead of the *value*.	application_name = 'myapp'
remote_host	Connection source(client side) host name or IP address If "on" is specified for the log_hostname parameter in the postgresql.conf file, specify a host name. Otherwise, specify the IP address. If using a local host, specify "[local]". To specify a blank value, use a pair of double quotation marks ("") instead of the *value*.	remote_host = 'ap_server'

timestamp

Specify a timestamp range from "*startTime*" to "*endTime*" for the log output target. The timestamp format is 'hh:mm:dd-hh:mm:dd' (hh is expressed in 24-hour notation, and hh, mm, and dd are expressed in two-digit notation).

The start time must be earlier than the end time. If specifying multiple ranges, specify each start and end timestamp using a comma as the delimiter.

End timestamps consider milliseconds. For example, if '11:00:00 - 11:59:59' is specified for the timestamp, "11:00:00:000" to "11:59:59:999" will be the target range.

The timestamps used by evaluation in the rule section of pgaudit are different to the timestamps issued in the log entries. That is because log entries are output after evaluation by pgaudit, with the timestamp being generated at that time.

class

The meaning of each class specified in the class parameter is below:

- READ: SELECT, COPY FROM
- WRITE: INSERT, UPDATE, DELETE, TRUNCATE, COPY TO
- FUNCTION: Function call, DO
- ROLE: GRANT, REVOKE, CREATE ROLE, ALTER ROLE, DROP ROLE
- DDL: All DDLs (such as CREATE and ALTER) other than the DDLs of the ROLE class
- CONNECT: Events relating to connecting (request, authenticate, and disconnect)
- SYSTEM: Instance start, promotion to primary server
- BACKUP: pg_basebackup
- ERROR: Event completed by an error (PostgreSQL error codes other than 00). This class can be used if ERROR or lower level is specified for the log_min_messages parameter in postgresql.conf.
- MISC: Other commands (such as DISCARD, FETCH, CHECKPOINT, and VACUUM)

Evaluation of the rule section

- When a log event occurs, all expressions in the rule section are evaluated at once. Log entries are only output if all parameters in the rule section are evaluated as being true.

  For example, if the rule below has been set, of the operations performed by 'apserver' to 'myschema.tbl1', the operations applicable to classes other than 'WRITE'' in the period from 10 a.m. to 11 a.m. will be output as audit logs.

  ```
  [rule]
  timestamp = '10:00:00-11:00:00'
  remote_host = 'apserver'
  object_name = 'myschema.tbl1'
  class != 'WRITE'
  ```

- Multiple rule sections can be defined in the pgaudit configuration file. Log events are evaluated using each rule section, and an audit log is output for each matching rule section.

  For example, if the rules below are set, duplicated audit logs will be output.

  ```
  [rule]
  object_name = 'myschema.tbl1'
  [rule]
  object_name = 'myschema.tbl1'
  ```

- If the same parameter is specified multiple times in one rule section, the last specified parameter is effective.

  For example, if the rule below has been set, "object_name = 'myschema.tbl3'" will take effect.

  ```
  [rule]
  object_name = 'myschema.tbl1'
  object_name = 'myschema.tbl2'
  object_name = 'myschema.tbl3'
  ```

Output format

In Session Audit Logging, audit logs are output in the format below:

```
AUDIT: SESSION,READ,2022-03-12 19:00:58 PDT,
   (1)         (2)     (3)
[local],19944,psql,appuser,postgres,2/8, 2, 1,SELECT,,TABLE,myschema.account, ,
  (4)    (5)   (6)   (7)      (8)   (9)(10)(11)(12)(13)(14)    (15)        (16)
SELECT * FROM myschema.account;,<not logged>
       (17)                      (18)
```

No	Content
(1)	Log header Fixed as "AUDIT: SESSION".
(2)	Class
(3)	SQL start time
(4)	Remote host name If using a local host, [local] is output.
(5)	Backend process ID
(6)	Application name If an application name is not specified, [unknown] is output.
(7)	User name
(8)	Database name
(9)	Virtual transaction ID
(10)	Statement ID
(11)	Substatement ID
(12)	Command tag
(13)	SQLSTATE
(14)	Object type
(15)	Object name
(16)	Error message
(17)	SQL If the SQL contains a password, such as for CREATE ROLE, and so on, it will be replaced with "<REDACTED>". Additionally, if "on" is specified for the log_statement_once parameter of the option section in the pgaudit configuration file, "<previously logged>" is output for the second and subsequent statements.
(18)	Depending on the log_parameter parameter value of the option section in the pgaudit configuration file, the output content will be as below. - log_parameter=on is specified If parameters are specified in the SQL, the parameters are concatenated and output, using a space as the delimiter. If parameters are not specified in the SQL, "<none>" is output. - log_parameter=off (default) is specified "<not logged>" is output. Additionally, if "on" is specified for the log_statement_once parameter of the option section in the pgaudit configuration file, "<previously logged>" is output for the second and subsequent statements.

Information

If accessing resources that use the features below, the command tag (12) may be output as "???".

- INSTEAD OF trigger
- RULE

- VIEW
- Security policy per row
- Table inheritance

Example

Below is an example of retrieving audit logs in Session Audit Logging.

1. Settings

   In the pgaudit configuration file, specify the rule section below.

```
[rule]
class = 'READ, WRITE'
object_name = 'myschema.account'
```

2. Retrieving logs

   Execute the SQL below from the client.

```
CREATE TABLE myschema.account
(
    id int,
    name text,
    password text,
    description text
);
INSERT INTO myschema.account (id, name, password, description) VALUES (1, 'user1', 'HASH1', 'blah,
blah');
SELECT * FROM myschema.account;
```

   The audit log below can be retrieved.

   'DDL' is not defined in the class parameter, so CREATE TABLE is not output as an audit log.

```
AUDIT: SESSION,WRITE,2022-03-12 19:00:49 PDT,[local],19944,psql,appuser,postgres,
2/7,1,1,INSERT,,TABLE,myschema.account,,"INSERT INTO myschema.account (id, name, password,
description) VALUES (1, 'user1', 'HASH1', 'blah, blah');",<not logged>
AUDIT: SESSION,READ,2022-03-12 19:00:58 PDT,[local],19944,psql,appuser,postgres,
2/8,2,1,SELECT,,TABLE,myschema.account,,SELECT * FROM myschema.account;,<not logged>
```

6.5 Object Audit Logging

In Object Audit Logging, retrieval of audit logs is achieved by using roles.

Roles are specified in the role parameter of the option section to retrieve audit logs. If there are privileges for commands executed by a role, or if privileges have been inherited from another role, those command operations are output as audit logs.

For example, after "auditor" is set for the role parameter of the option section, the SELECT and DELETE privileges for the account table are assigned to "auditor". In this case, when SELECT or DELETE is executed for the account table, audit logs are output.

Output format

In Object Audit Logging, audit logs are output in the format below:

```
AUDIT: OBJECT,1,1,READ,SELECT,TABLE,public.account,SELECT password FROM account;,<not logged>
     (1)    (2)(3)(4)  (5)    (6)        (7)               (8)                    (9)
```

No	Content
(1)	Log header Fixed as "AUDIT: OBJECT".
(2)	Statement ID
(3)	Substatement ID
(4)	Class name
(5)	Command tag
(6)	Object type
(7)	Object name
(8)	SQL If "on" is specified for the log_statement_once parameter of the option section in the pgaudit configuration file, "<previously logged>" is output for the second and subsequent statements.
(9)	Depending on the log_parameter parameter value of the option section in the pgaudit configuration file, the output content will be as below. - When log_parameter=on If parameters are specified in the SQL, the parameters are concatenated and output, using a comma as the delimiter. If parameters are not specified in the SQL, "<none>" is output. - When log_parameter=off (default) "<not logged>" is output. Additionally, if "on" is specified for the log_statement_once parameter of the option section in the pgaudit configuration file, "<previously logged>" is output for the second and subsequent statements.

Information

If accessing resources that use the features below, the command tag (5) may be output as "???".

- INSTEAD OF trigger
- RULE
- VIEW
- Security policy per row
- Table inheritance

Example

Below is an example of retrieving logs in Object Audit Logging.

By setting the target for assigning privileges to roles in detail, log output can be controlled.

In the example below, log retrieval of the account table is controlled by the privileges assigned to the columns, however, log retrieval of the account_role_map table is controlled by the privileges assigned to the table.

1. Settings

The role parameter below is specified for the option section in the pgaudit configuration file.

```
[option]
role = 'auditor'
```

2. Defining a role

A role is defined for Object Audit Logging.

```
CREATE USER auditor NOSUPERUSER LOGIN;
```

3. Retrieving logs

Execute the SQL below from the client.

```
CREATE TABLE account
(
    id int,
    name text,
    password text,
    description text
);
GRANT SELECT (password) ON public.account TO auditor;
SELECT id, name FROM account;
SELECT password FROM account;
GRANT UPDATE (name, password) ON public.account TO auditor;
UPDATE account SET description = 'yada, yada';
UPDATE account SET password = 'HASH2';
CREATE TABLE account_role_map
(
    account_id int,
    role_id int
);
GRANT SELECT ON public.account_role_map TO auditor;
SELECT account.password, account_role_map.role_id
  FROM account
  INNER JOIN account_role_map ON account.id = account_role_map.account_id;
```

The audit log below can be retrieved.

In the account table, only the operations for columns that privileges have been assigned to are output as logs.

In the account_role_map table, privileges are assigned to the table, so operations performed for the table are output as logs.

```
AUDIT: OBJECT,4,1,READ,SELECT,TABLE,public.account,SELECT password FROM account;,<not logged>
AUDIT: OBJECT,7,1,WRITE,UPDATE,TABLE,public.account,UPDATE account SET password = 'HASH2';,<not
logged>
AUDIT: OBJECT,10,1,READ,SELECT,TABLE,public.account,"SELECT account.password,
account_role_map.role_id
        FROM account
        INNER JOIN account_role_map ON account.id = account_role_map.account_id;",<not logged>
AUDIT: OBJECT,10,1,READ,SELECT,TABLE,public.account_role_map,"SELECT account.password,
account_role_map.role_id
        FROM account
        INNER JOIN account_role_map ON account.id = account_role_map.account_id;",<not logged>
```

6.6 Database Multiplexing

This section describes audit log retrieval while in database multiplexing mode.

6.6.1 Setup

If setting up the audit log feature in a database multiplexing environment that has already been built, follow the procedure below.

1. Copy the pgaudit files

   Copy the pgaudit files on the primary server and standby server.

   Refer to step 1 in "6.2 Setup" for details on copying the pgaudit files.

2. Create the pgaudit configuration file

   Create the pgaudit configuration file on the primary server. Copy the pgaudit configuration file you created to the standby server.

   Refer to step 2 in "6.2 Setup" for details on creating the pgaudit configuration file.

3. Configure postgresql.conf

   In the postgresql.conf file on the primary server and standby server, configure the parameters for using audit logs. Set the same values for the parameters.

   Refer to step 3 in "6.2 Setup" and "6.6.2 Configuring Audit Log Retrieval" for details on the parameters to configure.

4. Configure the *serverIdentifier*.conf file of Mirroring Controller

   In the *serverIdentifier*.conf file on the primary server and standby server, configure the parameters for using audit logs.

   Refer to "6.6.2 Configuring Audit Log Retrieval" for details on the parameters to be set.

5. Start the instance

   Start the instance of the primary server and standby server.

6. Create the pgaudit extension

   Execute CREATE EXTENSION on the primary server to create a pgaudit extension.

   Refer to step 5 in "6.2 Setup" for details on creating pgaudit extensions.

7. Configure the parameters in the pgaudit configuration file

   Add/change the parameters of the pgaudit configuration file on the primary server. Copy the pgaudit configuration file with the added/changed parameters to the standby server.

   Refer to "6.3 pgaudit Configuration File" and "6.6.2 Configuring Audit Log Retrieval" for details on the parameters to set.

8. Restart the instance

   Restart the instance of the primary server and standby server.

6.6.2 Configuring Audit Log Retrieval

In database multiplexing mode, Mirroring Controller periodically accesses the database to check the multiplexing status and detect failure. Due to this, audit logs are also periodically retrieved, so log files become used up. Therefore, set the parameters below so that audit logs are not retrieved by Mirroring Controller.

postgresql.conf

log_connections

Omit, or specify "off".

log_disconnections

Omit, or specify "off".

serverIdentifier.conf file of Mirroring Controller

target_db

Specify "template1".

Note

If creating a new database, create it after stopping Mirroring Controller, or specify a name other than "template1" for the template database.

pgaudit configuration file

option section audit_log_disconnections

If you want to output disconnection logs for connections other than Mirroring Controller, specify audit_log_disconnections = "on".

rule section database

Specify database != 'template1'.

6.7 View Audit Logs Using SQL

By using file_fdw of an additional module, audit logs can be accessed using SQL. This section describes how to view audit logs, using Session Audit Logging output to a dedicated log file as an example.

1. Install file_fdw

   Execute CREATE EXTENSION to install file_fdw as an extension.

```
$ psql
=# CREATE EXTENSION file_fdw;
=# \dx
                  List of installed extensions
Name     | Version |   Schema   |     Description
---------+---------+------------+-----------------------------------------
file_fdw | 1.0     | public     | foreign-data wrapper for flat file access
pgaudit  | 1.0     | public     | provides auditing functionality
plpgsql  | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

2. Create an external server

   Use CREATE SERVER to create an external server managed by file_fdw.

```
$ psql
=# CREATE SERVER auditlog FOREIGN DATA WRAPPER file_fdw;
```

3. Create an audit log table.

   Use CREATE FOREIGN TABLE to define the table columns of audit logs, CSV file name and format.

```
$ psql
=# CREATE FOREIGN TABLE auditlog (
header text,
class text,
sql_start_time timestamp with time zone,
remote_host_name text,
backend_process_id text,
application_name text,
session_user_name text,
database_name text,
virtual_transaction_id text,
statement_id text,
substatement_id text,
command_tag text,
sqlstate text,
object_type text,
object_name text,
error_message text,
```

```
sql text,
parameter text
) SERVER auditlog
OPTIONS ( filename '/database/inst1/pgaudit_log/pgaudit-2022-03-12.log', format 'csv' );
```

Note

If an audit log file is rotated and multiple audit log files exist, it is necessary to create a table for each audit log file.

4. View audit logs

   Use SELECT and view the audit logs.

```
$ psql
=# SELECT * FROM auditlog;
     header     |  class  |     sql_start_time     | remote_host_name | backend_process_id ...
----------------+---------+------------------------+------------------+------------------- ...
AUDIT: SESSION  | WRITE   | 2022-03-12 19:00:49+09 | [local]          | 19944              ...
AUDIT: SESSION  | READ    | 2022-03-12 19:00:58+09 | [local]          | 19944              ...
```

6.8 Removing Setup

This section describes how to remove the setup of pgaudit.

1. Start the instance
2. Remove the pgaudit extension

   Execute DROP EXTENSION to remove the pgaudit extension from the database.

```
$ psql -d <database name>
=# DROP EXTENSION pgaudit;
=# \q
```

3. Change postgresql.conf

   Remove the parameters below relating to pgaudit.

   - shared_preload_libraries
   - pgaudit.config_file

4. Restart the instance
5. Remove the pgaudit files

   As superuser, run the following command. Note that "<*x*>" in paths indicates the product version.

```
$ su -
Password:******
# rm -rf /opt/fsepv<x>server64/filesCopiedDuringSetup
```

Information

The files copied during setup can be checked below.

```
# find /opt/fsepv<x>server64/OSS/pgaudit
```

Chapter 7 Confidentiality Management

In order to prevent unauthorized use of the various data stored in the database, it is necessary to set appropriate privileges for each database resource for database users. However, there are multiple users to whom privileges are granted, and there are also many target database resources. So it takes a lot of effort to set it up. The confidentiality management feature reduces the time and effort required to do so and supports the setting and maintenance of appropriate privileges.

This chapter describes how to install, design and use confidentiality management features. For usage examples of the feature, refer to "7.8 Usage Example of Confidentiality Management".

7.1 Setup

This feature is provided as an EXTENSION of PostgreSQL. The name is pgx_confidential_management_support. Register the extension with the database cluster using the CREATE EXTENSION statement as follows:

This must be run by superuser. Because this extension registers PostgreSQL event triggers. By registering an event trigger, when a database object such as a table is deleted, related information is deleted from the information managed by the confidentiality management feature.

This extension can be created for any schema.

```
CREATE EXTENSION pgx_confidential_management_support
```

Note

- The various definitions described below are registered in the tables included in this extension. Note that dropping this extension with the DROP EXTENSION statement will therefore drop all these definitions as well.
- If the superuser also serves as the confidentiality management role, or if you have only one confidentiality management role, setup is complete. For confidentiality management roles, refer to "7.2.2 Determining Confidentiality Management Roles". However, if multiple confidentiality management roles manage different confidentiality matrices, they must be prevented from manipulating each other's confidentiality matrices. To do so, a superuser should run the script provided by the product as follows. This script defines policies for the row level security feature on the tables provided by the confidentiality management. ${install_dir} refers to the directory where you installed the product.

  ```
  psql -f ${install_dir}/share/extension/pgx_confidential_management_support_policy.sql
  ```

- 'public' is granted SELECT on tables included by this extension when CREATE EXETENSION statement is executed. Don't revoke the privilege from 'public'. For example, event trigger of the extension confirms to update its tables when a general user drop some table. The privilege is required at the time. Also, there is no problem that users refer content of the tables like pg_catalog.

7.2 Designing Confidentiality Management

Describes the design of confidentiality management.

7.2.1 Designing a Confidentiality Matrix

A confidentiality matrix is a matrix of confidentiality levels and confidentiality groups. Elements of the confidentiality matrix are confidentiality privileges. Here we define them.

A confidentiality level is a group of data with the same degree of confidentiality, and a confidentiality group is a group of roles with the same access to confidentiality data. The details are described below.

The end result is to classify database objects into confidentiality levels and roles into confidentiality groups. But for the first step, define your confidentiality matrix abstractly without thinking about concrete tables, roles, etc. By doing so, the relationship between confidentiality levels and confidentiality groups can also be applied to databases with different sets of database objects.

You can define multiple confidentiality matrices. These confidentiality matrices are identified by a name, but the name is unique within the database (not the database cluster).

A role that manages the confidentiality matrix is called a confidentiality management role. The details are described below.

How to create multiple confidentiality matrices

There are various design methods, but an example is shown below.

- Create one confidentiality matrix for a dataset

  For example, create one confidentiality matrix for one section of your organization.

- Copy the confidentiality matrix if the two sections have the same confidentiality design
- When multiple organizations share datasets, create a single confidentiality matrix for the shared datasets

  An example of this case is shown below.

  - Create a confidentiality matrix *A* for the dataset that only section 1 has access to.
  - Create a confidentiality matrix *B* for the dataset that only section 2 has access to.
  - Create a confidentiality matrix *S* for the dataset shared by section 1 and section 2.
  - Section 1 administrator *M1* is in charge of the confidentiality management role for confidentiality matrix *A*.
  - Section 2 administrator *M2* is in charge of the confidentiality management role for confidentiality matrix *B*.
  - Role group *G* to which *M1* and *M2* belong is in charge of confidentiality management role of confidentiality matrix *S*.

7.2.1.1 Defining Confidentiality Levels

Confidentiality level is a concept that indicates degree of confidentiality. For example, determine a confidentiality level that classifies tables that contain personal information and those that do not.

A collection of data that you classify as confidentiality level is a database object, such as a table. Such objects are called confidentiality objects. Refer to "What is a confidentiality objects?" for confidentiality objects.

Attributes can be set for the confidentiality level. For example, if a confidentiality level has an attribute that requires encryption, tables belonging to that confidentiality level are required to be encrypted. The confidentiality management feature may or may not automatically change the attribute of the confidentiality object to which it belongs. Also, each attribute targets a different confidentiality object.

The table below shows the attribute description, the confidentiality object targeted, and whether to change the attribute of the confidentiality object automatically.

Confidentiality level attributes	Specifiable values	Confidentiality object type	Auto change	Description
encryption_algorithm	AES128 AES256 none	table column rowset	No	Requires confidentiality objects to be encrypted with the specified algorithm. For details, refer to "Protecting Storage Data Using Transparent Data Encryption" in the Operation Guide. If ***none*** is specified, no encryption is requested. Default is ***none***. The attribute is not changed autlmatically because encryption and decryption involve large data updates. Instead, you cannot add a confidentiality object with an encryption strength lower than the specified encryption strength. Also, when changing to increase the encryption strength, confidentiality objects that do not meet the post-change conditions must not be included.

What is a confidentiality objects?

Confidentiality objects have the types shown in the following table. For example, some data contained in table *T* can be added to confidentiality level *L1* and other data to confidentiality level *L2*.

Confidentiality object type	Description
schema	It does not mean all the tables contained in the schema. It simply means an object that exists in the system catalogs. Means the schema itself that is the target of access control by the GRANT statement.
table	Includes views, materialized views and partitions as well as tables. Currently do not support foreign tables.
column	Means column.
rowset	A set of rows that satisfy a specified condition. Although there is no rowset object in the PostgreSQL system catalog, it is treated as a confidentiality object in the confidentiality management feature for ease of use. Access control for rowset-type confidentiality objects uses PostgreSQL's row-level security functionality internally. Therefore, how access is controlled follows the row-level security specification.

The following objects may also contain confidential information. However, the current confidentiality management feature does not support them as confidentiality objects. When managing these, it is recommended to manage them in a role different from the confidentiality management role. The details are described in "7.5 Suggestions for Monitoring Methods".

- Function
- Procedure
- Foreign server
- Foreign data wrapper
- Foreign table

7.2.1.2 Defining Confidentiality Groups

A confidentiality group is an object that indicates which confidentiality level roles in confidentiality group can acdess to. For example, decide which role groups have access to personal information and which groups do not. Confidentiality groups are actually automatically defined as PostgreSQL roles (role groups). This role is called a confidentiality group role.

What is a confidentiality group role?

Confidentiality group roles are targets of GRANT and REVOKE statements executed internally by the confidentiality management feature. And when you add a role to the confidentiality group, the added role becomes a member of the confidentiality group role. The privileges of the added role are given by inheriting the privileges given to the confidentiality group role by the confidentiality management feature.

The following attributes can be set for the confidentiality group role via the function that creates the confidentiality group. The meaning and default value of attribute are the same as in the CREATE ROLE statement specification. These are limited to attributes that may be necessary when managing confidentiality. Therefore, use the confidentiality management feature to change it. The confidentiality management feature automatically sets the following attributes for the role added to the confidentiality group.

- SUPERUSER
- CREATEDB
- CREATEROLE
- REPLICATION
- BYPASSRLS

Other attributes follow the PostgreSQL CREATE ROLE statement defaults. Even if the attribute is changed using the ALTER ROLE statement, the operation of confidentiality management feature will not be disturbed.

The naming convention for confidentiality group roles is as follows.

```
pgx_cgroup_role_${serial_number}
```

When deleting a confidentiality group, you can choose not to delete the confidentiality group role. In such a case, it is a means to know that it was a confidentiality group role later.

Do not define roles using strings that follow this naming convention. This is because the name of the new confidentiality group role will be the number next to the maximum *${serial_number}* of the role that follows this naming convention. For example, if a role named 'pgx_cgroup_role_999999999999999999' exists, you will not be able to create new confidentiality group roles.

7.2.1.3 Defining Confidentiality Privilege

Confidentiality privileges indicate how a confidentiality group can access confidentiality objects registered at a certain confidentiality level.

For example, a confidentiality group *R* can be defined to have SELECT, INSERT, UPDATE, and DELETE privileges on table-type confidentiality objects belonging to confidentiality level *L*.

Once a confidentiality object or role has been added to a confidentiality level or confidentiality group, the feature automatically uses the GRANT or CREATE POLICY statement to grant privileges to the actual database object according to the confidentiality privilege definition to the confidentiality group role.

For rowset types, confidentiality privileges are those that can be specified in the CREATE POLICY statement. In cases other than rowset, it is a privilege that can be specified in the GRANT statement. For detailed privileges, refer to the respective SQL statement reference in the PostgreSQL Documentation.

Do not change the definition of POLICY that this feature creates internally. Even if you change it, the function of this feature may return to the state before the change without warning.

For roles registered in the confidentiality matrix, if the privilege to a certain confidentiality object is more than the confidentiality privilege, revoke those privileges according to the confidentiality privilege.

If a role not registered in the confidentiality matrix or PUBLIC was granted privileges to a confidentiality object registered in the confidentiality matrix, functions of this feature for adding confidentiality objects or roles will fail. Because this function is not allowed to manage.

Even in a slightly more complicated situation, failing functions that add confidentiality objects and roles prevent roles added to the confidentiality matrix from deviating from confidentiality privileges:

- It is set so that a role registered in the confidentialmatrix can inherit a role not registered in the confidentiality matrix, and as a result, it has more privileges than the confidentiality privileges.
- Roles registered in the confidentiality matrix are granted privileges by a POLICY not created by this feature. In this case, the function will fail if such a POLICY exists without rigorously checking for violations of confidentiality privilege.

7.2.2 Determining Confidentiality Management Roles

The confidentiality management role performs all operations on the confidentiality matrix. As such, the confidentiality management role requires very strong privileges to either:

1. Has all of the following rights

   - Superuser privileges
   - Ownership of database objects belonging to the confidentiality level

2. Has all of the following privileges

   - CREATEROLE privilege
   - SELECT, INSERT, UPDATE, and DELETE privileges on all tables included in the extension 'public' is granted SELECT when CREATE EXTENSION statement is executed.
   - Ownership of database objects belonging to the confidentiality level

Note that the previous owner may not be able to execute GRANT statements, etc. once the confidentiality management role becomes the owner. This is because ownership of database objects cannot be shared by multiple roles belonging to different role groups. This is the PostgreSQL specification.

The confidentiality management role can create and manage multiple confidentiality matrices, but it is safer to distribute the authority. For that reason, we recommend that you decide on confidentiality management rules in the following order of priority.

- Create one-to-one confidentiality matrices and confidentiality management roles.
- Create a role group with multiple confidentiality management roles as members, or manage multiple confidentiality matrices with one confidentiality management role.
- A superuser is also a confidentiality management role.

To assign attributes that only a superuser can grant to a role managed using the confidentiality management feature, the superuser must also serve as the confidentiality management role. For example, the REPLICATION attribute is such an attribute. Refer to the reference of the PostgreSQL Documentation for details.

7.2.3 Classify Confidentiality Objects According to the Definition of Confidentiality Level

7.2.3.1 Defining Confidentiality Objects

First, refer to list of definitions of schemata, tables in your database, etc. that you want to manage, and define confidentiality objects. As mentioned earlier, there are various types of confidentiality objects such as column type and rowset type, so you can define confidentiality objects based on their data content.

Access control for rowset-type confidentiality objects uses PostgreSQL's row-level security functionality internally. Therefore, the method of specifying a set of rows follows the specifications of the AS clause, USING clause, and WITH CHECK clause of the CREATE POLICY statement. Also, the definition of the rowset-type confidentiality object is specified in the argument of the function that registers the rowset-type confidentiality object to the confidentiality level. This feature executes the ALTER TABLE statement with the ENABLE ROW LEVEL SECURITY clause to enable POLICY when a rowset-type confidentiality object is added.

Currently it is not possible to register a foreign table as a table-type confidentiality object.

Recommend that you do not grant privileges on confidentiality objects to PUBLIC. Granting privileges to PUBLIC is the same as granting privileges to all roles registered in the confidentiality matrix. This makes no sense if all roles that access confidentiality objects are managed

using this feature. If PUBLIC is granted privileges to confidentiality objects, the functions included in this feature will check that the privileges granted to each role are not exceeded, and will fail if they are.

Note

- Be careful when confidentiality objects are of column type. This is because if the table-type confidentiality object is set at the same time, the table type privilege takes precedence.

  For example, if you want to revoke the SELECT privilege only from a special column *C* in some table *T*, list the columns other than column *C* and grant the SELECT privilege to them without granting the SELECT privilege to table *T*. This follows the PostgreSQL's GRANT statement specification for column.

  If you have to enumerate a large number of columns, it might be a good idea to move the special columns to a new table and present your existing application with a VIEW that JOINs both tables.

- If the confidentiality object is of rowset type, set the same privileges for the table type as those specified for the rowset type. This is because when a SQL statement accesses data, it first checks that you have privilege to access the table. After that, the rowset privileges are checked for matching rows. This follows the PostgreSQL row level security specification.

7.2.3.2 Classify Confidentiality Objects

Classify confidentiality objects according to the confidentiality level definition.

A single confidentiality object cannot be classified in multiple confidentiality levels, even if they are in different confidentiality matrices.

7.2.4 Classify Roles According to Confidentiality Group Definitions

A role can belong to multiple confidentiality groups in different confidentiality matrices at the same time. However, a single role cannot be classified into multiple confidentiality groups within a single confidentiality matrix.

Note

- We strongly recommend that you do not group confidentiality group roles into other confidentiality groups. This is because the confidentiality management feature does not prohibit such usage, but it probably only complicates security management. Also note that PostgreSQL does not allow cyclic groupings of roles.

- After adding roles to a confidentiality group, it is highly recommended that such roles not be made members of role groups not managed by the confidentiality matrix. This is because although the confidentiality management feature does not prohibit such situations, for example, unreasonably increasing the privileges of such role groups may become a way out for roles managed by the confidentiality management feature.
  Note that this feature does not allow roles with such loopholes to be added to confidentiality groups.

7.3 How to Use Confidentiality Management Feature (Definition)

Confidentiality management feature provides functions and tables.

Some functions define, change, or drop confidentiality matrices, etc. These functions will output an audit log indicating that they were executed, so you can later confirm that an illegal operation was performed.

In addition, you can directly refer to the table provided by this feature and check the defined contents using functions that help referencing.

Also, some functions output the attributes of confidentiality objects defined in the PostgreSQL system catalog and the attributes that should be set for that confidentiality object. You can use these to compare attributes.

For details, refer to "Appendix B System Management Functions Used by Confidentiality Management Feature".

Tables included by the pgx_confidential_management_support extension

Table name	Description
pgx_confidential_matrix	A list of confidentiality matrices. You can refer to the attributes of the registered confidentiality matrix, the update time, or the time when the confidentiality level or confidentiality group was registered or deleted.
pgx_confidential_level	A list of confidentiality levels. You can refer to the registered confidentiality level attributes, update time, or the time when a confidentiality object was registered to the confidentiality level or removed from the confidentiality level.
pgx_confidential_group	A list of confidentiality groups. You can refer to the registered confidentiality group attributes, update time, or the time when a role was registered to the confidentiality group or removed from the confidentiality group.
pgx_confidential_privilege	A list of confidentiality privileges. You can refer to confidentiality privilege set for each confidentiality object, update time, and so on.
pgx_confidential_object	A list of confidentiality objects. You can refer to object attributes or update time, and so on.
pgx_confidential_role	A list of roles registered in the confidentiality group. You can refer to role attributes or update time, and so on.
pgx_confidential_policy	This is a list of policies created to set privileges for rowset-type confidentiality objects. You can refer to the name of the policy you created and the privileges it has set. Rows in this table are inserted when you add a rowset-type confidentiality object.

Rows in each table are added, deleted, or updated when you execute functions to add, delete, or update definitions.

Also, if the target confidentiality object is deleted by a DROP TABLE statement, etc., it will also be deleted from the following tables.

- pgx_confidential_object
- pgx_confidential_policy

Functions for adding, removing or updating definitions

These functions will print an audit log indicating that they were executed.

Function name	Description
pgx_create_confidential_matrix	Create a confidentiality matrix.
pgx_alter_confidential_matrix	Change the attributes of the confidentiality matrix.
pgx_drop_confidential_matrix	Drop the confidentiality matrix.
pgx_copy_confidential_matrix	Copy the confidentiality matrix.
pgx_create_confidential_level	Create confidentiality levels and register them in the confidentiality matrix.
pgx_alter_confidential_level	Change the confidentiality level attribute.
pgx_drop_confidential_level	Remove a confidentiality level from the confidentiality matrix and drop a confidentiality level.

Function name	Description
pgx_create_confidential_group	Create a confidentiality group and register it in the confidentiality matrix.
pgx_alter_confidential_group	Change the attributes of a confidentiality group.
pgx_drop_confidential_group	Remove confidentiality groups from the confidentiality matrix and drop confidentiality group.
pgx_grant_confidential_privilege	Grant confidentiality privileges.
pgx_revoke_confidential_privilege	Revoke confidentiality privileges.
pgx_add_object_to_confidential_level	Add confidentiality objects to the confidentiality level.
pgx_remove_object_from_confidential_level	Removes confidentiality objects from the confidentiality level.
pgx_add_role_to_confidential_group	Add a role to a confidentiality group.
pgx_remove_role_from_confidential_group	Remove a role from a confidentiality group.

Functions that Support Definition Referencing and Comparison with System Catalogs

Function name	Description
pgx_get_attribute_of_objects	For all confidentiality objects registered in the specified confidentiality matrix, displays the attributes defined in the confidentiality matrix and the attributes actually set in the database.
pgx_get_attribute_of_roles	For all roles registered in the specified confidentiality matrix, displays the attributes defined in the confidentiality matrix and the attributes actually set in the system catalog.
pgx_get_privileges_on_level_and_group	Displays a list of combinations of confidentiality objects registered with the specified confidentiality level and roles registered with the specified confidentiality group, such that the following can be compared. - Privileges specified by the confidentiality privilege settings that should be granted to the specified role - Privileges granted in the actual system catalog
pgx_get_privileges_on_object	For the specified confidentiality object, display a list that allows you to compare the following: - Privileges dictated by confidentiality privilege settings that should be granted to all roles - Privileges granted in the actual system catalog
pgx_get_privileges_on_role	For all confidentiality objects, display a list that allows you to compare: - Privileges specified by the confidentiality privilege settings that should be granted to the specified role - Privileges granted in the actual system catalog
pgx_get_privileges_on_matrix	For all objects registered in the specified confidentiality matrix, display a list that allows you to compare: - Privileges defined by confidentiality privilege settings that should be granted to all roles registered in the confidentiality matrix.

Function name	Description
	- Privileges granted in the actual system catalog

Describe the definition procedure.

7.3.1 Creating a Confidentiality Management Role

Create a confidentiality management role using the CREATE ROLE statement, or use an existing role as a confidentiality management role. Set the privileges and attributes shown in "7.2.2 Determining Confidentiality Management Roles" for the confidentiality management role.

Note

- Use caution when renaming a confidentiality management role. If you want to do so, delete all confidentiality matrices managed by that confidentiality management role, rename the confidentiality management role, and then define the same confidentiality matrix again. Otherwise, you will not be able to operate the confidentiality matrix. For example, you cannot change confidentiality privileges or remove confidentiality objects from the confidentiality level. If you accidentally renamed it first, change it back to the original name and then proceed as described above. In the future, we will simply allow the changed name of the confidentiality management role to be set in the confidentiality matrix.
- If you want to delete the confidentiality management role, delete the confidentiality management role after deleting the confidentiality matrix. Otherwise, you will not be able to create a confidentiality matrix with the same name because you will not be able to delete the confidentiality matrix left behind. If you accidentally delete the confidentiality management role before deleting the confidentiality matrix, create a confidentiality management role with the same name again and delete the confidentiality matrix, or delete the confidentiality matrix with a role that has SUPERUSER privileges.

7.3.2 Creating a Confidentiality Matrix

Create a confidentiality matrix with comments describing the confidentiality matrix as follows. This comment is stored in the pgx_confidential_matrix table.

```
select pgx_create_confidential_matrix('matrix_foo', 'This matrix is defined for foo.')
```

Point

The role executing this function is considered the confidentiality management role. If this function is executed after SET ROLE, the role specified in SET ROLE will be regarded as the confidentiality management role, not the role that executed SET ROLE.

You can also create a new confidentiality matrix 'matrix_dest' by duplicating the already created confidentiality matrix 'matrix_src' as follows. If you copy the confidentiality matrix and create it, the comments will also be copied. Comments can be changed using the pgx_alter_confidential_matrix function.

```
select pgx_copy_confidential_matrix('matrix_dest', 'matrix_src')
```

You can also check the created matrix by referring to the pgx_confidential_matrix table.

Point

The source must be within the same database. If you want to copy from different databases or different database instances, choose one of the following methods. If you choose the method using COPY statement, confirm the cautions shown below.

- Define using a function in the same way as the original.
- Perform the following steps

  Execute steps other than step 6 with a role that has SUPERUSER privileges.

1. Set up extensions in the target database.
2. From the table below, specify the ID of the source confidentiality matrix in the WHERE clause and extract the data using the COPY TO statement.
   - pgx_confidential_matrix
   - pgx_confidential_level
   - pgx_confidential_group
   - pgx_confidential_privilege
3. Refer to the *pg_sequences* system view and save the value of the *last_value* column of the following SEQUENCE in a file.
   - pgx_confidential_matrix_cmatid_seq
   - pgx_confidential_level_clevid_seq
   - pgx_confidential_group_cgroid_seq
   - pgx_confidential_privilege_cpriid_seq
4. Load the data extracted in step 2 into the above table in the target database using the COPY FROM statement. This operation can be performed by any confidentiality management role in any confidentiality matrix.
5. Use the *pg_catalog.setval* function to set the *last_value* for each SEQUENCE on the copy destination, specifying the value saved in step 3. An example is shown below.

```
SELECT pg_catalog.setval('pgx_confidential_matrix_cmatid_seq', 5)
```

6. Create a new confidentiality matrix using the *pgx_copy_confidential_matrix* function and specifying the confidentiality matrix loaded in step 4 as the copy source. Perform this operation in the confidentiality management role of the newly created confidentiality matrix.
7. Use the *pgx_drop_confidential_matrix* function specifying ***false*** for *drop_role* to drop the confidentiality matrix loaded in step 4.

Note

- If you specify true for *drop_role*, you will not be able to continue to be managed by the original confidentiality matrix. The reason is as follows.

  After completing step 6, you should have:

  a) Confidentiality matrix of source database

  b) the confidentiality matrix loaded in step 4

  c) Confidentiality matrix duplicated in step 6

  The b) and c) confidentiality matrices use different confidentiality group roles. This is the effect of the *pgx_copy_confidential_matrix* function. However, the confidentiality matrices of a) and b) share the same confidentiality group role. Although this is temporary, it is a bad situation. To do so, remove the confidentiality matrix in b) in step 7. If *true* is specified for *drop_role* at this time, the confidentiality group role used in a) will be deleted. Therefore, management by the confidentiality matrix of a) cannot be continued.

- The destination must have just installed this extension with CREATE EXTENSION. This is because the table data contained in this extension contains an ID representing the confidentiality matrix and confidentiality level, and this ID is generated by the SEQUENCE included in the extension. For example, if you have created at least one confidentiality matrix, the confidentiality matrix is numbered with an ID of 1, but that matrix is not the object numbered with an ID of 1 in the copy source.

7.3.3 Adding Confidentiality Levels to the Confidentiality Matrix

Create a confidentiality level and add it to the confidentiality matrix as follows: In the example below, the JSON format third argument requests that confidentiality objects belonging to 'level1' are encrypted. Specify *NULL* if you don't want anything. Refer to "B.2 Confidentiality Level Manipulation Functions" for the content of the request.

```
select pgx_create_confidential_level('matrix_foo', 'level1',  '{"encryption_algorithm":"AES256"}',
'The strongest encryption is required for this level.')
```

You can also check the added confidentiality level by referring to the pgx_confidential_level table.

7.3.4 Adding Confidentiality Groups to the Confidentiality Matrix

Create a confidentiality group and add it to the confidentiality matrix as follows. Attributes are given to the confidentiality group by the third argument in JSON format. Refer to "B.3 Confidentiality Group Manipulation Functions" for the attributes that can be assigned.

In the example below, a role belonging to this confidentiality group grants the CREATEDB privilege.

```
select pgx_create_confidential_group('matrix_foo', 'group1', '{"CREATEDB":true}', 'Roles belonging
to this confidentiality role are permitted to create db.')
```

You can also check the added confidentiality group by referring to the pgx_confidential_group table.

7.3.5 Granting Confidentiality Privileges to Confidentiality Groups

In the example below, roles belonging to 'group1' are granted SELECT, INSERT and DELETE privileges for tables belonging to 'level1'. Specify privileges for each type of confidentiality object by the fourth argument in JSON format. Refer to "B.4 Confidentiality Privilege Manipulation Functions" for the privileges that can be granted.

```
select pgx_grant_confidential_privilege('matrix_foo', 'level1', 'group1',  '{"schema":["ALL"],
"table":["SELECT","INSERT","DELETE"]}')
```

If a confidentiality object of the target type is registered in the confidentiality level, when this function is executed, the confidentiality management feature internally uses the GRANT statement to grant a confidentiality group role access to confidentiality objects that are registered at a confidentiality level. The effect of the GRANT statement can be checked as follows.

```
select pgx_get_privileges_on_level_and_group('matrix_foo', 'level1', '["role1","role2"]')
```

For the format of the table returned by a function that outputs authority information, such as this function, refer to "B.7 Functions that Support Definition Referencing and Comparison with System Catalogs".

7.3.6 Adding Confidentiality Objects to Confidentiality Level

Classify the intended database object into the appropriate confidentiality level according to the confidentiality level design. In the example below, 'table1' of 'schema1' and 'table2' of 'schema1' are added to 'level1' at the same time. Specify the confidentiality object type and confidentiality object name in the third argument in JSON format. If the type is same, you can enumerate multiple confidentiality objects. You can enumerate multiple objects you want to register for each sensitive object type. In this example, only the *table* type is registered, but *schema* type can also be registered at the same time.

```
select pgx_add_object_to_confidential_level ('matrix_foo', 'level1',
'[{
    "type":"table",
    "object":[
        {
          "schema":"schema1",
          "table":["table1","table2"]
        }
     ]
}]')
```

When specifying a rowset, you must declare what kind of set of rows it is in the value of the *rowset_expression* key. For details on how to specify it, refer to "B.5 Confidentiality Object Manipulation Functions".

When this function is executed, the confidentiality management feature internally uses the GRANT statement or CREATE POLICY statement to grant privileges to the specified confidentiality objects to all confidentiality groups set in the confidentiality matrix. The effect of the GRANT statement can be checked as follows. This example checks the privileges granted on 'table1' in 'schema1' and 'table2' in 'schema1'.

```
select pgx_get_privileges_on_object('matrix_foo',
'[{
    "type":"table",
    "object":[
      "schema":"schema1",
      "table":["table1","table2"]
    ]
}]')
```

7.3.7 Adding Roles to Confidentiality Groups

Classify the intended role into the appropriate confidentiality group according to the confidentiality group design. You can add multiple roles at once. In the example below, role role1 and role role2 are added to confidentiality group 'group1'.

```
select pgx_add_role_to_confidential_group('matrix_foo', 'group1', '["role1","role2"]')
```

The confidentiality management feature adds the specified role to the members of the confidentiality group role.

The feature does not execute GRANT statements for confidentiality objects at this time. This is because we have already executed the GRANT statement between the confidentiality group role and the confidentiality object, and we will not execute the GRANT statement between the individual roles and the confidentiality object.

By using the function below, you can check the privileges that can be exercised by the specified role after inheriting the privilege of the confidentiality group role with the INHERIT attribute on the role side or changing to the confidentiality group role with the SET ROLE statement. In other words, you can check the access privileges that the actually registered roles can exercise. For details on the output format, refer to "B.7 Functions that Support Definition Referencing and Comparison with System Catalogs". The example below checks the privileges of role 'role1' and role 'role2'.

```
select pgx_get_privileges_on_role('matrix_foo',  '["role1","role2"]')
```

7.4 How to Use Confidentiality Management Feature (Change and Deletion)

7.4.1 Renaming Confidentiality Objects

The current confidentiality management feature cannot follow when confidentiality objects are renamed. Therefore, if the name is changed, remove the old name confidentiality object with the *pgx_remove_object_from_confidential_level* function specifying the old name, and register the confidentiality object with the new name using the *pgx_add_object_to_confidential_level* function specifying the new name.

7.4.2 Renaming Roles

The current confidentiality management feature cannot follow when roles are renamed. Therefore, when renaming, exclude the role with the old name with the *pgx_remove_role_from_confidential_group* function specifying the old name, and after renaming the role, register the role with the new name with the *pgx_add_role_to_confidential_group* function specifying the new name.

7.4.3 Deleting Roles

The current confidentiality management feature cannot follow when roles are dropped. Therefore, when dropping, exclude the role with *pgx_remove_role_from_confidential_group* function.

7.4.4 Changing Confidentiality Matrix

Change the name or attributes of the confidentiality matrix as follows. Specify the name and attribute value to be changed in JSON format. You can change the name and multiple attributes at once. Attributes not specified here are not changed.

```
select pgx_alter_confidential_matrix('matrix_foo', '{"name": "matrix_bar", "comment":"This matrix is
defined for bar."}')
```

7.4.5 Deleting Confidentiality Matrix

Delete the confidentiality matrix as follows.

```
select pgx_drop_confidential_matrix('matrix_foo', false, false)
```

By specifying a second or third argument, you can choose to delete all confidentiality levels and confidentiality groups added to the confidentiality matrix, or remove confidentiality group roles. Confidentiality group roles that were not deleted can be identified using the previously mentioned naming convention. No matter what you choose, the roles registered in the confidentiality group will not be deleted. The above example simply deletes the confidentiality matrix only.

7.4.6 Changing Confidentiality Level

Change the name and attributes of the confidentiality levels that were added to the confidentiality matrix 'matrix_foo' as follows. Specify the value after changing the name and attributes in JSON format. You can change the name or multiple attributes at once. Attributes not specified here are not changed.

```
select pgx_alter_confidential_level('matrix_foo', 'level1', '{"name": "levelX", "comment":"This
level required the highest confidential clearance."}')
```

If a confidentiality object was already registered with the confidentiality level, the attributes of the confidentiality object are also changed automatically. However, as mentioned in "7.2.1.1 Defining Confidentiality Levels", it may not be possible to automatically change the attributes of confidentiality objects. At that time, the function checks that the attributes of the registered confidentiality object are stricter than the attribute of the modified confidentiality level. If the check fails, this function will also fail. For example, if you try to change the encryption_algorithm from AES128 to AES256 and an AES128 encrypted table is registered as a confidentiality object, this function will fail. In such a case, please execute this function again after encrypting the table with AES256.

7.4.7 Deleting Confidentiality Level

Delete the confidentiality level as follows.

```
select pgx_drop_confidential_level('matrix_foo', 'level1', false)
```

With a third argument, you can choose to delete those that depend on the specified confidentiality level. Of course, confidentiality objects registered with the confidentiality level are not deleted.

7.4.8 Changing Confidentiality Group

Change the name and attributes of the confidentiality groups that were added to the confidentiality matrix 'matrix_foo' as follows. Specify the value after changing the name and attributes in JSON format. You can change the name or multiple attributes at once. Attributes not specified here are not changed.

```
select pgx_alter_confidential_group('matrix_foo', 'group1', '{"name": "groupX", "comment":"Members
of this group have the highest confidential clearance."}')
```

7.4.9 Deleting Confidentiality Group

Delete the confidentiality group as follows.

```
select pgx_drop_confidential_group('matrix_foo', 'group1', true, true)
```

By specifying a second or third argument, you can choose to delete dependencies on the specified confidentiality group or delete the confidentiality group role. Any choice will not delete the roles belonging to the confidentiality group.

Even if you leave the confidentiality group role, this function will revoke privileges from the confidentiality group role. The privilege to revoke is the privilege defined in confidentiality privileges. Therefore, note that a role registered in a confidentiality group will be deprived of various privileges granted by inheriting the confidentiality group role.

7.4.10 Revoking Confidentiality Privileges

Revoke confidentiality privileges from confidentiality groups as follows. Specify the confidentiality object and confidentiality group and the privileges to revoke in JSON format. The following example revokes INSERT and DELETE privileges for table-type confidentiality objects belonging to 'level1' from 'group1'.

```
select pgx_revoke_confidential_privilege('matrix_foo','level1', 'group1',  '{"table":
["INSERT","DELETE"]}')
```

7.4.11 Removing Confidentiality Objects from Confidentiality Level

Remove confidentiality objects from the confidentiality level as follows. Specify the type and name of confidentiality objects to be excluded in JSON format. The following example removes table-type confidentiality objects schema1.table1 and schema1.table2 from the confidentiality level 'level1'.

```
select pgx_remove_object_from_confidential_level('matrix_foo', 'level1'
'[{
    "type":"table",
    "object":[
        {
          "schema":"schema1",
          "table":["table1","table2"]
        }
     ]
}]')
```

7.4.12 Removing Roles from Confidentiality Groups

Remove the role from the confidentiality group as follows. Specify the roles to remove in JSON format. The following example removes roles 'role1' and 'role2' from confidentiality group 'group1'.

```
select pgx_remove_role_from_confidential_group('matrix_foo', 'group1', '["role1","role2"]')
```

7.5 Suggestions for Monitoring Methods

Confidentiality management role must ensure that the database is operating securely as intended, even after confidentiality managements are defined. If you are only using the confidentiality management feature, you do not have to worry about such things. However, the confidentiality management feature does not prohibit changing the definitions of tables and roles without using this feature.

So you have to detect when such an action has taken place.

However, even if they detect it, they may forget to deal with it. Therefore, it is necessary to periodically check the difference between the confidentiality level and confidentiality group and the actual definition of confidentiality objects and roles. Of course, even if there was a mismatch, it would not be a problem if the confidentiality object or role had stricter attributes and privilege.

The procedure presented here aims at matching.

7.5.1 How to Detect Privilege Changes without Using Confidentiality Management feature

Use the audit log to detect unauthorized modification of the attributes of confidentiality objects or roles, or modification of privileges without going through the confidentiality management feature.

The basic method for detection is to detect actions by roles other than the confidentiality management role. However, there are exceptions such as:

When performing an operation without using the confidentiality management feature for a legitimate reason

For example, when changing the authority of a function that the confidentiality management feature does not treat as a confidentiality object. In order to identify this, it is recommended to determine roles that perform changes that do not involve the confidentiality management feature. This is because when various roles do this, it becomes difficult to detect audit logs that deviate from operational rules.

When monitoring the activity of the confidentiality management role

For example, it would be a good idea to create rules that allow access only at specified times and from specified terminals, and to detect activities that violate those rules from audit logs. It is important to set rules so that violations cannot be covered up. For example application_name is not suitable as it can be easily spoofed.

7.5.2 How to Check Confidentiality Objects and Roles

This section describes how to confirm that the contents registered in the confidentiality matrix match the confidentiality objects and roles managed in the PostgreSQL system catalog. The method shown here is just an example.

Inspector

The inspector must be granted SELECT privilege on tables with the pg_confidential_management_support extension and SELECT privilege on the confidentiality object and role information added to the subject confidentiality matrix in the system tables. At a minimum, the confidentiality management role for the confidentiality matrix being reviewed and superuser certainly have such privileges.

Procedure

The confirmation procedure is divided into the following three phases. If you detect a discrepancy, use the audit log to investigate what caused the discrepancy. And fix it to match.

- Various objects may be changed (modified) in a chain reaction when returning to the correct state. Doing so may erase traces of unauthorized manipulation. Therefore, it is recommended that you investigate any design inconsistencies and ensure that you have the necessary audit logs for the investigation before fixing them.
- The *pgx_get_privileges_on_matrix* function presented here can output a very large table if the number of confidentiality objects or roles is large. If the size of this table exceeds the value of PostgreSQL's work_mem parameter, I/O will occur according to PostgreSQL's specifications, resulting in a slowdown. Therefore, it is recommended that work_mem be set as high as possible in the session in which this function is executed.

(1) Make sure the state of the confidentiality matrix is the same as the design

If you detect a mismatch, please restore the correct state by executing functions such as *pgx_alter_confidential_level* function provided by this extension.

- Attributes of the confidentiality matrix

  Check using the *pgx_confidential_matrix* table.

- Number of registered confidentiality levels and attributes of each confidentiality level

  Check the row that matches the *clevmatid* to be checked from the *pgx_confidential_level* table. To know the confidentiality matrix identifier, see *cmatid* in the *pgx_confidential_matrix* table.

- Number of registered confidentiality groups and attributes of each confidentiality group

  Check the row in the *pgx_confidential_role* table whose *ccolmatid* matches the identifier of the confidentiality matrix to be checked.

- Confidentiality group privileges set to confidentiality level

  Refer to the *pgx_confidential_privileges* table. However, in this table confidentiality levels and confidentiality groups are represented as identifiers. Join with *pgx_confidential_level table* and *pgx_confidential_role table* if you want to check by confidentiality level name or confidentiality group name.

(2) Verify correct confidentiality objects and roles registered

Confirm that the definition of the confidentiality management feature matches the definition of the confidentiality objects and roles. If you detect a mismatch, please restore the correct state by executing functions such as *pgx_alter_confidential_level* function provided by this extension.

- Is the number of confidentiality objects registered in the confidentiality level the same as designed?
- Do the attributes of each confidentiality object match the attributes of the confidentiality level to which it belongs?

  It is recommended to check these using *pgx_get_attribute_of_objects* function. Because this function gets the state of the confidentiality object from the PostgreSQL system catalog and outputs a table that can be compared with the definition. For the format of the table, refer to "B.7 Functions that Support Definition Referencing and Comparison with System Catalogs".

- Is the number of roles registered in the confidentiality group the same as designed?
- Do the attributes of each role match the attributes of the confidentiality group to which it belongs?

  It is recommended to check these using *pgx_get_attribute_of_roles* function.For the format of the table returned by a function that outputs authority information, such as this function, refer to "B.7 Functions that Support Definition Referencing and Comparison with System Catalogs".

(3) Check privilege of confidentiality objects

Verify that the roles granted access to confidentiality objects and what the privileges are are consistent with the definitions in the confidentiality matrix. A good help is to use the *pgx_get_privileges_on_matrix* function. For details on the output format, see "7.3.5 Granting Confidentiality Privileges to Confidentiality Groups".

If you want to focus on any confidentiality level or role, use the function below.

- pgx_get_privileges_on_level_and_group()
- pgx_get_privileges_on_object()
- pgx_get_privileges_on_role()

Point

It is inefficient to output a large table many times in order to analyze it with various SQL statements. You can perform efficient analysis by specifying a query that executes this function in the INSERT statement as shown below.

```
INSERT INTO temp_table_for_analysis SELECT pgx_get_privileges_on_matrix('matrix_foo')
```

7.6 Backup/Restore

There are 4 methods below.

- Use pg_basebackup
- Use pg_dumpall to back up the entire database cluster as a script file
- Use pg_dump and pg_restore to back up and restore only some databases
- Use pg_dump's pg_restore to back up and restore only some tables and schemas

Of these, if you adopt the method of using pg_basebackup, you do not need to be particularly careful. The following special precautions should be taken when adopting other methods.

Use pg_dumpall to back up the entire database cluster as a script file

Note the following.

- Do not use the -O option of the pg_dumpall command. This feature does not work properly if the confidentiality object changes ownership.
- Do not use the -x option of the pg_dumpall command. Of course, this would change tightly controlled privileges.
- Execute the backup script file as a superuser. A backup script file contains the CREATE EXTENSION statement for this extension. This is because the CREATE EXTENSION statement must be executed as a superuser.

Backup and restore only some databases using pg_dump and pg_restore

Note the following.

- Use the -r option of the pg_dumpall command to back up and restore role information as well. Of course, because roles are essential to confidentiality management.
- Do not use the -O option of the pg_dump command. This feature will not work properly if the confidentiality object changes ownership.
- Do not use the -x option of the pg_dump command. Of course, this would change tightly controlled privileges.
- Restore by a superuser for CREATE EXTENSION.

Use pg_dump's pg_restore to backup and restore only some tables and schema

Note the following.

- Use the -e option of the pg_dump command to back up and restore this extension as well. This is because the tables contained in this extension store the information required for confidentiality management.
- Use the -r option of the pg_dumpall command to back up and restore role information as well. Of course, because roles are essential to confidentiality management.
- Do not use the -O option of the pg_dump command. This feature will not work properly if the confidentiality object changes ownership.
- Do not use the -x option of the pg_dump command. Of course, this would change tightly controlled privileges.
- Restore by a superuser for CREATE EXTENSION.

7.7 Removing Setup

If you are no longer using this feature and want to continue using your database as before, we recommend simply DROP EXTENSION. This is because if you delete definitions such as confidentiality matrices and confidentiality levels created using this extension using the deletion functions provided by this extension, the roles automatically created by this extension will be deleted, the privileges of roles granted by this extension are revoked. Simply dropping this extension will only drop the tables it contains, it will not do any of these things.

7.8 Usage Example of Confidentiality Management

Here is an example of the concept of confidentiality levels and confidentiality groups.

This section assumes a simple business that handles customer purchase information.

First, create a confidentiality matrix for confidentiality management of this information.

```
SELECT pgx_create_confidential_matrix('matrix_purchase_managenet' , 'Confidentiality management of
customer purchase information');
```

The data we process may also contain personally identifiable information. Access to such data should be restricted to those who have access to it to minimize the risk of information disclosure.

Customer purchase information, including personally identifiable information, is managed in the following table.

```
CREATE TABLE purchase.customer_info(     -- Customer information
    customer_id     integer,               -- Customer ID
    name            text,
    address         text,
    phone_number    char(12),
    rank            integer                -- Customer's service rank
);

CREATE TABLE purchase.history(           -- History of a customer's purchase of goods
    customer_id         integer,            -- ID of the customer who purchased
    purchase_date       date,               -- Date the goods were purchased
    item_code           char(12),           -- Code of the goods purchased
    purchase_number     integer,            -- Quantity purchased
    purchase_amount     integer             -- Amount of purchased goods
);
```

Among customer information, name, address, and telephone number are personal information because they can identify an individual when combined. In order to properly handle such information, we have made it so that it can only be handled by employees belonging to a specific group who have received appropriate training.

Therefore, we have prepared two confidentiality levels: "level_personal_info", which means highly confidential personal information, and "level_customer_info", which means other information.

```
SELECT pgx_create_confidential_level('matrix_purchasee_management', 'level_personal_info',
                                     NULL, 'Personally identifiable information');
SELECT pgx_create_confidential_level('matrix_purchase_management', 'level_customer_info',
                                     NULL, 'Non-personally identifiable information');
```

In addition, we will prepare two confidentiality groups: "group_qualified " who have been educated about handling personal information and can handle personal information appropriately, and "group_non_qulified" who are not qualified.

```
SELECT pgx_create_confidential_group ('matrix_purchase_management', 'group_qualified',
                                     NULL, 'Qualified staff handling personal information');
SELECT pgx_create_confidential_group ('matrix_purchase_management', 'group_non_qulified',
                                     NULL, 'General employee');
```

Let's take a closer look at the data we're dealing with.

Since the customer information table contains personal information, it corresponds to personal information. However, the customer_id and rank contained in the customer information table are not personal information because they are not personally identifiable information. In addition, since this customer_id and rank are also information necessary for business analysis, it is inconvenient that only those who are qualified to handle personal information can access such information.

Therefore, the customer information table uses columns for confidentiality management. The entire customer information table is protected as personal information, and the range of access is expanded by making the columns that are not personal information general customer information.

Follow this policy to set confidentiality level privilege for confidentiality group.

```
SELECT pgx_grant_confidential_privilege('matrix_purchase_management',
                                     'level_personal_info',
                                     'group_qualified', '{"table":["ALL"]}');
SELECT pgx_grant_confidential_privilege('matrix_purchase_management',
                                     'level_customer_info',
                                     'group_qualified', '{"table":["ALL"]}');
SELECT pgx_grant_confidential_privilege('matrix_purchase_management',
                                     'level_customer_info',
                                     'group_not_qualified',
                                     '{"table":["ALL"], "column":["SELECT"]}');
```

Only "qualified personnel" can handle "personal information". "Customer information" can be handled by both "qualified personnel" and "general employees". Some columns of tables that handle "personal information" are allowed to be referred to as "customer information".

This completes the authorization settings in the confidentiality matrix.

Next, we will register the database objects that handle purchase information in the confidentiality matrix.

```
SELECT pgx_add_object_to_confidential_level('matrix_purchase_management', 'level_personal_info',
                                    '[{
                                        "type":"table",
                                        "object":[{
                                                    "schema": "purchase",
                                                    "table": ["customer_info"]
                                         }]
                                     }]');
SELECT pgx_add_object_to_confidential_level('matrix_purchase_management', 'level_customer_info',
                                    '[{
                                        "type":"column",
                                        "object":[{
                                                     "schema": "purchase",
                                                     "table": "customer_info",
                                                     "column": ["customer_id", "rank"]
                                         }]
                                     }]');
SELECT pgx_add_object_to_confidential_level('matrix_purchase_management', 'level_customer_info',
                                    '[{
                                        "type":"table",
                                        "object":[{
                                                     "schema": "purchase",
                                                     "table": ["history"]
                                     }]
                                     }]');
```

The entire customer information table is "personal information", the customer_id column and rank column of the customer information table are "customer information", and the entire purchase history table is also "customer information".

Finally, enroll the employee in the confidentiality group. "Alex" and "Bola" are "qualified persons" who have received training in personal information management. Also, "Charlie" and "Dana" are "general employees" because they have not yet received training on personal information management.

```
SELECT pgx_add_role_to_confidential_group('matrix_purchase_management',
                                          'group_qualified',
                                          '["Alex", "Bola"]');
SELECT pgx_add_role_to_confidential_group('matrix_purchase_management',
                                          'group_non_qualified',
                                          '["Charlie", "Dana"]');
```

Appendix A Tables Used by Confidentiality Management Feature

This section describes the tables used by the confidentiality management feature.

A.1 pgx_confidential_matrix

A list of confidentiality matrices.

You can refer to the attributes of the registered confidentiality matrix, the update time, or the time when the confidentiality level or confidentiality group was registered or deleted.

Column name	Type	Constraint	Description
cmatid	bigint	primary key generated always as identity	Identifier of the confidentiality matrix.
cmatname	varchar(63)	unique not null	Name of the confidentiality matrix.
cmatowner	name	not null	Owner of the confidentiality matrix. The role that created the confidentiality matrix becomes the owner.
cmatcomment	text		Comment.
cmatupdatetime	timestamp with time zone	not null	Update time of the confidentiality matrix itself.
cmatoperationtime	timestamp with time zone		The time when the confidentiality level and confidentiality group were added/deleted.

A.2 pgx_confidential_level

A list of confidentiality levels.

You can refer to the registered confidentiality level attributes, update time, or the time when a confidentiality object was registered to the confidentiality level or removed from the confidentiality level.

Column name	Type	Constraint	Description
clevid	bigint	primary key generated always as identity	Identifier of the confidentiality level.
clevname	varchar(63)	not null	Name of the confidentiality level.
clevmatid	bigint	not null references pgx_confidential_matrix(cmatid)	Identifier of the confidentiality matrix to which the confidentiality level belongs.
clevcomment	text		Comment.
clevupdatetime	timestamp with time zone	not null	Update time of the confidentiality level itself.
clevoperationtime	timestamp with time zone		The time when the confidentiality object was added/deleted.
clevencalgorithm	text	not null	Encryption method. "none" for no encryption. AES128 and AES256 can be set.

A.3 pgx_confidential_group

A list of confidentiality groups.

You can refer to the registered confidentiality group attributes, update time, or the time when a role was registered to the confidentiality group or removed from the confidentiality group.

Column name	Type	Constraint	Description
cgroid	bigint	primary key generated always as identity	Identifier of the confidentiality group.
cgroname	varchar(63)	not null	Name of the confidentiality group.
cgromatid	bigint	not null references pgx_confidential_matrix(cmatid)	Identifier of the confidentiality matrix to which the confidentiality group belongs.
cgrocomment	text		Comment.
cgroupdatetime	timestamp with time zone	not null	Update time of the confidentiality group itself.
cgrooperationtime	timestamp with time zone		The time when the role was added/deleted.
cgrorolename	name	not null	Name of the confidentiality group role.
cgrosuperuser	bool	not null	true if the confidentiality group role has SUPERUSER privileges.
cgrocreatedb	bool	not null	true if the confidentiality group role has CREATEDB privileges.
cgrocreaterole	bool	not null	true if the confidentiality group role has CREATEROLE privileges.
cgroreplication	bool	not null	true if the confidentiality group role has REPLICATION privileges.
cgrobypassrls	bool	not null	true if the confidentiality group role has BYPASSRLS privileges.

A.4 pgx_confidential_privilege

A list of confidentiality privileges.

You can refer to confidentiality privilege set for each confidentiality object, update time, and so on.

Column name	Type	Constraint	Description
cpriid	bigint	primary key generated always as identity	Identifier of the privilege.
cprimatid	bigint	not null references pgx_confidential_matrix(cmatid)	Identifier of the confidentiality matrix to which the privilege belongs.
cprilevid	bigint	not null references pgx_confidential_level(clevid)	Identifier of the confidentiality level for which privilege is set.
cprigroid	bigint	not null references	Identifier of the confidentiality group for which privilege is set.

Column name	Type	Constraint	Description
		pgx_confidential_group(cgroid)	
cpritype	text	not null	Type of the confidentiality object which privilege is set.
cpriupdatetime	timestamp with time zone	not null	Update time when privilege was set/ changed.
cpriacl	text[]	not null	Access privileges that have been set. (*1)

*1: The character string indicating authority appears in the following order in the text type array of cpriacl.

ALL, SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, CREATE, CONNECT, TEMPORARY, EXECUTE, USAGE

If the appropriate privilege is not set, the string simply does not appear. For example, {'INSERT','TRUNCATE'} if you only have INSERT and TRUNCATE privileges.

A.5 pgx_confidential_object

A list of confidentiality objects.

You can refer to object attributes or update time, and so on.

Column name	Type	Constraint	Description
cobjid	bigint	primary key generated always as identity	Identifier of the confidentiality object.
cobjmatid	bigint	not null references pgx_confidential_matrix(cmatid)	Identifier of the confidentiality matrix to which the confidentiality object belongs.
cobjlevid	bigint	not null references pgx_confidential_level(clevid)	Identifier of the confidentiality level to which the confidentiality object belongs.
cobjtype	text	not null	The type of confidentiality object.
cobjschema	name	not null	Schema name of the confidentiality object.
cobjtable	name	not null	Table name of the confidentiality object.
cobjname	text	not null	Name of the confidentiality object.
cobjupdate	timestamp with time zone	not null	Registration time of the confidentiality object.
cobjpolicy	jsonb		Conditions that determine the range of rowsets when the type is rowset. It is expressed by the setting contents in POLICY.

A.6 pgx_confidential_role

A list of roles registered in the confidentiality group.

You can refer to role attributes or update time, and so on.

Column name	Type	Constraint	Description
crolid	bigint	primary key generated always as identity	Identifier of the role.
crolmatid	bigint	not null references pgx_confidential_matrix(cmatid)	Identifier of the confidentiality matrix to which the role belongs.
crolgroid	bigint	not null references pgx_confidential_group(cgroid)	Identifier of the confidentiality group to which the role belongs.
crolname	name	not null	Name of the role.
crolupdate	timestamp with time zone	not null	Registration time of the role.

A.7 pgx_confidential_policy

This is a list of policies created to set privileges for confidentiality objects of rowset type. You can refer to the name of the policy you created and the privileges it has set.

Rows in this table are inserted when you add a rowset type confidentiality object.

Column name	Type	Constraint	Description
cpolid	bigint	primary key generated always as identity	Identifier of the policy.
cpolmatid	bigint	not null references pgx_confidential_matrix(cmatid)	Identifier of the confidentiality matrix to which the policy belongs.
cpollevid	bigint	not null references pgx_confidential_level(clevid)	Identifier of the confidentiality level to which the policy belongs.
cpolgroid	bigint	not null references pgx_confidential_group(cgroid)	Identifier of the confidentiality group to which the policy belongs.
cpolobjid	bigint	not null references pgx_confidential_object(cobjid)	Identifier of the rowset object using this policy.
cpolprivilege	text	not null	Privilege this policy has (SELECT, INSERT, UPDATE, DELETE, ALL).
cpolname	name	not null	Name of the policy.
cpolexpression	jsonb	not null	Expression of the policy.

Appendix B System Management Functions Used by Confidentiality Management Feature

This section describes the system management functions used by the confidentiality management feature. All functions abort the transaction on failure.

Note

- Be careful when performing operations that involve deleting confidentiality groups.

  If you remove a confidentiality group along with a confidentiality group role, you simply no longer have the role that can access the confidentiality object. However, when you leave the confidentiality group role, the function revokes privileges from the confidentiality group role. The privilege to revoke is the privilege defined in confidentiality privileges.

- The *pgx_get_privileges_on_matrix* function may output a very large table if the number of confidentiality objects or roles is large. If the size of this table exceeds the value of PostgreSQL's *work_mem* parameter, I/O will occur according to PostgreSQL's specifications and will be slow. To prevent this, it is recommended that *work_mem* be set as high as possible in the session in which this function is executed.

B.1 Confidentiality Matrix Manipulation Functions

Function name	Return value	Description
pgx_create_confidential_matrix(confidential_matrix_name varchar, comment text)	void	Create the confidentiality matrix with the specified name. The created confidentiality matrix is registered in the *pgx_confidential_matrix* table along with the comment. Only roles with the required entitlements for the confidentiality management role can execute this function. The role that executes this function is the confidentiality management role for the confidentiality matrix. Functions such as executing by specifying a confidentiality matrix name require that the executed role is a confidentiality management role for the specified confidentiality matrix. For details, please refer to "7.2.2 Determining Confidentiality Management Roles". The length of *confidential_matrix_name* must be less than 64 characters. Note that the units are not bytes. There are no restrictions on the characters that can be used in the *confidential_matrix_name*. When you specify the name of the confidentiality matrix to other functions, you must specify the same string that you specified to this function. Note that unlike most CREATE statements, the name of the confidentiality matrix is case sensitive.
pgx_copy_confidential_matrix(confidential_matrix_name varchar, source_confidential_matrix_name varchar)	void	Copy the source confidentiality matrix specified by *source_confidential_matrix_name* to the confidentiality matrix named *confidential_matrix_name*. Confidentiality matrix, confidentiality levels, confidentiality groups, and confidentiality privileges details are replicated. However, the information of confidentiality objects registered in the confidentiality level and roles registered in the confidentiality group are not duplicated. Any confidentiality management role in any confidentiality matrix can execute this function.

Function name	Return value	Description
		The owner of the cloned confidentiality matrix is the role that executed this function. Restrictions on *confidential_matrix_name* as strings are the same as for the *pgx_create_confidential_matrix* function. Comments are also duplicated.

pgx_alter_confidential_matrix(confidential_matrix_name varchar, alter_object json)	void	Change the attributes of the confidentiality matrix named by *confidential_matrix_name*.
		
		Only confidentiality management role for the specified confidentiality matrix can execute this function.
		
		For *alter_object*, specify the attribute you want to change and the value after change in key-value format as follows. Attributes not specified remain unchanged.
		
		`'{`
		`  "name":"matrix_foo",`
		`  "comment":"This matrix is defined for foo.")`
		`}'`
		
		name: Specify the name of the modified confidentiality matrix. Cannot be *null*. The function will fail if you specify the name of a confidentiality matrix that already exists.
		
		comment: Specify a comment after the change. Can be *null*.

pgx_drop_confidential_matrix(confidential_matrix_name varchar, cascade bool, drop_role bool)	void	Drop the confidentiality matrix with the specified name. Only confidentiality management role for the specified confidentiality matrix can execute this function. Specify *true* for cascade to recursively check and remove objects that depend on this confidentiality matrix. For example, delete the confidentiality groups and confidentiality levels registered in this confidentiality matrix. Then remove the confidentiality privileges associated with that confidentiality level. To drop a confidentiality level, execute internally *pgx_drop_confidential_level* function with a *cascade* value. When dropping a confidentiality group, execute internally *pgx_drop_confidential_group* function with *cascade* and *drop_role* values. See also the descriptions of these functions. In particular, how the privilege of confidentiality objects are changed is important. Specify *false* for cascade simply removes the confidentiality matrix. The function will fail if there are objects that depend on this confidentiality matrix. If *true* is specified for *drop_role*, the confidentiality group role registered in this confidentiality matrix will be deleted. Naturally, it only makes sense when *cascade* is *true*.

B.2 Confidentiality Level Manipulation Functions

Function name	Return value	Description
pgx_create_confidential_level(confidential_matrix_name varchar, confidential_level_name varchar, options json, comment text)	void	Creates a confidentiality level, registers it with the specified confidentiality matrix, and adds it to the *pgx_confidential_level* table with the specified comment and attributes specified in *options*.

Function name	Return value	Description

		Only confidentiality management role for the specified confidentiality matrix can execute this function.
		
		The length of *confidential_level_name* must be less than 64 characters.
		
		Note that the units are not bytes.
		
		There are no restrictions on the characters that can be used in the *confidential_level_name*.
		
		When specifying a confidentiality level name for any other function, you must specify the same string as specified for this function
		
		Note that unlike most CREATE statements, confidentiality level names are case-sensitive.
		
		Specify a comment for *comment*.
		
		For *options*, specify the attribute of the confidentiality level as follows. If you specify NULL, the default value for each attribute will be set.
		
		`'{`
		`  "encryption_algorithm":"AES256"`
		`}'`
		
		encryption_algorithm: Specify the encryption algorithm. The algorithms and default values that can be specified are the same as the *tablespace_encryption_algorithm* parameter of Transparent Data Encryption of Fujitsu Enterprise Postgres. Must not be *null*.

pgx_alter_confidential_level(confidential_matrix_name varchar, confidential_level_name varchar, alter_object json)	void	Change the confidentiality level attribute.
		
		Only confidentiality management role for the specified confidentiality matrix can execute this function.
		
		For *alter_object*, specify the attribute you want to change and the value after change in key-value format as follows.
		
		`'{`
		`  "name":"level_new",`
		`  "comment":"This level is the highest confidentiality level.",`
		`  "encryption_algorithm":"AES256"`
		`}'`
		
		name: Specify the name of the modified confidentiality level. Cannot be null.
		
		comment: Specify a comment after the change. Can be null.
		
		Other attributes are the same as options of *pgx_create_confidential_level* function. Attributes not specified are not changed.
		
		Be careful when increasing the degree of confidentiality. For example, if you increase the encryption strength and there are confidentiality objects with a lower strength than the new strength, this function will fail.

pgx_drop_confidential_level(confidential_matrix_name varchar, confidential_level_name varchar, cascade bool)	void	Remove a confidentiality level from the confidentiality matrix and delete a confidentiality level.

Function name	Return value	Description
		Only confidentiality management role for the specified confidentiality matrix can execute this function. If *true* is specified for *cascade*, the confidentiality level can be deleted even if confidentiality objects are registered in this confidentiality level. If *false* is specified for *cascade*, it is not possible to delete a confidentiality level that has confidentiality objects registered.

B.3 Confidentiality Group Manipulation Functions

Function name	Return value	Description

pgx_create_confidential_group(confidential_matrix_name varchar, confidential_group_name varchar, options json, comment text)	void	Create the confidentiality group, registers it with the specified confidentiality matrix, and adds it to the *pgx_confidential_group* table with the specified comment and attributes specified in *options*.
		
		Only confidentiality management role for the specified confidentiality matrix can execute this function. However, setting some attributes requires superuser privileges, as described below. Therefore, by necessity, the confidentiality management role must be superuser if those attributes are to be set.
		
		This function internally uses the CREATE ROLE statement to create a confidentiality group role.
		
		The length of *confidential_group_name* must be less than 64 characters. Note that the units are not bytes.
		
		There are no restrictions on the characters that can be used in the *confidential_group_name*.
		
		When you specify the name of the confidentiality group to other functions, you must specify the same string that you specified to this function.
		
		Note that unlike most CREATE statements, confidentiality group names are case-sensitive.
		
		For *options*, specify the attributes of the confidentiality group as follows. If you specify NULL, the default value for each attribute will be set.
		
		`'{`
		`  "SUPERUSER":false,`
		`  "CREATEDB":true,`
		`  "CREATEROLE":false,`
		`  "REPLICATION":false,`
		`  "BYPASSRLS":false`
		`}'`
		
		The only attributes that can be specified are the SUPERUSER, CREATEDB, CREATEROLE, REPLICATION, and BYPASSRLS attributes that relate to access privileges to data.
		
		These attributes are some of the role attributes that can be specified in the CREATE ROLE statement. The attribute semantics and default values are the same as in the CREATE ROLE statement specification.

Function name	Return value	Description
		As noted in the CREATE ROLE statement description, this function fails if a non-superuser specifies *true* for the SUPERUSER, REPLICATION, and BYPASSRLS attributes.

pgx_alter_confidential_group(confidential_matrix_name varchar, confidential_group_name varchar, alter_object json);	void	Change the attributes of a confidentiality group.
		
		Only confidentiality management role for the specified confidentiality matrix can execute this function. As described in *pgx_create_confidential_group* function, you must be superuser to set some attributes.
		
		This function internally uses the ALTER ROLE statement to change the attributes of the confidentiality group role.
		
		For *alter_object*, specify the attribute you want to change and the value after change in key-value format as follows.
		
		`'{`
		`  "name":"group_new",`
		`  "comment":"Members of this group have the highest confidential clearance.",`
		`  "CREATEDB":false`
		`}'`
		
		name: Specify the name of the confidentiality group after modification. Cannot be null.
		
		comment: Specify a comment after the change. Can be null.
		
		Other attributes are the same as options of *pgx_create_confidential_group* function. Attributes not specified are not changed.
		
		For example, if you change the attribute to a weaker one, such as changing CREATEDB to *false*, the attributes of roles registered in the confidentiality group will be similarly weakened.

pgx_drop_confidential_group(confidential_matrix_name varchar, confidential_group_name varchar, cascade bool, drop_role bool)	void	Drop the confidentiality group from the confidentiality matrix and delete a confidentiality group. Only confidentiality management role for the specified confidentiality matrix can execute this function. If *true* is specified for cascade, the confidentiality group can be deleted even if roles are registered in this confidentiality group. If *false* is specified for *cascade*, confidentiality groups that have roles registered cannot be deleted. Specify true for *drop_role* to drop the confidentiality group role. Roles registered in confidentiality groups remain. If false is specified for *drop_role*, the confidentiality group role will not be deleted. When you leave the confidentiality group role, the function revokes privileges from the confidentiality group role. The privilege to revoke is the privilege defined in confidentiality privileges.

B.4 Confidentiality Privilege Manipulation Functions

Function name	Return value	Description
pgx_grant_confidential_privilege(confidential_matrix_name varchar,	void	Grant confidentiality privileges.

Function name	Return value	Description

confidential_level_name varchar, confidential_group_name varchar, privilege json)		Only confidentiality management role for the specified confidentiality matrix can execute this function.
		
		Grants access to the confidentiality level specified by *confidential_level_name* to the confidentiality group specified by *confidential_group_name*.
		
		When you run this function repeatedly, it simply adds more privileges to grant. Granting the same privilege more than once does not result in an error.
		
		The privilege to be granted is specified in *privilege*. *privilege* specifies the type of the confidentiality object as the key and an array of privileges as the value, like this:
		
		`'{`
		`  "table":["SELECT", "INSERT", "UPDATE", "DELETE"],`
		`  "schema":["CREATE", "USAGE"],`
		`  "rowset":["ALL"]`
		`}'`
		
		The privileges that can be specified depend on the type of confidentiality object.
		
		Privilege for rowset type confidentiality object is privilege that can be specified in the FOR clause of the CREATE POLICY statement.
		
		Except for the rowset type, it is a privilege that can be granted with the GRANT statement according to the confidentiality object type.
		
		If ALL is specified, it is assumed that all privileges that can be specified for that type are listed.
		
		That is, ALL does not appear in the *cpriacl* column of the *pgx_confidential_privilege* table.
		
		The same specification as the WITH GRANT OPTION clause of the GRANT statement cannot be specified. This is because only confidentiality management roles should use this feature to change privilege to confidentiality objects.
		
		Be carefull when PUBLIC is granted to target confidentiality object. This is because granting privileges to PUBLIC is the same as granting privileges to all roles registered in the confidentiality matrix. This function will fail if a privilege granted indirectly to each role using PUBLIC is defined in the confidentiality privileges that should not be granted to that role. Similarly, this function also checks privileges granted indirectly through group roles that are not registered in the confidentiality matrix. In doing so, it recursively checks the chain of inheritance.

pgx_revoke_confidential_privilege(confidential_matrix_name varchar, confidential_level_name varchar, confidential_group_name varchar, privilege json)	void	Revoke confidentiality privileges. Only confidentiality management role for the specified confidentiality matrix can execute this function. Revokes privilege to the confidentiality level specified by *confidential_level_name* from the confidentiality group specified by *confidential_group_name*. Revoking ungranted privileges does not fail. Privilege to be revoked is specified in *privilege*. The specification method is the same as *pgx_grant_confidential_privilege* function.

Function name	Return value	Description
		Be carefull when PUBLIC is granted to target confidentiality object. This is because granting privileges to PUBLIC is the same as granting privileges to all roles registered in the confidentiality matrix. This function will fail if a privilege granted indirectly to each role using PUBLIC is defined in the confidentiality privileges that should not be granted to that role. Similarly, this function also checks privileges granted indirectly through group roles that are not registered in the confidentiality matrix. This time, it checks the chain of inheritance up to its ancestors.

B.5 Confidentiality Object Manipulation Functions

Function name	Return value	Description

pgx_add_object_to_confidential_level(confidential_matrix_name varchar, confidential_level_name varchar, object_name json)	void	Only confidentiality management role for the specified confidentiality matrix can execute this function.
		
		Adds the confidentiality object specified by *object_name* to the confidentiality level specified by *confidential_level_name*.
		
		This function internally uses the GRANT statement to grant privileges to the confidentiality group role according to the confidentiality privileges associated with this confidentiality level.
		
		However, if the confidentiality object is of type rowset, it internally uses the CREATE POLICY statement to grant privileges to the confidentiality group role according to the confidentiality privileges associated with this confidentiality level. Also, to enable POLICY, execute the ALTER TABLE statement on the target table with the ENABLE ROW LEVEL SECURITY clause.
		
		Currently it is not possible to register a foreign table as a table type confidentiality object.
		
		Be carefull when PUBLIC is granted to target confidentiality object. This is because granting privileges to PUBLIC is the same as granting privileges to all roles registered in the confidentiality matrix. This function will fail if a privilege granted indirectly to each role using PUBLIC is defined in the confidentiality privileges that should not be granted to that role. Similarly, this function also checks privileges granted indirectly through group roles that are not registered in the confidentiality matrix. In doing so, it recursively checks the chain of inheritance.
		
		For rowset type confidentiality objects, this function will fail if the target table has a POLICY defined that was not created using this feature, regardless of what privileges are granted.
		
		These checks only apply to the confidentiality group role or to roles that are registered with the confidentiality group. If a POLICY exists that targets a role that is not, the function will not fail.
		
		Specify object_name as follows: Only the rowset is slightly different. The example below attempts to register multiple types of objects in one go.
		
		`'[{`
		`  "type":"schema",`
		`  "object":[`
		`  {"schema":"schema1"},`
		`  {"schema":"schema2"}`

Function name	Return value	Description

		]
		},
		
		{
		"type":"table",
		"object":[
		{
		"schema":"schema1",
		"table":["table1","table2"]
		},
		{
		"schema":"schema2",
		"table":["table8","table9"]
		}
		]
		},
		
		{
		"type":"column",
		"object":[
		{
		"schema":"schema1",
		"table":"table1",
		"column":["column1", "column2"]
		},
		{
		"schema":"schema1",
		"table":"table2",
		"column":["column8", "column9"]
		}
		]
		}]'
		
		For the rowset type, you define rowset and give it a name, as in the example below. This name is used by the *pgx_remove_object_from_confidential_level* function to identify the rowset type confidentiality object when removing it.
		
		This example shows:
		
		- In schema1.table1, represents a set of rows (rowset) for which the conditional expression (user = current_user OR manger = current_user) is true.
		
		- If manager=current_user is true, the col1 value of the record to be INSERTed or the record after UPDATE must be greater than zero.
		
		'[{
		"type":"rowset",
		"object":[
		{
		"schema":"schema1",
		"table":"table1",
		"rowset_name":"rowset1",
		"rowset_expression": [
		{
		"as":"permissive",
		"using":"user = current_user"
		},
		{
		"as":"permissive",

Function name	Return value	Description
		"using":"manager = current_user", "with check":"col1 > 0" }] }] }]' Each key (as, *using*, *with check*) has the same meaning as the clause of the same name in the CREATE POLICY statement. As you can see from this, one element of the array specified in *rowset_expression* corresponds to one POLICY object created by the CREATE POLICY statement. In fact, this function internally executes as many CREATE POLICY statements as there are elements in the array. The name of POLICY at this time is 'pgx_cms_policy_${cpolid}'. ${cpolid} is automatically numbered by this extension. <Note> Do not create policies with names that begin with *pgx_cms_policy_*. Because this function may fail.
pgx_remove_object_from_confidential_level(confidential_matrix_name varchar, confidential_level_name varchar, object_name json)	void	Removes confidentiality objects from the confidentiality level. Only confidentiality management role for the specified confidentiality matrix can execute this function. Removes the confidentiality object specified by *object_name* from the confidentiality level specified by *confidential_level_name*. The format of *object_name* is the same as *object_name* in the *pgx_add_object_to_confidential_level* function. But the *rowset_expression* key is optional. If specified, it is simply ignored. Therefore, *object_name* in *pgx_add_object_to_confidential_level* can be specified to this function without modifying it. At this time, any privileges granted to the confidentiality group based on confidentiality privileges are revoked. If the confidentiality object is of rowset type, delete the internally created policy. At this time, if there are zero policies associated with the table, the ALTER TABLE statement with the DISABLE ROW SECURITY clause is internally executed to disable row level security.

B.6 Role Manipulation Functions

Function name	Return value	Description
pgx_add_role_to_confidential_group(confidential_matrix_name varchar, confidential_group_name varchar, role_name json)	void	Add a role to a confidentiality group. Only confidentiality management role for the specified confidentiality matrix can execute this function. Adds the role specified by *role_name* to the confidentiality group specified by *confidential_group_name*. If the role to be added has been granted broader privileges than the confidentiality privileges, revoke the privileges according to the confidentiality privileges.

Function name	Return value	Description
		Be carefull when PUBLIC is granted to target confidentiality object. This is because granting privileges to PUBLIC is the same as granting privileges to all roles registered in the confidentiality matrix. This function will fail if a privilege granted indirectly to each role using PUBLIC is defined in the confidentiality privileges that should not be granted to that role. Similarly, this function also checks privileges granted indirectly through group roles that are not registered in the confidentiality matrix. In doing so, it recursively checks the chain of inheritance. Also, if the added role has stronger attributes than the confidentiality group, change the attributes to match the confidentiality group. This function will fail if a strong attribute is indirectly assigned using a group role that is not registered in the confidentiality matrix. *role_name* is specified as follows. '{["role1","role"]}'
pgx_remove_role_from_confidential_group(confidential_matrix_name varchar, confidential_group_name varchar, role_name json)	void	Remove a role from a confidentiality group. Only confidentiality management role for the specified confidentiality matrix can execute this function. Removes the role specified by *role_name* from the confidentiality group specified by *confidential_group_name*. This function internally executes a REVOKE statement to remove the role from the confidentiality group role. It does not change the attributes of the role being removed, nor the privileges granted to that role. Simply banish it from the group so that it cannot inherit privileges. The method of specifying role_name is the same as *pgx_add_role_to_confidential_group* function.

B.7 Functions that Support Definition Referencing and Comparison with System Catalogs

Functions that Support Definition Referencing and Comparison with System Catalogs

Function name	Return value	Description
pgx_get_attribute_of_objects(confidential_matrix_name varchar)	setof record	Returns a table of attributes defined in the confidentiality matrix and attributes actually set in the database for all confidentiality objects registered in the specified confidentiality matrix. Refer to "Tables returned by pgx_get_attribute_of_objects" for the table format. Only confidentiality management role for the specified confidentiality matrix can execute this function.
pgx_get_attribute_of_roles(confidential_matrix_name varchar)	setof record	Returns a table of attributes defined in the confidentiality matrix and attributes actually set in the system catalog for all roles registered in the specified confidentiality matrix. Refer to "Table returned by pgx_get_attribute_of_roles" for the format of the table.

Function name	Return value	Description
		Only confidentiality management role for the specified confidentiality matrix can execute this function.
pgx_get_privileges_on_level_and_group(confidential_matrix_name varchar, confidential_level_name varchar, confidential_group_name varchar)	setof record	Returns a table that allows you to compare the following for combinations of confidentiality objects registered in the specified confidentiality level and roles registered in the specified confidentiality group. Refer to "Table returned by pgx_get_privileges_on_level_and_group" for the format of the table. - Privileges specified by the confidentiality privilege settings that should be granted to the specified role - Privileges granted in the actual system catalog Only confidentiality management role for the specified confidentiality matrix can execute this function.
pgx_get_privileges_on_object(confidential_matrix_name varchar, object_name json)	setof record	Returns a table that allows you to compare the following for the specified confidentiality objects. Refer to "Table returned by pgx_get_privileges_on_object" for the format of the table. - Privileges dictated by confidentiality privilege settings that should be granted to all roles - Privileges granted in the actual system catalog Only confidentiality management role for the specified confidentiality matrix can execute this function.
pgx_get_privileges_on_role(confidential_matrix_name varchar, role_name json)	setof record	Returns a table that allows you to compare the following for the all confidentiality objects. Refer to "Table returned by pgx_get_privileges_on_role" for the format of the table. - Privileges specified by the confidentiality privilege settings that should be granted to the specified role - Privileges granted in the actual system catalog Only confidentiality management role for the specified confidentiality matrix can execute this function.
pgx_get_privileges_on_matrix(confidential_matrix_name varchar)	setof record	Returns a table that allows you to compare the following for the all confidentiality objects in the specified confidentiality matrix. Refer to "Table returned by pgx_get_privileges_on_matrix" for the format of the table. - Privileges defined by confidentiality privilege settings that should be granted to all roles registered in the confidentiality matrix. - Privileges granted in the actual system catalog Only confidentiality management role for the specified confidentiality matrix can execute this function.

Tables returned by pgx_get_attribute_of_objects

Column name	Type	Description
matrix_name	varchar(63)	Confidentiality matrix name
confidential_level_name	varchar(63)	Confidentiality level name
object_type	text	Confidentiality object type

Column name	Type	Description
object_schema	name	Confidentiality object schema name
object_table	name	Confidentiality object table name
object_name	text	Confidentiality object name
rowset_expression	json	Conditional expression when the confidentiality object is a row
encrypt_on_matrix	text	Encryption method and strength specified in the confidentiality matrix
encrypt_on_object	text	Actual encryption method and strength of confidentiality objects

Tables returned by pgx_get_attribute_of_roles

Column name	Type	Description
matrix_name	varchar(63)	Confidentiality matrix name
confidential_group_name	varchar(63)	Confidentiality group name
role_name	name	Role name
confidential_group_role	bool	Indicates whether it is a confidentiality group role or not. *true* if it is a confidentiality group role
superuser_on_matrix	bool	SUPERUSER attribute specified in the confidentiality matrix
superuser_on_role	bool	Actual SUPERUSER attribute of the role
createdb_on_matrix	bool	CREATEDB attribute specified in the confidentiality matrix
createdb_on_role	bool	Actual CREATEDB attribute of the role
createrole_on_matrix	bool	CREATEROLE attribute specified in the confidentia matrix
createrole_on_role	bool	Actual CREATEROLE attribute of the role
replication_on_matrix	bool	REPLICATION attribute specified in the confidentia matrix
replication_on_role	bool	Actual REPLICATION attribute of the role
bypassrls_on_matrix	bool	BYPASSRLS attribute specified in the confidentia matrix
bypassrls_on_role	bool	Actual BYPASSRLS attribute of the role

Tables returned by pgx_get_privileges_on_level_and_group, pgx_get_privileges_on_object, pgx_get_privileges_on_role and pgx_get_privileges_on_matrix

Column name	Type	Description
matrix_name	varchar(63)	Confidentiality matrix name
confidential_level_name	varchar(63)	Confidentiality level name

Column name	Type	Description
confidential_group_name	varchar(63)	Confidentiality group name
object_type	text	Confidentiality object type
object_scheme	name	Confidentiality object schema name
object_table	name	Confidentiality object table name
object_name	text	Confidentiality object name
role_name	name	Role name
privilege_list_on_matrix	text[]	Privileges specified by the confidentiality matrix settings. Output is separated by commas
privilege_list_on_object	text[]	Privileges specified by the confidentiality matrix settings. Output is separated by commas
policy_name	name	NULL if the confidentiality object is not of type rowset In the case of rowset type, rowset name (*1)
policy_setting_on_matrix	jsonb	NULL if the confidentiality object is not of type rowset In the case of rowset type, the rowset policy information set in the confidentiality matrix (*1)
policy_setting_on_policy	jsonb	NULL if the confidentiality object is not of type rowset In case of rowset type, row policy information set in the actual policy (*1)

*1: When adding a rowset type confidentiality object, multiple privileges can be set at once, which is not represented by a single row in this table. For example, if you set SELECT and DELETE privileges, you will see a row for SELECT privileges and a row for DELETE privilege. This is because rowset type access control uses PostgreSQL's row-level security POLICY. In this specification, POLICY for SELECT privilege is different from POLICY for DELETE privilege.