Fujitsu
Enterprise Postgres 17 SP2

Knowledge Data Management Feature User's Guide

Windows/Linux

Preface

Purpose of this document

This document describes the knowledge data management features of Fujitsu Enterprise Postgres.

Intended readers

This document is intended for those who use the knowledge data management feature.

To read this document, you need to have the following knowledge:

- Fujitsu Enterprise Postgres

- PostgreSQL

Structure of this document

The structure and content of this manual is shown below.

Chapter 1 Knowledge Data Management Feature

This chapter describes the overview of the knowledge data management feature and examples of how knowledge data can be used.

Chapter 2 Vector Data Management Feature

This chapter describes the vector data management feature.

Chapter 3 Semantic Text Search and Automatic Vectorization Feature

This chapter describes the semantic text search and the automatic vectorization feature for semantic search.

Chapter 4 Graph Management Feature

This chapter describes the graph management feature.

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 2.0: September 2025
Edition 1.0: March 2025
```

Copyright

Copyright 2025 Fujitsu Limited

Contents

Chapter 1 Knowledge Data Management Feature

This section describes an overview of the knowledge data management feature and examples of using knowledge data.

1.1 Overview of the Knowledge Data Management Feature

The Knowledge Data Management feature allows you to search based on semantic relationships using vectors and graphs, and manage those data.

When you use a large language model (LLM) in a retrieval-augmented generation (RAG) approach, you need external knowledge data to give to LLM. Fujitsu Enterprise Postgres can manage knowledge data in vector and graph formats, eliminating the need for a dedicated database for each format. You can also use Fujitsu Enterprise Postgres database multiplexing, access control, data encryption, and other features to securely manage your knowledge data.

In addition to existing searches, the Knowledge Data Management feature enables you to search knowledge data in three ways:

- Similar search of vector data

  Performs a similarity search between the specified vector data and the vector data stored in Fujitsu Enterprise Postgres.

- Searching text data based on semantic similarity

  Searches for semantic similarity between the specified text and the text stored in Fujitsu Enterprise Postgres using embedding in a semantic vector.

  When you store text data for semantic search in Fujitsu Enterprise Postgres, a vector (semantic vector) that captures the semantic similarity of the text data is automatically generated and stored in the database. When searching, the specified text data is automatically converted to vector data, and semantic search is performed by the vector similarity search.

  A hybrid search that combines text semantic similarity search and full-text search can also be utilized.

- Searching graphs based on relationships

  A graph is a data structure that represents entities and their relationships as nodes and the edges that connect them. You can search graphs based on properties and relationships.

Knowledge data can be accessed by executing SQL queries from the application to Fujitsu Enterprise Postgres. You can also use LangChain, an AI application development framework, to access knowledge data in Fujitsu Enterprise Postgres.
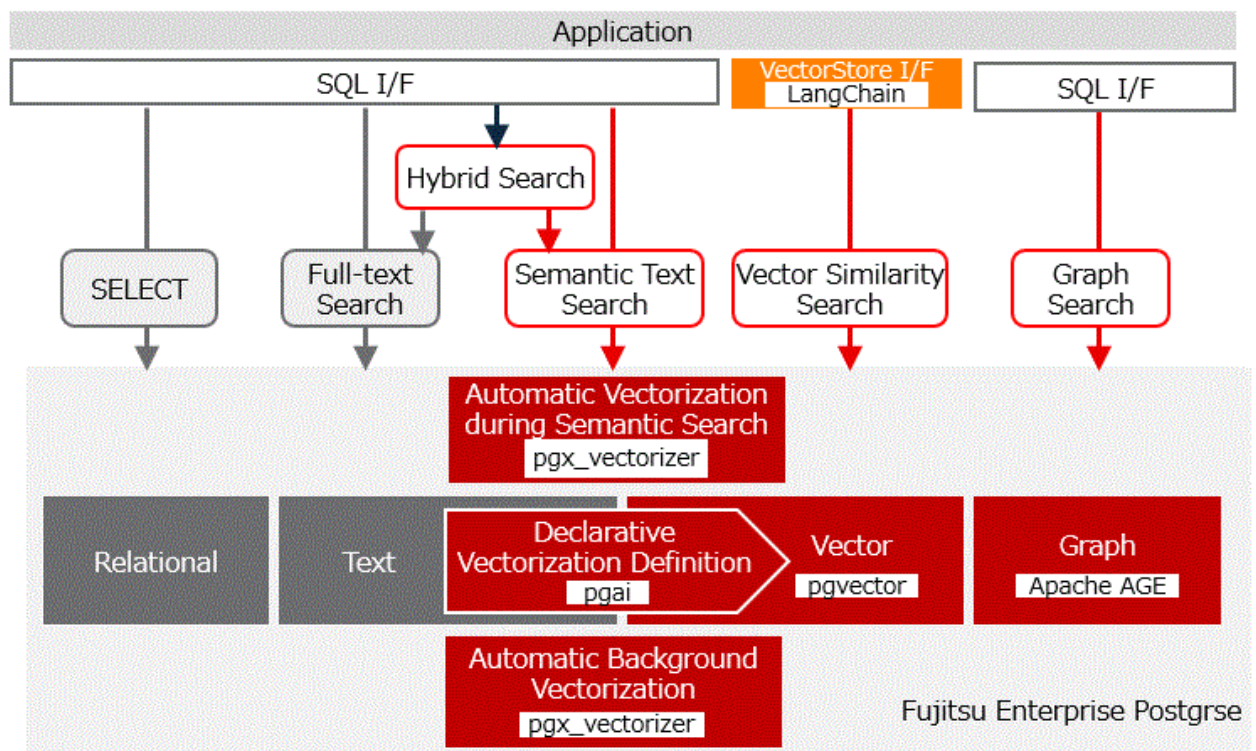
What the knowledge data management feature can do

The Knowledge Data Management feature enables you to:

- Vector data storage and retrieval using the vector data management feature

  - Storage of float32, float16, bit vectors, and sparse vectors

  - Vector neighbor search by cosine or euclidean distance

  - HNSW and IVFFlat vector index

  - DiskANN-based vector index suitable for large data sets

- Semantic text search and automatic vectorization

  - Automated background vectorization of newly added text data

  - Semantic search with automatic query vectorization, consistent with stored vector representation

  - Use of external vector embedding models such as Ollama and OpenAI

  - Hybrid search combining semantic text search and full-text search

- Graph data storage and retrieval using the graph data management feature

  - Storing graphs

  - Exploring and updating graphs with openCypher

- Application development support

  - LangChain linkage

    For access from Python applications using LangChain, refer to the technical documentation available at:

    https://www.postgresql.fastware.com/resource-center

1.2 Examples of Using Knowledge Data

Here are some examples of using knowledge data.

1.2.1 Example of Searching Text Data Based on Semantic Similarity

Here is an example of a typical RAG-based application that utilizes vector embedding of text data. Instead of performing vectorization on the application side, we use the semantic search feature for text data provided by Fujitsu Enterprise Postgres.

1. Creating text data

Use different algorithms and models to extract text from different data in your organization.

2. Storing text data in Fujitsu Enterprise Postgres

Store the text data in Fujitsu Enterprise Postgres. To use the semantic search function in text, declaratively define the vector representation to be used for the stored textual knowledge data. This automatically creates the vector data necessary for semantic search.

3. Searching text data

The text semantic search feature returns text that is semantically similar to the text you are searching for. No conversion to vector representation is required on the application side.
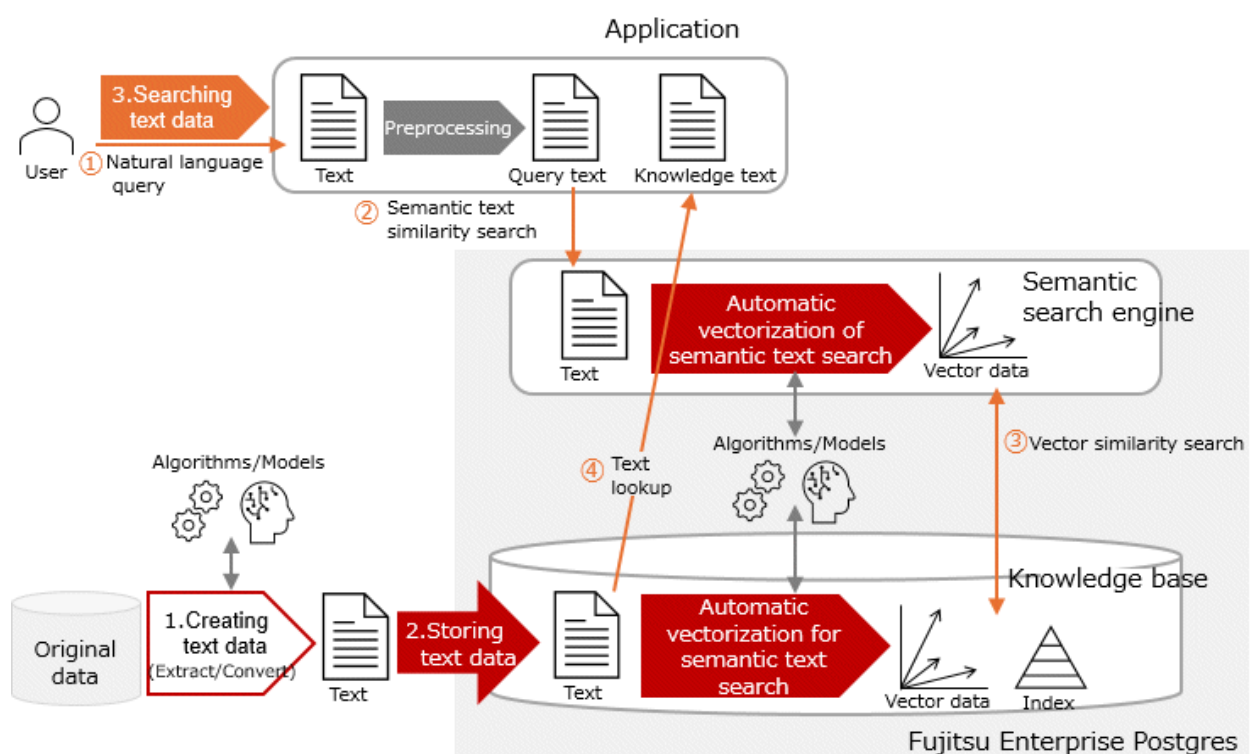
1.2.2 Example of Searching a Graph Based on Relationships

Here is an example of a graph as knowledge data is queried in natural language, and the obtained knowledge is converted into text for use.

1. Creating graph data

Extract graph data from a variety of data in your organization, such as using a relation extraction model.

2. Storing graph data in Fujitsu Enterprise Postgres

Store the created graph data in Fujitsu Enterprise Postgres. Nodes and edges can be added with Cypher queries using SQL functions.

3. Searching graph data

To search a graph, you specify a Cypher query to the SQL function. Converts the retrieved properties into a text representation for use.
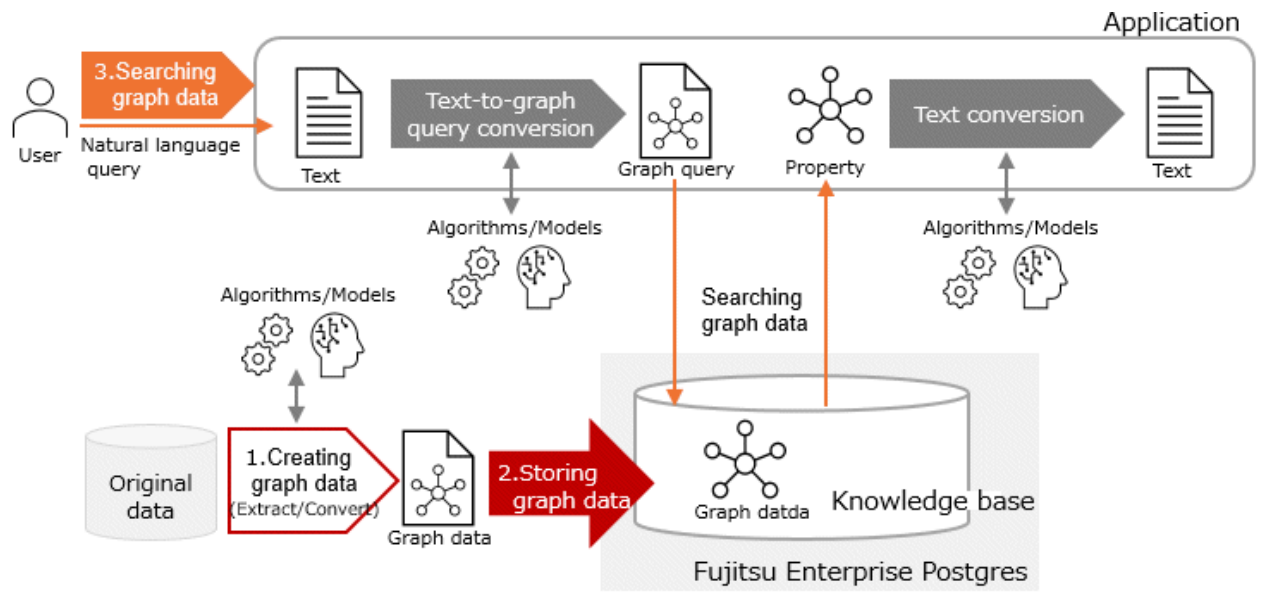
Chapter 2 Vector Data Management Feature

This chapter describes the vector data management feature, which provides the storage and search of vector data.

L pgvectorscale, described below, is only available on Linux.

2.1 Setting Up the Vector Data Management Feature

Vector data management features are provided by the OSS's pgvector and pgvectorscale.

Refer to "pgvector" in the Installation and Setup Guide for Server to set up pgvector.

If you use the StreamingDiskANN index, refer to "pgvectorscale" in the Installation and Setup Guide for Server to set up pgvectorscale.

In addition, to use the StreamingDiskANN index, a CPU that supports the AVX2 and FMA instruction sets is required. If you use a CPU that does not support AVX2 and FMA, an error will occur when creating a StreamingDiskANN index for a table that stores vector data, or when inserting data into a table with a StreamingDiskANN index created. Refer to the documentation for each CPU to see if the CPU supports AVX2 and FMA.

2.2 Storing and Searching Vector Data

The vector data management feature provides a new vector data type for storing vector data. Create a table with columns of vector data type and store vector data.

The similarity search of vector data is performed by calculating the distance between two vector data using the distance operator added by the vector data management feature, and by the closeness and ordering of the calculated distance.

Example) Vector similarity search

```
SELECT * FROM items ORDER BY embedding <-> '[3,1,2]' < 5 LIMIT 5;
```

Creating a vector index on a column of the vector data type causes the vector index to be used when searching for similarities using the distance operator.

P Point

Similarity searches using vector index are approximate similarity searches.

See

Refer to the pgvector documentation for information about vector data types, vector operations, distance types between vectors, and HNSW and IVFFlat vector indexes.

Refer to the pgvectorscale documentation for information about StreamingDiskANN indexes.

2.3 Protecting Vector Data

Vector data is stored as column values in tables, and can be protected by encryption, backup, multiplexing, replication, access control, and audit settings that specify the table, tablespace, database, or instance in which the vector data is stored.

2.4 Performance Tuning for Similar Search of Vector Data

You can verify that indexes are being used for similar searches of vector data by checking the access plan in the EXPLAIN statement.

Example) Access Plan for Vector Similar Search

Here is an example using pgvectorscale:

```
EXPLAIN SELECT * FROM items ORDER BY embedding <=> '[3,1,2]' LIMIT 5;
                                 QUERY PLAN
-------------------------------------------------------------------------------
Limit  (cost=24.75..24.96 rows=5 width=33)
   ->  Index Scan using idx_diskann  on items  (cost=24.75..445.75 rows=10000 width=33)
         Order By: (embedding <=> '[3,1,2]'::vector)
(3 rows)
```

See

Each vector index has parameters for tuning. See the documentation for pgvector and pgvectorscale for details.

2.5 Using Vector Data by Applications

If your application wants to work directly with vector data types, introduce a driver for each language in your application. If no driver is used, the vector data type is returned to the application as a string or array type.

See

For information on pgvector drivers for various languages, refer to below.

https://github.com/pgvector/pgvector?tab=readme-ov-file#languages

2.6 Quantitative Limits

Refer to the pgvector documentation for the quantitative limits of the data types provided by pgvector.

Chapter 3 Semantic Text Search and Automatic Vectorization Feature

This chapter describes the semantic text search and the automatic vectorization feature for semantic text search.

3.1 Overview of the Semantic Text Search and Automatic Vectorization Feature

Semantic text search

The semantic text search is a feature that searches for highly relevant texts based on semantic similarity of the text. This is achieved by utilizing vector representations that maintain the semantic similarity of text data. It is possible to use hybrid search that combines semantic text search using vector representation and full-text search based on string matching.

Automatic vectorization feature

The automatic vectorization feature automatically generates and stores vector representations corresponding to the text data inserted into the table. Use an external service called an embedded provider to generate vector representations of text.

The automatic vectorization feature is executed in the background by defining vectorization for the table to be searched. The definition of vectorization includes the embedded model that determines the vector representation used for semantic text search, the method of text segmentation, and the definition of the index.

When a vectorization is defined for table, a corresponding "embedded table" is created. This embedded table stores text chunks, which are units of the original table's text data divided into chunks, and the corresponding vector data. At the same time as defining the vectorization, you can define a vector index for the vector data and a full-text search index for the text chunks. An "embedded view" that combines the original table and the embedded table is also created.

Semantic text search and embedded view

Semantic text search and hybrid search are executed using embedded view as the search target.

When semantic text search and hybrid search are performed for embedded view, the text given as a query is internally converted into a vector representation, and a similarity search is conducted between the vector representation of the stored knowledge data.
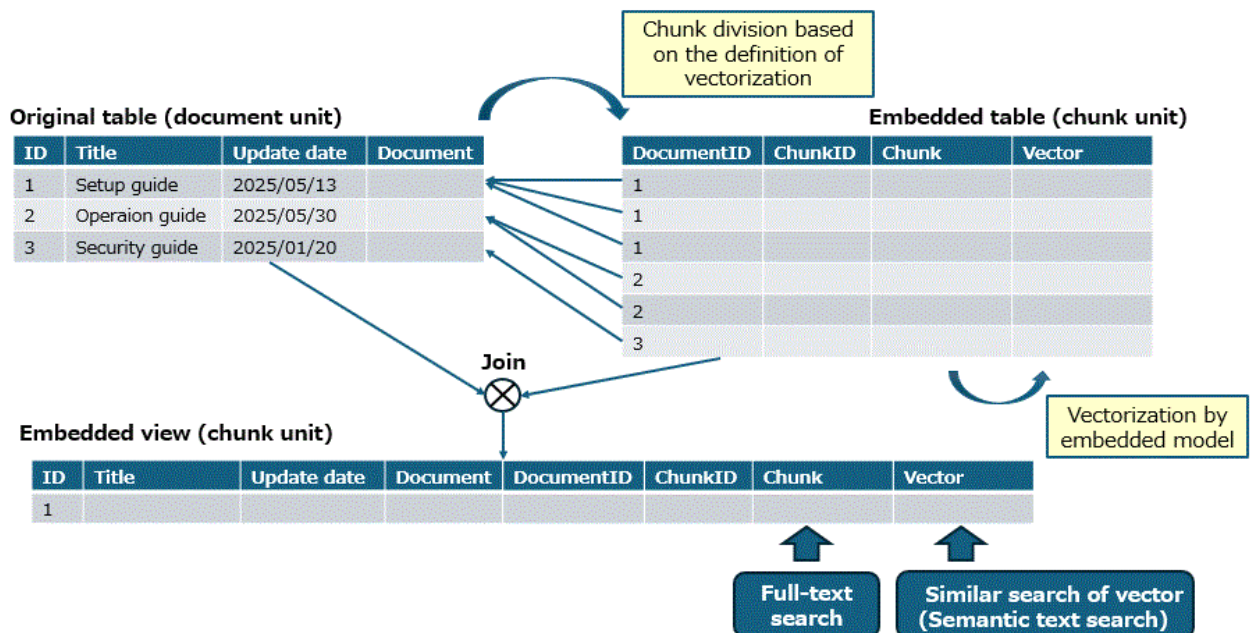
3.2 Installation of Semantic Text Search and Automatic Vectorization

3.2.1 Operating Environment

For details about the packages required for this feature to work, refer to the following sections in the Installation and Setup Guide for Server.

- Required Packages

- Related Software

3.2.2 Setup

Support for semantic text search and automatic vectorization is provided as an extension called pgx_vectorizer. pgx_vectorizer uses pgai, which relies on pgvector and plpython3u.

Refer to "pgvector" in the Installation and Setup Guide for Server to set up pgvector.

3.2.2.1 Setting Up pgai

As a superuser, execute the following command to set up pgai. "*<x>*" indicates the product version.

```
$ su -
# cp -r /opt/fsepv<x>server64/OSS/pgai-extension/* /opt/fsepv<x>server64/
```

📓 **Note**

....

With the Fujitsu Enterprise Postgres server feature installed, plpython3u is configured to use the following.

- For RHEL8, RHEL9: Python 3.9

- For SLES15: Python 3.6

By setting up pgai as above, plpython3u will be configured to use Python 3.11. Therefore, existing PL/Python programs that use plpython3u may no longer work.

If you want to return plpython3u to the configuration immediately after installation, refer to "3.2.3 Removing".

....

Configure the following before starting the instance.

Set the *<Python package installation destination>* to the installation destination listed in the "Related Software" of the "Installation and Setup Guide for Server".

```
PYTHONPATH=/opt/fsepv<x>server64/psycopg/python3.11/site-packages/:<Python package
installation destination>:$PYTHONPATH
```

3.2.2.2 Setting Up pgx_vectorizer

Setting parameters in the postgresql.conf file

Set the following parameters.

- Add pgx_vectorizer to the shared_preload_libraries parameter.

- Specify the maximum parallelism for vectorization processing in the pgx_vectorizer.max_vectorize_worker parameter.

- Add the following value to the value of the max_worker_processes parameter:

  - number of databases to enable pgx_vectorizer functionality +pgx_vectorizer.max_vectorize_worker+2

- If you have changed the installation destination of the Fujitsu Enterprise Postgres Server feature to a location other than the standard installation destination, specify the following for the pgx_vectorizer.pgai_worker_path parameter.

```
<Fujitsu Enterprise Postgres server feature Installation Directory>/OSS/pgai-
worker/bin/pgai
```

Enabling the pgx_vectorizer extension

Execute CREATE EXTENSION for the database that will use this feature.

Adding the CASCADE option will also enable the dependent pgai, pgvector, and plpython3u at the same time.

After CREATE EXTENSION, execute the start_vectorize_scheduler function to start the vectorize scheduler. The vectorize scheduler is a feature that schedules workers to perform vectorization.

Example) Connecting to the database "rag_database" using the psql command

```
rag_database=# CREATE EXTENSION IF NOT EXISTS pgx_vectorizer CASCADE;
CREATE EXTENSION
rag_database=# SELECT pgx_vectorizer.start_vectorize_scheduler(); -- Starting the
vectorize scheduler
```

After enabling the extended features, update the postgresql.conf parameter settings using commands such as pg_ctl reload.

Create a database user and set up a connection

Create a database user that the automatic vectorization feature will use when converting vectors in the background, and register it as the user that will convert vectors. Specify a password to use password authentication.

```
CREATE ROLE <worker_user>  PASSWORD `<worker password>` … LOGIN;
SELECT pgx_vectorizer.set_worker_setting('user', 'VECTORIZE_USER', '<worker_user>');
```

This database user will connect to Fujitsu Enterprise Postgres as an application, so modify the pg_hba.conf file for client authentication. Set it so that the database user created above can connect to the database that uses the pgx_vectorizer function from localhost using password authentication.

```
host    <ai-database>        <worker_user>         127.0.0.1/32       scram-sha-256
host    <ai-database>        <worker_user>         ::1/128            scram-sha-256
```

The background vectorization process runs with the privileges of the OS user that starts Fujitsu Enterprise Postgres. The password required for the above database user to connect to Fujitsu Enterprise Postgres is referenced from the password file of the OS user that starts Fujitsu Enterprise Postgres. Specify the information required to connect to the vectorization process. The password file used is in the default location. For details about the password file, refer to "The Password File" in the PostgreSQL Documentation.

3.2.2.3 Migration from Fujitsu Enterprise Postgres 17 SP1

If you have set up the pgx_vectorizer extension with Fujitsu Enterprise Postgres 17 SP1, you need to upgrade the pgx_vectorizer extension. Follow the steps below to perform the upgrade.

1. If the database server is running, stop the database server.

```
$ pg_ctl stop -D <data_directory>
```

2. Remove pgx_vectorizer from shared_preload_libraries parameter in postgresql.conf.

3. Upgrade the product.

4. Start the database server.

```
$ pg_ctl start -D <data_directory>
```

5. Update pgvector, pgai, and pgx_vectorizer.

```
ALTER EXTENSION vector UPDATE;
ALTER EXTENSION ai UPDATE;
ALTER EXTENSION pgx_vectorizer UPDATE;
```

6. Add pgx_vectorizer to shared_preload_libraries parameter in postgresql.conf.

7. Restart the database server.

```
$ pg_ctl restart -D <data_directory>
```

3.2.3 Removing

1. Connect to the database that is using this feature and execute DROP EXTENSION.

```
rag_databse=# DROP EXTENSION pgx_vectorizer;
DROP EXTENSION
```

2. As a superuser, execute the following command to remove pgai. "*<x>*" indicates the product version.

```
$ su -
# rm /opt/fsepv<x>server64/<Files copied during setup>
```

Information

The files copied during setup can be checked below.

```
# find /opt/fsepv<x>server64/OSS/pgai-extension
```

3. As a superuser, execute the following command to reset the plpython3u configuration to the state it was in immediately after the server feature was installed. "*<x>*" indicates the product version.

```
$ su -
# cp /opt/fsepv<x>server64/OSS/pgai-extension/lib/plpython3.9.so /opt/
fsepv<x>server64/lib/plpython3.so
```

3.2.4 Stopping the vectorize scheduler

To stop vectorize scheduler, refer to pg_stat_activity to check the pid of vectorize scheduler connected to the target database, and then use the pg_ctl kill TERM <pid> command or the SQL function pg_terminate_backend() to stop it. Because vectorize scheduler is continuously connected to the database, if you want to delete or change a database that has CREATE EXTENSIONed pgx_vectorizer, you must stop vectorize scheduler before performing the operation.

You can check the pid of vectorize scheduler connected to the current database by executing the following SQL. The backend_type of vectorize scheduler includes 'vectorize scheduler'.

```
SELECT pid FROM pg_stat_activity WHERE datid = (SELECT oid FROM pg_database WHERE datname =
current_database()) AND backend_type LIKE '%vectorize scheduler%';
```

Below is an example of using pg_terminate_backend().

```
rag_database=>SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE datid = (SELECT
oid FROM pg_database WHERE datname = current_database()) AND backend_type LIKE '%vectorize
scheduler%';

 pg_terminate_backend
---------------------
 t
(1 row)
```

3.2.5 Credentials Protection

3.2.5.1 Encrypting Credentials

The pgx_vectorizer.worker_setting_table, which is created when pgx_vectorizer is set up, should be protected because it stores the API key that the worker performing vectorization uses to access the embedded provider that offers the vector embedding feature.

To encrypt this table, either encrypt the entire database to which the pgx_vectorizer feature is being added in advance, or move this table to an encrypted tablespace after adding the extension feature.

Example) When encrypting the entire database for which this feature is enabled

```
postgres=# CREATE DATABASE rag_database TABLESPACE = encrypted_tablespace;

rag_database=> CREATE EXTENSION pgx_vectorizer;
```

Example) Moving to encrypted table space

```
rag_database=> ALTER TABLE pgx_vectorizer.worker_setting_table SET TABLESPACE
encrypted_tablespace;
```

3.2.5.2 Restrict Access to Credentials

The pgx_vectorizer.worker_setting_table, which stores the credential information, is configured so that only the user who executed the CREATE EXTENSION of the pgx_vectorizer feature can access it. If you want to allow access by other users, use the GRANT statement to grant access privilege to those users.

3.3 Preparation for Semantic Text Search

Semantic text search is achieved by similarity search between vectors (semantic vectors) that store the semantic similarity of text. Therefore, in order to use the semantic text search feature, you need to define vectorization (what kind of vector representation to use) for the text data to be searched and create vector data.

3.3.1 Configuring Embedded Providers (for Workers)

Before defining a vectorization, you must configure the worker process that will perform the vectorization to access the embedded provider.

Example) Settings when using Ollama as an embedded provider

```
rag_database=> SELECT pgx_vectorizer.set_worker_setting('ollama', 'OLLAMA_BASE_URL',
'http://your.ollama.server:11434');
```

Example) Settings when using OpenAI as an embedded provider

```
rag_database=> SELECT pgx_vectorizer.set_worker_setting('openai', 'OPENAI_API_KEY', 'your
api key');
```

📖 Information

. . . .

Automatic vectorization is performed by worker processes that are regularly initiated. Additionally, the worker processes are initiated by Fujitsu Enterprise Postgres.

. . . .

📘 Point

. . . .

In this configuration, credentials for the worker processes to access the embedded provider are stored in an area accessible by Fujitsu Enterprise Postgres. Because the configuration information for the embedded provider, including the API key, is

stored in the table as plain text, place the table in an encrypted tablespace and protect it with the Fujitsu Enterprise Postgres transparent data encryption feature. For instructions, refer to "3.2.5 Credentials Protection".

3.3.2 Configuring Embedded Providers (for Semantic Text Search)

To access embedded providers for semantic text search, use the pgai settings. Refer to the pgai documentation for details.

3.3.3 Definition of Vectorization

The definition of vectorization is done with the pgx_vectorizer.pgx_create_vectorizer function provided by pgx_vectorizer. In the vectorization definition, you can specify information about the table that contains the text data to be vectorized, the embedding model and dimensions directly related to vector representation, chunking processing, as well as specify the timing of vectorization as a schedule. If you want to immediately schedule automatic vectorization in the background with Fujitsu Enterprise Postgres, specify the pgx_vectorizer.schedule_vectorizer function. Additionally, if you want to specify the vectorization later or manually control the timing of vectorization, specify the ai.scheduling_none function.

When using hybrid search, you can simultaneously define a GIN or GiST index for full-text search used in hybrid search.

When defining vectorization, an embedded view combining a table containing text data and a table containing vector data is defined. The embedded view is a view that adds the following columns to the original table, and for each piece of text data in the original table, it holds records divided into chunks.

Column	Type	Description
embedding_uuid	uuid	Text chunk ID
chunk_seq	integer	Sequential number of chunks in a text data
chunk	text	Text chunk
embedding	vector	Vector representation of the chunk

Information

In the following cases, refer to "Definition of full-text search index" to define the index.

- When using an index other than GIN or GiST in hybrid search

- When you have already performed only vectorization and want to add an index for full-text search later

Fujitsu Enterprise Postgres provides pg_trgm and pg_bigm as full-text search features using GIN and GiST indexes. When using these as full-text search features, set up pg_trgm and pg_bigm in advance. Refer to the following for setup instructions.

- pg_trgm

  "Additional Supplied Modules and Extensions" in the PostgreSQL Documentation

- pg_bigm

  "pg_bigm" in the Installation and Setup Guide for Server

Example) Definition of vectorization and full-text search index

Below is an example of simultaneously defining a vectorization and a GIN index for full-text search on the knowledge data table sample_table in text format.

```
rag_database=> SELECT pgx_vectorizer.pgx_create_vectorizer(
    'sample_table'::regclass,
    destination => 'sample_embeddings',
    embedding => ai.embedding_ollama('all-minilm', 384),
    chunking => ai.chunking_recursive_character_text_splitter('contents'),
    processing => ai.processing_default(batch_size => 200, concurrency => 1),
```

```
    scheduling => pgx_vectorizer.schedule_vectorizer(interval '1 hour'),
    indexing => ai.indexing_hnsw(min_rows =>50000, opclass => 'vector_cosine_ops'),
    fulltext_indexing => pgx_vectorizer.pgx_fulltext_indexing_gin(opclass =>
'gin_trgm_ops')
    );
pgx_create_vectorizer
------------------
             1 - The ID of the created vectorizer
(1 row)
```

The definition of the full-text search index created by the pgx_create_vectorizer function can be checked in the pgx_vectorizer.pgx_fulltext_index table.

Note

Regarding the table containing text subject to vectorization, the following cannot be changed:

- Changes to the schema containing the table

- Changes to the table name

- Changes to the primary key, addition or removal of columns constituting the primary key

- Name and data type of columns included in the primary key

- Name and data type of columns containing text data subject to vectorization

Changing these may cause automatic vectorization or semantic text search to fail. If you want to change them, or if you have already changed them, you need to redefine the vectorization. In this case, re-vectorization of all text in the target table is required.

Point

Users can implement their own full-text search functionality, other than pg_trgm or pg_bigm provided by Fujitsu Enterprise Postgres, by defining indexes supported by that full-text search functionality and specifying search conditions using its operators, allowing for hybrid search with custom full-text search capabilities.

Definition of full-text search index

If you use an index other than GIN or GiST index for full-text search, manually define the index for the embedding table created after defining the vectorization. The name of the embedding table is the name with "_store" added to the end of the embedding view name specified in the destination argument when defining the vectorization. The column name for text chunks in the embedding table is chunk.

In the following example, a full-text search index my_iam, other than GIN and GiST, is defined in the embedded table sample_embeddings2_store.

```
CREATE INDEX sample_fti ON sample_embeddings2_store (chunk) USING my_iam;
```

Deleting full-text search index

You can delete the full-text search index by removing the information of the full-text search index from the vectorization and then executing the DROP INDEX statement.

```
rag_database=> SELECT
pgx_delete_fulltext_index_config(pgx_vectorizer.get_vectorizer_id(view_name =>
'sample_embeddings');
rag_database=> SELECT indexname FROM pg_indexes where tablename =
'sample_embeddings_store';
indexname
-----------------------------------------
```

```
sample_embeddings_store_chunk_idx - It is created for the column called "chunk".

rag_database=> DROP INDEX sample_embeddings_store_chunk_idx;
```

Changing the full-text search index

When changing the full-text search index defined by the pgx_create_vectorizer function, first delete the full-text search index and then define a new index.

```
--Deleting full-text search index
rag_database=> SELECT
pgx_delete_fulltext_index_config(pgx_vectorizer.get_vectorizer_id(view_name =>
'sample_embeddings');
rag_database=> SELECT indexname FROM pg_indexes where tablename =
'sample_embeddings_store';
indexname
----------------------------------------
sample_embeddings_store_chunk_idx - It is created for the column called "chunk".

rag_database=> DROP INDEX sample_embeddings_store_chunk_idx;

-- Adding full-text search index
rag_database=> CREATE INDEX ON sample_embeddings_store USING gin (chunk gin_bigm_ops);
```

3.3.4 Granting Privilege to Execute Functions

When executing the semantic text search and full-text search, you must grant the executing user function privileges that exist in the ai schema.

```
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA ai TO <user>;
```

3.4 Storing Vector Data for Semantic Text Search

Vectorization for text semantic text search is performed automatically according to the vectorization definition, and is saved as a vector data type column value in an internally created table so that the same text does not need to be vectorized again.

Vector data is generated and stored in the background according to the schedule specified when defining the vectorization. If new text data is added to the table on which the vectorization is defined, the corresponding vector data is automatically added asynchronously.

Information

Until the vector data has been created, the text data cannot be used for semantic text search.

3.5 Protecting Vector Data for Semantic Text Search

Vector data for semantic text search is automatically protected by backup, multiplexing, or replication settings that specify the tablespace, database, or instance in which it is stored.

This section describes the points to keep in mind when using Fujitsu Enterprise Postgres features to protect vector data for semantic text search.

Vectorization data using the semantic vector embedding model is a conversion that corresponds the semantic similarity between the original data to the closeness of the distance between vectors. Because there is a risk that the meaning of the original data can be inferred from vector data, it should be protected at the same level as the original data.

Multiple database objects are created by the vectorization definition. These database objects are the targets of protection.

3.5.1 Encrypting Vector Data for Semantic Text Search

Several database objects are created for semantic text search, and if you want to encrypt these database objects together, place the entire database in an encryption tablespace.

Example) Encrypting the entire database

```
postgres=# CREATE DATABASE rag_database TABLESPACE = encrypted_tablespace;

rag_database=> CREATE EXTENSION pgx_vectorizer;
```

To encrypt database objects for semantic text search when a single tablespace cannot be used per database, temporarily change the default tablespace to an encrypted tablespace before defining vectorization.

Indexes for vector data are created when the index creation conditions are met. To encrypt an index, change the tablespace after index creation, or create the index manually without specifying an index in the create_vectorizer function. For information about tables containing vector data created by this feature, refer to "3.13.4 Tables/Views Created by Semantic Text Search and Automatic Vectorization Feature".

3.5.2 Restricting Access to Vector Data for Semantic Text Search

Access to vector data is restricted by controlling access to the tables in which the vector data is stored. Access restrictions are set using the confidentiality management feature of Fujitsu Enterprise Postgres.

The following is an example of using the confidentiality management feature to grant reference rights to the table sample, which contains text data, and the table sample_embeddings_store, which contains vector data generated by this feature.

1. Refer to the "Confidentiality Management" in the Security Operations Guide to define confidentiality management role, confidentiality matrix, confidentiality level, and confidentiality group.

2. Grant confidentiality privilege for the table to the confidentiality group.

```
SELECT pgx_grant_confidential_privilege('rag_matrix', 'level1', 'group1', '{"schema":
["ALL"], "table": ["SELECT"]}');
```

3. Add tables containing text data and embedded tables as confidentiality object to the confidentiality level.

```
SELECT pgx_add_object_to_confidential_level ('rag_matrix', 'level1',
'[{
    "type":"table",
    "object":[
        {
            "schema":"public",
            "table":["sample"]
        },
        {
            "schema":"public",
            "table":["sample_embeddings_store"]
        },
    ]
}]');
```

4. Add the role to the confidentiality group.

```
SELECT pgx_add_role_to_confidential_group('rag_matrix', 'group1', '["rag_user"]');
```

Information

A vector table contains foreign keys, vector data, and chunks, but it is not necessary to set access privileges on a column-by-column basis; setting access privileges on a table-by-table basis is sufficient.

The following is an example of setting row-level security for table sample, which contains text data, and table sample_embeddings_store, which contains vector data generated by this function. In this example, sample and sample_embeddings_store contain user names in a column called username, and sample_embeddings_store has the primary key (id) of sample as a foreign key (id). When using row-level security, grant the BYPASSRLS attribute to the user set in VECTORIZE_USER.

1. Refer to the "Confidentiality Management" in the Security Operations Guide to define confidentiality management role, confidentiality matrix, confidentiality level, and confidentiality group.

2. Grant confidentiality privilege for the rowset to the confidentiality group.

```
SELECT pgx_grant_confidential_privilege('rag_matrix', 'level1', 'group1', '{"table":
["SELECT"], "schema": ["ALL"],"rowset":["SELECT"]}');
```

3. Enable row-level security for the table.

```
ALTER TABLE sample ENABLE ROW LEVEL SECURITY;
ALTER TABLE sample_embeddings_store ENABLE ROW LEVEL SECURITY;
```

4. Ensure that the tables on which the target view is based are checked against the privileges of the view's user.

```
ALTER VIEW sample_embeddings SET (security_invoker = true);
```

5. Add the ai schema, tables containing text data and embedded tables and embedded views as confidentiality objects to the confidentiality level.

```
SELECT pgx_add_object_to_confidential_level ('rag_matrix', 'level1',
'[{
    "type":"schema",
    "object":[
        {"schema":"ai"},
        {"schema":"pgx_vectorizer"}
    ]
},
{
    "type":"table",
    "object":[
        {
            "schema":"public",
            "table": ["sample", "sample_embeddings_store", "sample_embeddings"]
        },
        {
            "schema":"ai",
            "table": ["vectorizer"]
        }
    ]
}]');
```

6. Add the table containing text data and the rowset of the embedded table as confidentiality objects to the confidentiality level.

```
SELECT pgx_add_object_to_confidential_level ('rag_matrix', 'level1',
'[{
    "type":"rowset",
    "object":[
    {
        "schema":"public",
        "table": "sample",
        "rowset_name": "rowset1",
        "rowset_expression":[
        {
            "as": "permissive",
            "using": "username = current_user"
        }]
```

```
        },
        {
            "schema":"public",
            "table": "sample_embeddings_store",
            "rowset_name": "rowset1",
            "rowset_expression":[
            {
                "as": "permissive",
                "using": "EXISTS (SELECT 1 FROM public.sample WHERE public.sample.id =
public.sample_embeddings_store.id AND public.sample.\"username\" = current_user)"
            }]
        }
        ]
}]');
```

7. Add the role to the confidentiality group.

```
SELECT pgx_add_role_to_confidential_group('rag_matrix', 'group1', '["rag_user"]');
```

3.5.3 Recording Access to Vector Data for Semantic Text Search

The audit log feature of Fujitsu Enterprise Postgres is used to record access to vector data in the audit log. When specifying individual database objects to be audited, also specify the tables that store vector data as audit targets.

For information about tables containing vector data created by this feature, refer to "3.13.4 Tables/Views Created by Semantic Text Search and Automatic Vectorization Feature".

3.6 Monitoring Vectorization Processing for Semantic Text Search

If you use automatic background vectorization in your database, the corresponding vector data will be automatically changed when text data is added, updated, or deleted. However, because the generation of vector data for new data is performed asynchronously, newly added data will not be reflected immediately in semantic similarity search. Also, if you use an external service to create vector data, you will need to check for errors.

3.6.1 Checking the Vectorization Queue

You can check the number of texts waiting to be vectorized for each vectorization definition by referencing the ai.vectorizer_status view, and you can check the number for a specific vectorization definition by using the ai.vectorizer_queue_pending function.

Example) Check the ai.vectorizer_status view

```
rag_database=> SELECT * FROM ai.vectorizer_status;
-[ RECORD 1 ]-+------------------------------------
id            | 1
source_table  | public.sample_table
target_table  | public.sample_embeddings_store
view          | public.sample_embeddings
pending_items | 1000
disabled      | f
```

Example) Check with the ai.vectorizer_queue_pending function

```
SELECT ai.vectorizer_queue_pending( pgx_vectorizer.get_vectorizer_id(view_name =>
'sample_embeddings') );
-[ RECORD 1 ]------------+--
vectorizer_queue_pending | 1000
```

If this value remains at 0 or a low value, you can determine that the vectorization process is on time. If this value tends to increase beyond the execution interval specified in the schedule or data addition interval, refer to 3.6.2 Checking the Status of Vectorization Processing" to check whether an error has occurred in the vectorization process, and refer to "3.6.3 Checking the Scheduler for Vectorization Processing" to check whether the vectorize scheduler is running.

If no errors have occurred, the vectorization processing speed is likely slow compared to the data addition speed. If the load is temporarily high, start a temporary vectorization process with the pgx_vectorizer.run_vectorize_worker function. If not, change the schedule with the pgx_vectorizer.alter_vectorizer_schedule function, or change the parallelism or the upper limit of the amount of data to be processed in one startup with the pgx_vectorizer.alter_vectorizer_processing function.

Example) Changing the execution interval to 5 minutes

```
SELECT
pgx_vectorizer.alter_vectorizer_schedule(pgx_vectorizer.get_vectorizer_id(view_name =>
'sample_embeddings'), interval '5 m');
```

Example) When changing the parallelism to 2 and the amount of data to be processed at one time to 200

```
SELECT
pgx_vectorizer.alter_vectorizer_processing(pgx_vectorizer.get_vectorizer_id(view_name =>
'sample_embeddings'), batch_size => 200, concurrency => 2);
```

Use a monitoring tool to check the time trend of the number of texts waiting for vectorization. If there is an error in the parameter value set by set_worker_setting, the vectorization process will not be executed. A message will be output to the server log, so please check it together with the ai.vectorizer_status view.

3.6.2 Checking the Status of Vectorization Processing

By referring to the ai.vectorizer_errors view, you can check the details of the errors that occurred during the vectorization process, the time of occurrence, the number of occurrences, etc.

Example) Check the details of the most recent error

```
rag_database=> SELECT * FROM ai.vectorizer_errors ORDER BY recorded DESC LIMIT 50;
-[ RECORD 1 ]----------------------------------------------------------
--------------------------
id      | 1
message | embedding provider failed
details | {"provider": "ollama", "error_reason": "model \"all-minilm\" not fou
nd, try pulling it first"}
recorded | 2025-02-03 06:47:35.958882+00
-[ RECORD 2 ]----------------------------------------------------------
--------------------------
id      | 1
message | embedding provider failed
details | {"provider": "ollama", "error_reason": "model \"all-minilm\" not fou
nd, try pulling it first"}
recorded | 2025-02-03 06:47:41.250279+00
```

Example) Check the number of errors for a vectorization definition

```
rag_database=> SELECT COUNT(*) FROM ai.vectorizer_errors WHERE id =
pgx_vectorizer.get_vectorizer_id(view_name => 'sample_embeddings');
 count
-------
    20
(1 row)
```

Check the details of the error and remove the cause. Some embedded providers have a maximum load per period. If an error occurs because the load exceeds these conditions, use the pgx_vectorizer.alter_vectorizer_schedule or pgx_vectorizer.alter_vectorizer_processing function to adjust the worker schedule or parallelism.

3.6.3 Checking the Scheduler for Vectorization Processing

You can confirm that the scheduler for vectorization is running in the pg_stat_activity view.

```
SELECT * FROM pg_stat_activity WHERE backend_type LIKE '%pgx_vectorizer%';
```

3.7 Temporarily Disabling Vectorization Processing for Semantic Text Search

The vectorization process of this feature is performed periodically according to the specified schedule. For example, if you want to temporarily stop vectorization processing to reduce the impact on other work, you can disable vectorization processing using the ai.disable_vectorizer_schedule function.

```
rag_database=> SELECT
ai.disable_vectorizer_schedule(pgx_vectorizer.get_vectorizer_id(view_name =>
'sample_embeddings')); -- Specify the ID of the vectorizer you want to disable
```

To enable it, run the ai.enable_vectorizer_schedule function.

```
rag_database=> SELECT
ai.enable_vectorizer_schedule(pgx_vectorizer.get_vectorizer_id(view_name =>
'sample_embeddings')); -- Specify the ID of the vectorizer you want to enable
```

3.8 Semantic Text Search

For text data with vectorization defined, you can use the semantic text search feature to search for semantically similar text. The pgx_vectorizer.pgx_similarity_search function is used for the semantic text search.

The values in the distance column represent the distance between the text specified as an argument and the chunk, and the smaller the distance value, the higher the similarity to the text.

Example) Semantic text search

```
SELECT * FROM pgx_vectorizer.pgx_similarity_search('sample_embeddings'::regclass, 'text
for search', 5,
'<=>');
embedding_uuid                                 | chunk                    | distance
-----------------------------------------------+--------------------------+-------------
89A-927B-4271-82A3-6A73E8962B1C                | Action items assigned.   | 0.1027381927364
8E73B5F2-461A-4622-89A9-C1D364F4E19B           | Next steps discussed.    | 0.2938471928374
5D39A6E1-B226-4197-9F03-A78B80A509C2           | Timeline adjusted.       | 0.5183749128735
2F97C1E5-6E8A-400F-8692-177D77740B6A           | Budget approved.         | 0.7815239187521
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11           | Status update.           | 0.9274618273645
(5 rows)
```

Information

•••

The pgx_vectorizer.pgx_similarity_search function specifies the type of distance to be used in vector similarity searches performed for semantic text searches. If this specification differs from the distance specified when defining an index for vector data, the index will not be used during the search.

•••

3.9 Changing the Vector Representation Used in Semantic Text Search

You can have multiple vector representations for the text you want to perform semantic text search on. A vector representation is associated one-to-one with a vectorization definition, and you can have a different vector representation by defining a new vectorization. In the vectorization definition, you can specify the embedding model to be used, etc. If you do not need the previous vector representation, delete the vectorization definition.

Example) Defining a new vectorizer and deleting the old one

```
rag_database=> SELECT ai.drop_vectorizer(pgx_vectorizer.get_vectorizer_id(view_name =>
'sample_embeddings'), drop_all => true);

rag_database=> SELECT pgx_vectorizer.pgx_create_vectorizer(…);
```

3.10 Performance Tuning of Semantic Text Search

A semantic text search internally performs a similarity search on vector data. If an index is not specified for the vector data, all vector data is scanned for each semantic text search. You can check whether an index is being used in a semantic text search by running pgx_similarity_search_checking_index.

Example) Check if an index is used in a semantic text search

```
rag_database=> SELECT * FROM
pgx_similarity_search_checking_index('sample_embeddings'::regclass, 'text for search', 5,
'<=>');
ERROR: opclass of index is vector_ip_ops, but the distance_operator is <+>
```

If search results are output when you execute pgx_similarity_search_checking_index, the index is being used. If the search ends with an error as shown above, the distance operator specified in the index definition may differ from the distance type specified in the semantic text search. If the distance type specified in the semantic text search is incorrect, correct the distance specified. If the distance operator specified in the index definition is incorrect, delete the index and re-create the correct index.

Indexes for vectors explicitly define for the table that stores vectors that are created internally when vectorization is defined.

Example) Changing the vector table index

```
-- Check and delete the old index name
rag_database=> SELECT indexname FROM pg_indexes where tablename =
'sample_embeddings_store';
               indexname
----------------------------------------
 sample_embeddings_store_pkey
 sample_embeddings_store_id_chunk_seq_key
 sample_embeddings_store_embedding_idx - It is created for a column called embedding.
(3 rows)
rag_database=> DROP INDEX sample_embeddings_store_embedding_idx
-- Add the correct index
rag_database=> CREATE INDEX ON sample_embeddings_store USING hnsw (embedding
vector_l1_ops);
```

3.11 Hybrid Search

In hybrid search, results from both semantic text search and full-text search are combined, ranked based on scores considering both results, and the specified number of results are returned in order of highest score.

Hybrid search flow
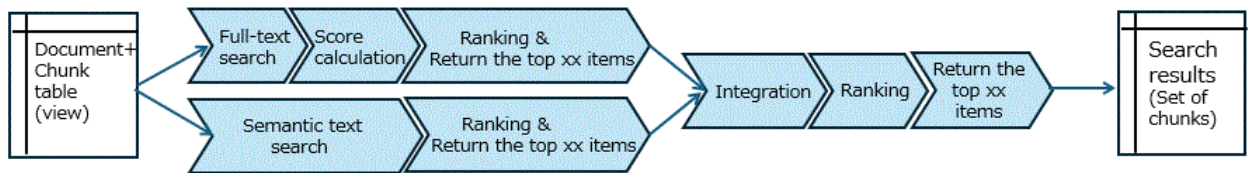
Hybrid search uses the SQL function pgx_vectorizer.pgx_hybrid_search function. For pgx_vectorizer.pgx_hybrid_search function, refer to "3.13.3.2 Hybrid Search".

In the following example, a search is performed on the embedded view sample_embeddings created by the vectorization definition. The union of the top 10 results of full-text search and semantic text search of text is taken, and the top 20 results based on the final score are returned. In this example, partial match search using the LIKE operator is performed as a full-text search.

Example) Hybrid search

```
rag_database=> SELECT context_id, chunk, score FROM pgx_vectorizer.pgx_hybrid_search('
{
  "target_view": "sample_embeddings",
  "search_fusion": "UNION",
  "topN": 20,
  "semantic": {
    "search_text": "text for search",
    "num_result": 10,
    "score_weight": 10
  },
  "fulltext": {
  "search_condition": "chunk LIKE ''%text%''",
  "score_expression": "CASE WHEN chunk LIKE ''%text%'' THEN 1 ELSE 0 END"
  }
}'::JSONB
);
```

For the return results of the hybrid search, refer to "Hybrid search results".

Information
....
Combined search results are ordered using a method called weighted RRF (Reciprocal Rank Fusion). For more details, refer to "Score calculation".
....

3.12 Improving the Accuracy of Hybrid Search

Hybrid search is a search method that combines multiple search techniques to leverage their strengths and complement their weaknesses. To improve search accuracy by combining multiple search techniques, it is necessary to understand the search accuracy of each technique and choose a combination method accordingly. Below, we will explain how to calculate the search accuracy of individual search techniques (subqueries) that make up hybrid search.

3.12.1 Overview of Hybrid Search Tuning

The accuracy of search processing can be calculated by preparing pairs of query conditions for evaluation and correct answers (for example, the search results that should be returned for that query) and comparing the results searched with those query conditions to the correct answers. The result obtained from hybrid search is only the final search result. Since we cannot know what the search results of individual subqueries were, their search accuracy cannot be calculated. Therefore, to calculate the search accuracy of subqueries, we use hybrid search trace information. Trace information is a record of the executed hybrid search, including the search conditions and search results of each subquery executed internally. The search accuracy can be calculated from the subquery search results included in the trace information. The accuracy of hybrid search using trace information can be calculated in the following two ways:

a. Obtain the search results of each subquery recorded as traces and calculate using external evaluation tools.

b. Calculate search accuracy within the database by inputting evaluation results for each text chunk that is a search result into the database.

When evaluating the entire system, such as RAG-based applications, it is appropriate to use method a., which utilizes external evaluation tools. On the other hand, if you want to analyze the accuracy of individual subqueries or text chunks in detail without using tools, method b., which calculates accuracy within the database, is recommended.

Below are the steps for each method.
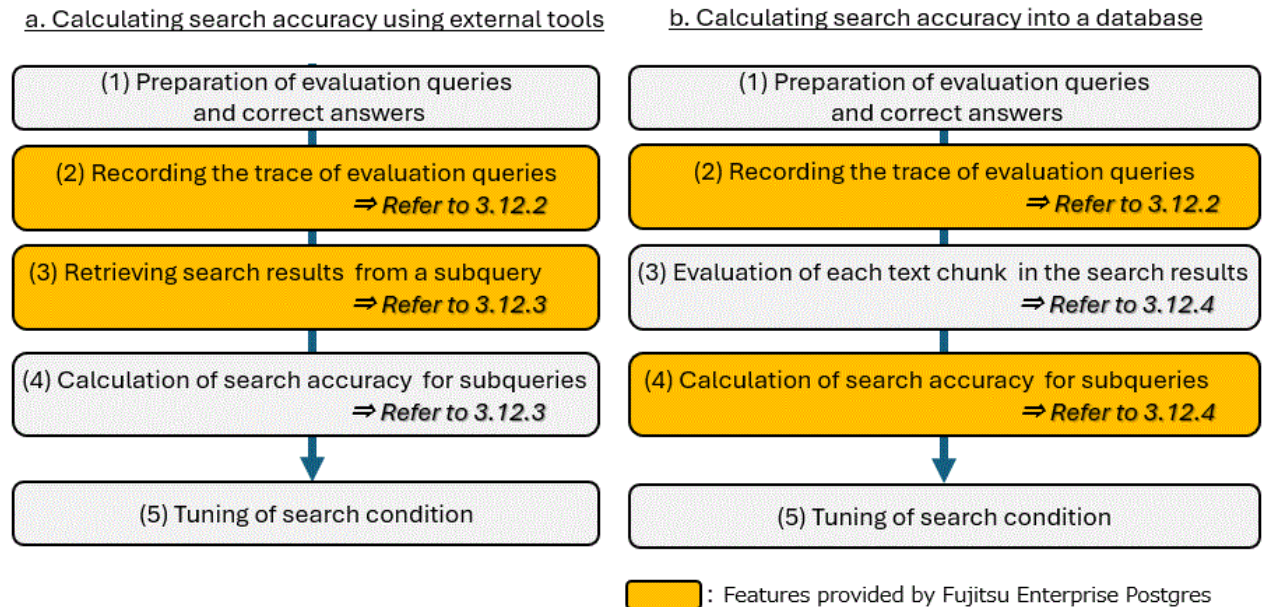
For basic ideas on the evaluation of knowledge data search, refer to "3.13.6 Evaluation of Knowledge Data Search".

3.12.2 Recording and Deleting Traces of Hybrid Search

To record trace for evaluation queries, a user with trace recording privileges enables the trace and executes a hybrid search using the evaluation queries. As a result, the trace information is recorded in the table. Trace recording can only be done on the primary server. Even if a hybrid search with trace enabled is performed on a hot standby server, the trace will not be recorded. The trace privileges are granted to the database user performing the search by the extension owner using the pgx_grant_access_on_hybrid_search_trace function. To enable the trace, the database user performing the hybrid search sets the enable_hybrid_search_trace parameter to on in their session. Regarding the enable_hybrid_search_trace parameter, refer to "3.13.5 Parameters".

Example) Privilege granted by the owner of the extension

```
rag_databse=# SELECT pgx_vectorizer.pgx_grant_access_on_hybrid_search_trace('app_user');
```

Example) Setting session parameters by users performing searches and executing hybrid searches

```
rag_database=> SET pgx_vectorizer.enable_hybrid_search_trace = 'on';
rag_database=> SELECT * FROM pgx_vectorizer.pgx_hybrid_search(...);
```

When a trace is recorded, a query ID that identifies the search request required for subsequent operations is issued. The query ID is returned as a result of a hybrid search, and can also be confirmed by referring to the table where trace information is recorded (pgx_trace_query table). This table records the database user who executed the hybrid search, the time, and the application name, allowing you to identify the query ID using these details.

Example) The database user who executes the command refers to the query ID of the hybrid search executed within an hour

```
rag_database=> SELECT queryid FROM pgx_vectorizer.pgx_trace_query WHERE retrieval_time >=
(current_timestamp - interval '1 hour') ORDER BY retrieval_time;
queryid
---------
```

```
   10001
(1 row)
```

After completing the evaluation and tuning, trace information and evaluation values will be deleted. By deleting records with the target query ID from the pgx_trace_query table, all related information will be removed.

Example) Delete trace information by specifying the query ID

```
rag_database=> DELETE FROM pgx_vectorizer.pgx_trace_query WHERE queryid = 10001;
```

Example) The administrator deletes all trace information older than 30 days

```
rag_databse=# DELETE FROM pgx_vectorizer.pgx_trace_query WHERE retrieval_time <=
(current_timestamp - interval '30 days') ;
```

Information

The trace information includes the search conditions and the knowledge data itself as search results, and is recorded in a common table within the database. Privileges are set so that only the database users and administrators who executed the search process can refer to, update, and delete that trace information.

3.12.3 Calculation of Search Accuracy Using External Tools

When calculating the search accuracy of subqueries using external tools, the search results of each subquery are obtained from trace information.
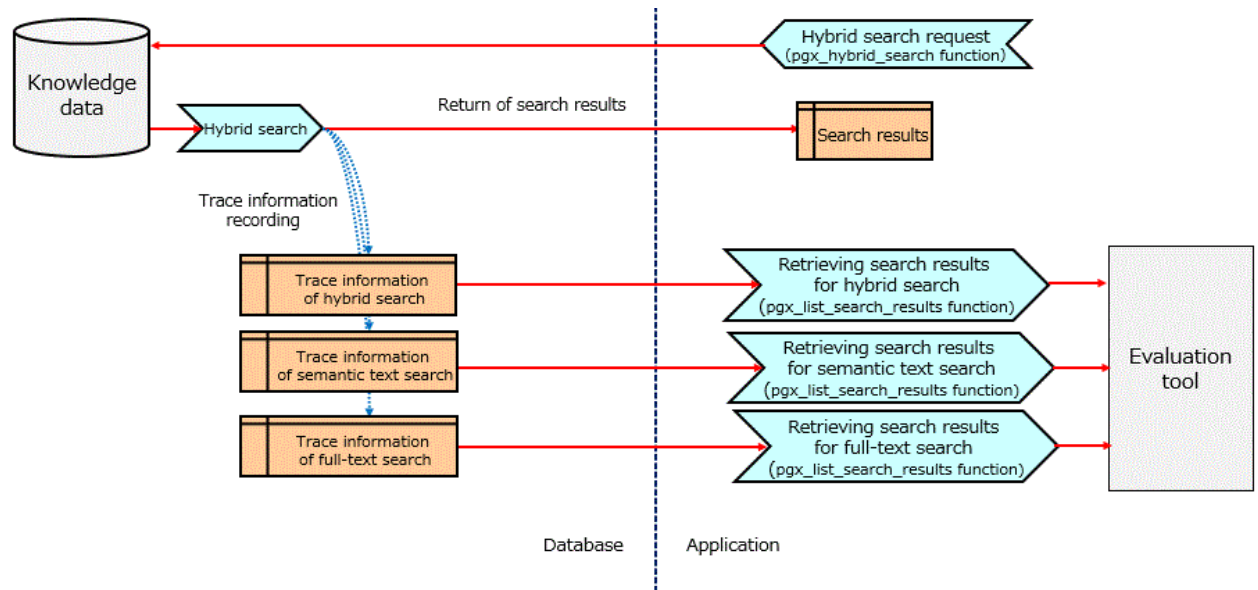
To retrieve the search results of a subquery, execute the pgx_list_search_results function by specifying the query ID and the type of subquery. When executed, a set of text chunks is returned in descending order of score, in the same format as the hybrid search results.

Example) Retrieve the search results of the full-text search subquery with query ID 10001

```
rag_database=> SELECT pgx_vectorizer.pgx_list_search_results(queryid => 10001,
subquery_type => 'fulltext');
```

As a result, by providing the search results of subqueries for evaluation queries and the correct answers corresponding to evaluation queries to the evaluation tool, the search accuracy of subqueries can be calculated.

3.12.4 Calculation of Search Accuracy in the Database

The calculation of search accuracy within a database consists of two major steps: "the calculation of evaluation values for each text chunk in the search results" and "the calculation of search accuracy based on those evaluation values".
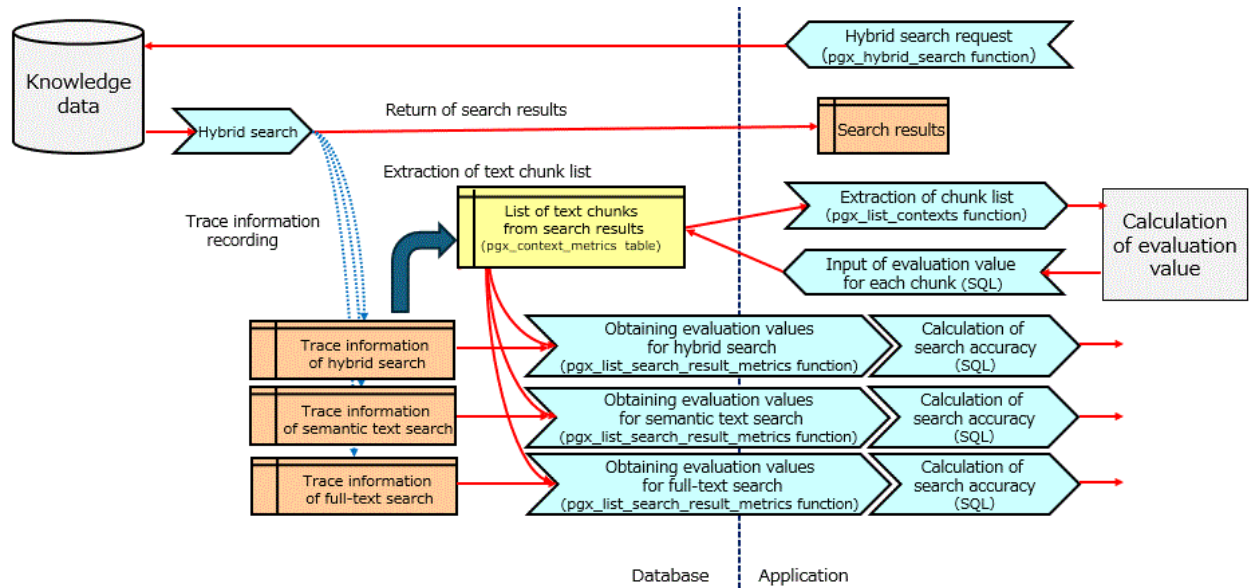
Calculation of evaluation values for each text chunk in the search results

Use the pgx_list_contexts function to extract the text chunks returned by the hybrid search executed and any of its constituent subqueries.

Example) Extract the text chunk returned by the hybrid search with query ID 10001 and any of the subqueries

```
rag_database=> SELECT pgx_vectorizer.pgx_list_contexts(queryid => 10001);
```

Match the extracted text chunks with the correct answers for the evaluation queries, calculate the evaluation values for each, and input them into the evaluation value table (pgx_context_metrics table). To input the evaluation values, use the UPDATE statement on the pgx_context_metrics table. For an example of using the UPDATE statement, refer to "3.12.4.1 Example of Calculating Search Accuracy in a Database". The evaluation values can be stored in JSON format to allow calculation of search accuracy using any metric.

Calculation of search accuracy based on evaluation value

The text chunk of the search result from the subquery and its evaluation value can be referenced using the pgx_list_search_result_metrics function. This can be used to calculate search accuracy.

Example) Refer to the search results of full-text search and their evaluation values

```
rag_database=> SELECT pgx_vectorizer.pgx_list_search_result_metrics(queryid => 10001,
subquery_type => 'fulltext');
```

3.12.4.1 Example of Calculating Search Accuracy in a Database

Explain an example of calculating search accuracy.

First, store the evaluation value in the pgx_context_metrics table. Here, input the evaluation value into the relevance_indicator in JSON format.

```
{
  "relevance_indicator": 1
}
```

The evaluation value is set to 1 if the context IDs of the search results are 'AAAA', 'BBBB', or 'DDDD', indicating relevance to the search criteria, and 0 if they are not relevant. In this case, an UPDATE statement is executed on the pgx_context_metrics table using queryid as the key, and the evaluation values for all text chunks are input collectively.

Example) Input of evaluation value using the UPDATE statement

```
rag_database=> UPDATE pgx_vectorizer.pgx_context_metrics
SET context_metrics = jsonb_set(
  context_metrics,
  '{relevance_indicator}',
  CASE
    WHEN context_id IN ('AAAA', 'BBBB', 'DDDD') THEN '1'::jsonb ELSE '0'::jsonb
  END
)
WHERE queryid = 10001;
```

Next, refer to the search results of subqueries that include evaluation values and calculate the search accuracy for each subquery.

Search accuracy is calculated using a search accuracy metric called precision@k. precision@k is a metric that represents the precision when focusing on the top k search results, and can be calculated using the following formula.

$$precision@k = \frac{The\ number\ of\ records\ related\ to\ the\ query\ among\ the\ top\ k\ search\ results\ returned}{k}$$

Below, the top 10 search precision (precision@10) for semantic text search is calculated using the relevance_indicator. The same can be calculated for full-text search.

```
rag_database=> SELECT
SUM( (context_metrics ->> 'relevance_indicator'):: integer )::float / COUNT(*) AS
precision_at_10
FROM(
  SELECT context_metrics FROM
  pgx_vectorizer.pgx_list_search_result_metrics(queryid => 10001, subquery_type =>
'semantic')
  ORDER BY score DESC LIMIT 10
);
```

3.12.5 Tuning of Hybrid Search

Tune hybrid search based on calculated search accuracy.

To reduce application modifications due to changes in hybrid search parameters, it is useful to create search functions for each application as user-defined functions.

Below is an example of defining a SQL function that only takes the string you want to search as input. By defining such a function, you can change the topN parameter and score_weight parameter on the database server side.

```
rag_database=> CREATE OR REPLACE FUNCTION mysearch(in text, in text, out uuid, out text,
out float8)
AS $$
SELECT context_id, chunk, score FROM pgx_vectorizer.pgx_hybrid_search(
jsonb_set(
  jsonb_set('{
    "target_view": "sample_embeddings",
    "search_fusion": "UNION",
    "topN": 20,
    "semantic": {
      "num_result": 10,
      "score_weight": 10
    },
    "fulltext": {
```

```
        "score_expression": "1"
    }}'::jsonb,
  '{semantic, search_text}', ('"' || $1 || '"')::jsonb),
'{fulltext, search_text}', ('"' || $2 || '"')::jsonb)
)
$$ LANGUAGE SQL
```

3.13 Reference

3.13.1 Vectorization Functions

3.13.1.1 Defining Vectorization

For the parameters to specify in the definition of vectorization,refer to the pgai documentation. If you want to perform vectorization within the database, you must specify pgx_vectorizer.schedule_vectorizer as the scheduling.

3.13.1.2 Vectorization Schedule

You can generate vector data from text data in the following ways.

- Manually start vectorization processing

- Perform vectorization processing periodically within the database

The timing of vectorization is determined by the schedule specified when defining vectorization. If schedule_none is specified, periodic vectorization will not be performed. To perform vectorization at a specified time, run the run_vectorize_worker function. If schedule_vectorizer is specified, periodic vectorization will be performed within the database.

Automatic vectorization can be disabled with the ai.disable_vectorizer_schedule function. It can also be re-enabled with the ai.enable_vectorizer_schedule function.

3.13.1.3 Vectorizer Management Functions

The following functions are provided to use for vectorization feature.

Function	Return type	Description
pgx_create_vectorizer([Arguments that can be specified with ai.create_vectorizer], fulltext_indexing => jsonb)	integer	It is used to automatically define full-text search indexes for vectorization and chunks. The ID of the created vectorizer will be returned. In addition to the arguments that can be specified for ai.create_vectorizer, specify the definition of the full-text search index (pgx_fulltext_indexing_gin() or pgx_fulltext_indexing_gist()). fulltext_indexing is optional when only defining vectorization and when using full-text search indexes other than GiST or GIN. If omitted, a full-text search index will not be created.
pgx_fulltext_indexing_gin(target_column text, fastupdate boolean, gin_pending_list_limit integer, opclass text, min_rows integer, create_when_queue_empty boolean)	jsonb	It is used when creating a full-text search index as GIN. The value specified for fulltext_indexing in

Function	Return type	Description
		pgx_create_vectorizer will be returned. target_column is used when using expression indexes or covering indexes. The column name for text chunks is chunk, and the default value for target_column is chunk. Specify the operator class to use in opclass. min_rows specifies the threshold for creating an index. The default is 100000. A full-text search index is created when the number of records in the embedded table exceeds this threshold. create_when_queue_empty is a parameter that specifies the timing for creating a full-text search index. The default is true, and the full-text search index is created after the vector conversion is completed. Other specifiable parameters refer to index storage parameters.
pgx_fulltext_indexing_gist(target_column text, fillfactor integer, buffering text, opclass text, min_rows integer, create_when_queue_empty boolean)	jsonb	It is used when creating a GIST as a full-text search index. The value specified for fulltext_indexing in pgx_create_vectorizer will be returned. Specify the operator class to use in opclass. The settings for target_column, min_rows, and create_when_queue_empty are the same as those of pgx_fulltext_indexing_gin. Other specifiable parameters refer to index storage parameters.
pgx_delete_fulltext_index_config(vectorizer_id integer)	void	It is used when deleting the vector transformation defined by pgx_create_vectorizer. Delete information related to full-text search of vectorization created by the pgx_create_vectorizer function.
get_vectorizer_id(view_name pg_catalog.pg_regclass)	integer	Returns the ID of the vectorization definition that corresponds to the specified embedded view.
schedule_vectorizer(schedule_interval interval)	json	Specify the interval for the vectorization process in schedule_interval. This function returns a JSON to be specified in the scheduling argument

Function	Return type	Description
		of the create_vectorizer function. If schedule_interval is not specified, 10 minutes will be specified.
alter_vectorizer_processing(vectorizer_id integer, batch_size integer, concurrency integer)	void	Changes the amount of data to be converted at one time and the worker multiplicity for the vectorization definition with the id specified in vectorizer_id.
alter_vectorizer_schedule(vectorizer_id integer, schedule_interval interval)	void	Changes the interval for the vectorization process for the vectorization definition with the id specified in vectorizer_id. If schedule_interval is not specified, 5 minutes will be specified.
run_vectorize_worker(vectorizer_id integer)	integer	Immediately starts the vectorization process for the vectorization definition with the ID specified in vectorizer_id, and starts the vectorization process in the background. Returns the PID of the started process.
start_vectorize_scheduler(void)	void	This will start the vectorize scheduler that connects to the database where this SQL function was executed. If the vectorize scheduler is already running, an error will occur.

See

For information about the arguments that can be specified with ai.create_vectorizer, refer to the pgai documentation.

3.13.2 Embedded Provider Management Functions

The following functions are provided to set and reference parameters of the embedded provider.

Function	Return type	Description
get_worker_setting(type text, param text)	text	Specify a combination of type and parameter (param) to get the value set for that parameter. Only the user who executed CREATE EXTENSION can execute this command.
set_worker_setting(type text, param text, value text)	void	Specify the combination of type and parameter (param), and set the value (value) for that parameter. Only the user who executed CREATE EXTENSION can execute this command.

The possible embedded provider names and parameters are:

type	param	Description
openai	OPENAI_API_KEY	OpenAI API key value
voyage	VOYAGE_API_KEY	VoyageAI API key value
ollama	OLLAMA_BASE_URL	Ollama API base url
user	VECTORIZE_USER	Username to connect to the database the worker that performs the vectorization.

3.13.3 Search Functions

3.13.3.1 Semantic Text Search

The following functions are provided for semantic text search.

Function	Return type	Description
pgx_similarity_search(view pg_catalog.regclass, query text, num_result integer defaut 5, distance_operator text default '<=>', OUT embedding_uuid uuid, OUT chunk text, OUT distance float8);	SETOF record	Searches the embedding view specified in view to obtain text similar to the text specified in query. You can display results up to the number specified in num_result. You can specify the distance calculation method using distance_operator.
pgx_similarity_search_checking_index(view pg_catalog.regclass, query text, num_result integer defaut 5, distance_operator text default '<=>', OUT embedding_uuid uuid, OUT chunk text, OUT distance float8);	SETOF record	An error occurs if the index operator defined in the embedding column of the table that references the view does not match the operator specified in distance_operator. Other than the above, it is the same as the pgx_similarity_search function.

3.13.3.2 Hybrid Search

The following functions are provided for hybrid search.

Function	Return type	Description
pgx_hybrid_search(query jsonb, OUT queryid bigint, OUT context_id uuid, OUT chunk text, OUT score float8)	SETOF record	Search the specified embedded view and retrieve the relevant text. For the return value, refer to "3.13.3.3 Details of the pgx_hybrid_search Function". For query details, refer to "Hybrid search results".
pgx_list_search_results(queryid bigint, subquery_type text)	SETOF record	If trace information for the specified query ID is recorded, it returns the search results. If

Function	Return type	Description
		subquery_type is not specified, it returns the results of a hybrid search. If subquery_type is semantic or fulltext, it returns the results of semantic text search or full-text search conducted internally, respectively.
pgx_list_search_result_metrics(queryid bigint, subquery_type text)	SETOF record	Returns search results along with the evaluation value of the text chunk. Other specifications follow the pgx_list_search_results function.
pgx_list_contexts(queryid bigint)	SETOF record	If trace information for the specified query ID is recorded, it extracts and returns a list of text chunks returned as search results in either hybrid search processing or subqueries executed internally. The extracted list of text chunks is inserted into the evaluation value table.
pgx_hybrid_search_trace_size()	bigint	Return the size of the trace information and evaluation value table in bytes.
pgx_grant_access_on_hybrid_search_trace(role name)	None	Grant the necessary access rights to the role for recording and evaluating trace information.
pgx_revoke_access_on_hybrid_search_trace(role name)	None	Revoke the access rights necessary for recording and evaluating trace information assigned to the role.

The pgx_list_search_results function returns a set of records of the following type, in descending order of score.

Column	Type	Description
queryid	bigint	Quiry ID for hybrid search feature.
context_id	uuid	Identifier of the returned text chunk. Unique within the embedded view.
chunk	text	Returned text chunk.
score	real	Hybrid search score. A higher score indicates a better match with the conditions.

The pgx_list_search_results_metrics function returns a set of records of the following type, in descending order of score.

Column	Type	Description
queryid	bigint	Quiry ID for hybrid search feature.
context_id	uuid	Identifier of the returned text chunk. Unique within the embedded view.

Column	Type	Description
chunk	text	Returned text chunk.
score	real	Hybrid search score. A higher score indicates a better match with the conditions.
metrics	jsonb	Evaluation value for each input text chunk.

The pgx_list_contexts function returns a set of records of the following type, in descending order of score.

Column	Type	Description
queryid	bigint	Quiry ID for hybrid search feature.
context_id	uuid	Identifier of the returned text chunk. Unique within the embedded view.
chunk	text	Returned text chunk.

3.13.3.3 Details of the pgx_hybrid_search Function

query argument

Specify search conditions in JSON format for the query argument. Specify the following for each key.

Key	JSON format	Deccription
target_view	string	Embedded view to be searched. Interpreted as pg_catalog.regclass. Cannot be omitted.
search_fusion	string	How to combine semantic text search and full-text search. The default is "UNION". "UNION": Returns the union of each search result. "INTERSECT": Returns the intersection of each search result. "TEXT_ONLY": Returns the intersection with all results of the full-text search. "VECTOR_ONLY": Returns the intersection with all results of the semantic text search. "MINUS_TEXT": Returns the results of the semantic text search excluding the intersection. "MINUS_VECTOR": Returns the results of the full-text search excluding the intersection.
rrf_k	number	Weighted RRF's weight. When a floating-point type is specified, the decimal part is truncated. The default is 60.
topN	number	Maximum number of results returned from hybrid search. When a floating-point type is specified, an error occurs. The default is 20.
semantic.search_text	string	Text to search with semantic text search. Cannot be omitted.
semantic.distance_operator	string	Distance operator in vector comparison.

Key	JSON format	Deccription
		The default is "<=>".
semantic.num_result	number	Maximum number of items to be retrieved by semantic text search. When a floating-point type is specified, an error occurs. If omitted, it is calculated based on topN. If the combination method is "UNION", it is half of topN, otherwise it is the same as topN.
semantic.score_weight	number	Relative weighting (importance) of semantic text search. When a floating-point type is specified, the decimal part is truncated. The default is 1.
fulltext.search_condition	string	Full-text search condition (an expression that returns a boolean value specified in the WHERE clause). The name of the column targeted for full-text search in the embedded view is chunk. Refer to the "3.3.3 Definition of Vectorization" for the definition of the embedded view. Cannot be omitted.
fulltext.num_result	number	Maximum number of items to be retrieved by full-text search. When a floating-point type is specified, an error occurs. If omitted, it is calculated in the same way as semantic.num_result.
fulltext.score_weight	number	Relative weight (importance) of full-text search. When a floating-point type is specified, the decimal part is truncated. The default is 1.
fulltext.score_expression	string	An expression that returns a floating-point score for full-text search. You can refer to tableoid and ctid. It cannot be omitted.

See

For the function used in calculating scores in full-text search, refer to "Full Text Search" in the PostgreSQL Documentation.

Information

You can specify a matching operator for full-text search as a condition for full-text search. Specify the operator supported by the full-text search index defined in the table to be searched. The search text for full-text search must be specified in the pattern required by the matching operator.

Example) Query argument of the pgx_hybrid_search function

```
{
  "target_view": "sample_embeddings",
  "search_fusion": "UNION",
```
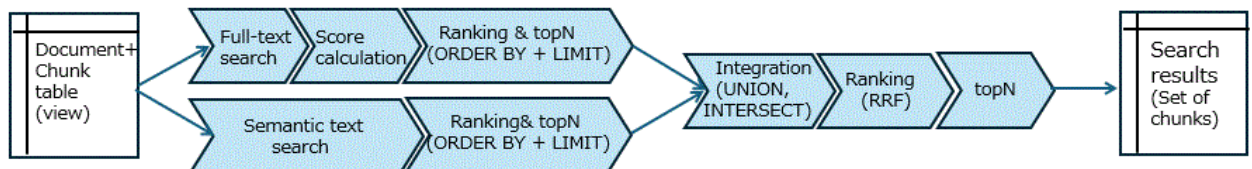
```
  "topN": 20,
  "semantic": {
  "search_text": "text for search",
  "distance_operator": "<=>",
  "num_result": 10,
  "score_weight": 1
  },
  "fulltext": {
    "search_condition": "chunk @@ websearch_to_tsquery('search text')",
    "num_result": 10,
    "score_weight": 1,
    "score_expression": "ts_rank_cd(to_tsvector(chunk), websearch_to_tsquery('search
text'))"
  }
}
```

The search conditions specified by the query argument

The search conditions specified by the query argument correspond to the following flow. (This flow is for explaining the meaning of the search process and is not the same as the actual access plan.)

- Full-text search: Perform a full-text search based on the conditions specified in fulltext.search_condition.

- Full-text search score calculation: Perform score calculation using fulltext.score_expression. If not specified, full-text search will result in an error.

- Full-text search ranking & topN: Rank the results in order of highest score. Only the number of search results specified by semantic.num_result from the top will be adopted, and the rest will be discarded.

- Semantic text search: Perform semantic text search based on vector similarity search using semantic.search_text and semantic.distance_operator.

- Semantic text search ranking & topN: Rank results in order of smallest distance from vector similarity search results. Only the number of search results specified by semantic.num_result from the top will be adopted, and the rest will be discarded.

- Integration: Integrate the results of full-text search and semantic text search according to the method specified in search_fusion.

- Ranking: Calculate scores based on the ranks of full-text search and semantic text search and assign final ranks. Use fulltext.score_weight, semantic.score_weight and rrf_k.

- topN: Return only the specified number of results from the top based on the final rank.

Score calculation

The score of the integrated result of multiple ranked results is calculated using a method called weighted RRF (Reciprocal Rank Fusion). Unweighted RRF considers the reciprocal of the rank of each search method as the score and takes the sum of multiple search methods as the final score. The final rank is determined based on that score. Weighted RRF calculates the sum of multiple search methods by multiplying each score by the weight for each search method as a coefficient.

The method for calculating the score WRRF(d) using weighted RRF for a certain search result d is shown below. i represents the search method, and the sum of scores for all search methods is obtained.

$$WRRF(d) = \sum_i w_i \times \frac{1}{k + rank_i(d)}$$

The value of k can be changed with the rrf_k parameter, and the default is 60. The smaller this parameter is, the greater the impact of differences in the original rank.

The rank (ranki(d)) for search results not returned by one of the search methods is considered a sufficiently large value (considered to have no impact on the final score).

Hybrid search results

The pgx_hybrid_search function returns a set of records of the following type.

Column	Type	Description
queryid	bigint	Query ID of the hybrid search feature used to refer to trace information. A valid value will be returned if trace information is recorded. Otherwise, 0 will be returned.
context_id	uuid	Identifier of the returned text chunk. Unique within the embedded view.
chunk	text	Returned text chunk.
score	real	Hybrid search score. A higher score indicates a better match with the conditions.

3.13.4 Tables/Views Created by Semantic Text Search and Automatic Vectorization Feature

For information about the tables and views that pgai creates, refer to the pgai documentation.

pgx_vectorizer creates the following tables.

pgx_vectorizer.worker_setting_table table

Stores information about the parameters used by the embedded provider.

Column	Type	Constraint	Description
type	text	PRIMARY KEY	The type of parameter to set. The name of the embedded provider or user
parameter	text	PRIMARY KEY	Parameter name
value	text	NOT NULL	Value to set for the parameter

pgx_vectorizer.pgx_fulltext_index table

The pgx_fulltext_index table stores the definition information of the full-text search index created by the pgx_create_vectorizer function.

Column	Type	Reference destination	Description
vectorizerid	integer	ai.vectorizer.id	Vectorization identifier
config	jsonb		Definition information of full-text search index.

Column	Type	Reference destination	Description
			Configurable values are the return values of the pgx_fulltext_indexing_gin function or the pgx_fulltext_indexing_gist function.

pgx_vectorizer.pgx_trace_query table

Provides information on hybrid search where trace information is recorded.

When a hybrid search is performed with tracing enabled, one record is inserted.

Deleting records from this table also deletes records from the pgx_trace_subquery table, pgx_trace_results table, and pgx_context_metrics table with the same queryid.

Column	Type	Description
queryid	bigint	Hybrid search ID
retrieval_time	timestamp with timezone	Time when hybrid search was conducted
retrieval_user	oid	Database user who performed the hybrid search
application_name	text	Application name that issued the hybrid search. The value of application_name specified
query	jsonb	Search criteria for hybrid search

pgx_vectorizer.pgx_trace_subquery table

Provides information about the subqueries for hybrid searches where trace information is recorded.

When a hybrid search is performed with tracing enabled, multiple records are inserted for each subquery.

Column	Type	Description
queryid	bigint	Hybrid search ID
subquery_seq	smallint	Serial number of subquery conducted within a hybrid search(A serial number closed in a certain hybrid search process)
subquery_type	text	Types of subqueries '-' : Overall hybrid search (main query) 'semantic' : Semantic text search 'fulltext' : Full-text search
subquery_text	text	Content of the executed subquery (SQL statement)

pgx_vectorizer.pgx_trace_results table

Provides information about the text chunks returned by each search process of hybrid search and subquery.

When performing a hybrid search with trace information enabled, multiple records corresponding to the text chunks of the search results of each subquery are inserted.

Column	Type	Description
queryid	bigint	Hybrid search ID

Column	Type	Description
subquery_seq	smallint	Serial number of subquery conducted within a hybrid search
context_id	uuid	Returned text chunk ID
chunk	text	Returned text chunk
rank	integer	Rank of text chunks within the same subquery type
score	real	Score of text chunk. Hybrid search overall : Score calculated by RRF Semantic text search : Score based on distance between vectors in text meaning search Full-text search : Score calculated by score_expression in hybrid search processing

In the case of text meaning search, the score is calculated using the distance d between the vector representations of the text chunk and the query text, according to the following formula.

$$score = \frac{1}{1 + d}$$

pgx_vectorizer.pgx_context_metrics table

This is a list of text chunks returned by either hybrid search or subquery. When the pgx_list_contexts function is executed, records for the specified query ID are inserted. You can enter any evaluation value in the context_metrics column.

Column	Type	Description
queryid	bigint	Hybrid search ID
context_id	uuid	Returned text chunk ID
context_metrics	jsonb	You can input evaluation values for text chunks.

pgx_vectorizer.pgx_trace_contexts view

Provide information about the text chunks returned by either hybrid search or subqueries.

For hybrid searches conducted with trace information enabled, there are records for each text chunk of search results for each subquery.

Column	Type	Description
queryid	bigint	Hybrid search ID
retrieval_time	timestamp with timezone	Time when hybrid search was conducted
retrieval_user	oid	Database user who performed the hybrid search
application_name	text	Application name that issued the hybrid search. The value of application_name specified
query	jsonb	Search criteria for hybrid search
subquery_type	text	Types of subqueries '-' : Overall hybrid search (main query) 'semantic' : Semantic text search 'fulltext' : Full-text search

Column	Type	Description
subquery_text	text	Content of the executed subquery (SQL statement)
context_id	uuid	Returned text chunk ID
chunk	text	Returned text chunk
rank	integer	Rank of text chunks within the same subquery type
score	real	Score of text chunk. Hybrid search overall : Score calculated by RRF Semantic text search : Score based on distance between vectors in text meaning search Full-text search : Score calculated by score_expression in hybrid search processing
context_metrics	jsonb	You can input evaluation values for text chunks.

3.13.5 Parameters

Describes the parameters to be set in the postgresql.conf file when using the semantic text search and automatic vectorization feature.

- pgx_vectorizer.max_vectorize_worker

  Specify the maximum number of workers that can run simultaneously within an instance and perform vectorization. The number of workers that perform vectorization is determined by the number of vectorization definitions created. This parameter can only be set at server startup. The default is 1. Because workers act as background workers, add the value set for this parameter plus the number of databases with the pgx_vectorizer feature enabled plus 2 to the max_worker_processes parameter, which specifies the maximum number of background workers. If the value set for max_worker_processes is insufficient, the instance cannot start.

- pgx_vectorizer.pgai_worker_path

  Specify the path to the program that performs vectorization processing. Specify *<Fujitsu Enterprise Postgres server feature installation directory>*/OSS/pgai-worker/bin/pgai. The default value is /opt/fsepv*<x>*server64/OSS/pgai-worker/bin/pgai (where "*<x>*" indicates the product version). You can reflect the changes by reloading the configuration file.

- pgx_vectorizer.enable_hybrid_search_trace(boolean)

  Enable trace information for hybrid search. The default is disabled. This parameter can be specified with the SET statement by any user, but additional access rights are required to record trace information. Also, even if this parameter is enabled on a hot standby server, trace information will not be recorded.

3.13.6 Evaluation of Knowledge Data Search

3.13.6.1 Concept of Evaluation for Knowledge Data Search

A knowledge database exists to provide users with appropriate knowledge. It is necessary to evaluate whether it is providing users with appropriate knowledge. For the evaluation of knowledge data (offline evaluation), a set of pairs of queries and results related to those queries (search results that should be returned for those queries) is prepared, and search accuracy is calculated using it.

Basic indicators in offline evaluation of search systems are recall and precision. A high recall state means that records related to the request are included in the search results without omission. A high precision state means that only records related to the request are included in the search results. Recall and precision can be calculated using the following formulas.

$$recall = \frac{The\ number\ of\ records\ related\ to\ the\ query\ in\ the\ returned\ search\ results}{The\ number\ of\ records\ related\ to\ the\ inquiry}$$

$$precision = \frac{The\ number\ of\ records\ related\ to\ the\ query\ in\ the\ returned\ search\ results}{The\ number\ of\ returned\ records}$$

Recall and precision are indicators that well represent whether necessary data is included or unnecessary data is not included. However, there are the following challenges in directly using recall and precision for evaluating knowledge data retrieval.

- Difficulty in defining query-related results

  When the same knowledge is contained in multiple records, returning even one record as a search result satisfies the user's request. Therefore, it is impossible to define a single set of search results that should be returned for that query.

- High cost of preparing query-related results

  Strictly defining query-related results requires determining whether each record in the database is related to the query or not.

To address the first issue, instead of defining the correct set of records to be returned, there is a method to define it by the content and meaning of the data. For example, there is a method to evaluate whether the sentences included in the records support the expected claim.

The preparation cost for the correct answer, which is the second issue, can also be reduced by the above approach. This is because the definition of the correct answer can be made from the content of the query or the content of the correct answer to the query, rather than from (all) records included in the database.

"Judging based on the content of the text" refers to things like the following.

When the statement "Mount Fuji is the highest peak in Japan" is considered correct, if a record contains the sentence "Mount Fuji is 3776 meters high and is recorded as the tallest mountain in Japan", this sentence can be judged to support the correct fact.

In the method of judging based on the content and meaning of the text, it is necessary to extract the statement of fact from the query or correct content and determine whether these statements are included in the search results. While it is certainly possible for humans to perform this task, there is also a method to have language models do it. By doing so, the preparation cost for correctness can be further reduced.

When adopting a method to evaluate search accuracy based on the content included in the records, it is important to note that the evaluated target is not only the implementation method of the search process. This method is a direct evaluation approach of the content obtained as search results. Therefore, not only the search method of the database but also what kind of knowledge is included in the database and how the user converts the content they want to know into queries for the database will affect the evaluation. When evaluating the accuracy of search methods with this approach, it is necessary to identify which part is causing the issue before starting to tune the search method.

3.13.6.2 Evaluation Value per Record and Search Accuracy

There are various metrics to represent search accuracy, but many calculations of search accuracy consist of two steps: obtaining evaluation values for individual records obtained as search results, and calculating search accuracy using the overall evaluation values of individual records (text chunks).
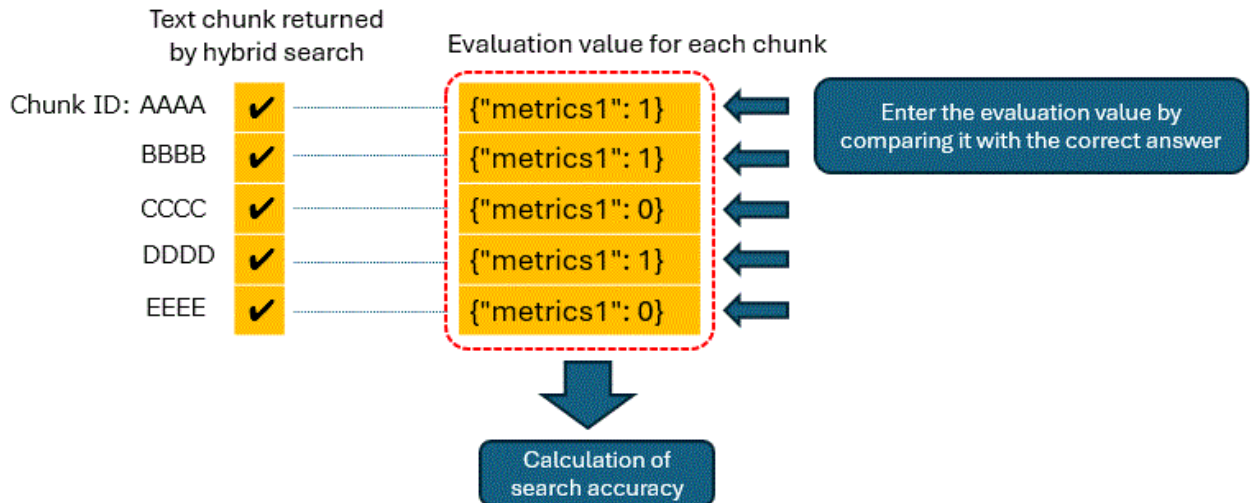
The search accuracy of subqueries can be calculated in the same way. The search accuracy can be evaluated in the same way for both full-text search subqueries and semantic text search. However, the text chunks returned by subqueries and the final hybrid search contain the same ones. Since the evaluation values for the same text chunks are common, these evaluation values can be determined and inputted together, and from there, only the evaluation values related to the text chunks for each subquery can be used to calculate the search accuracy of each subquery.
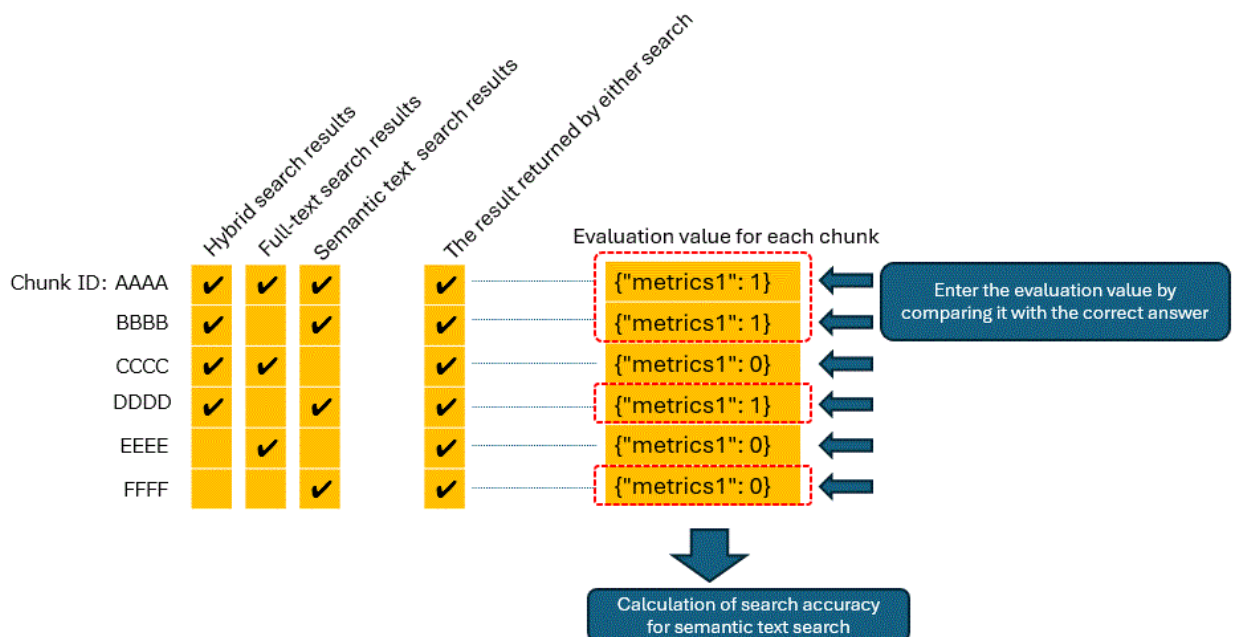
3.13.6.3 Tuning the Combination Method of Hybrid Search

Based on the search accuracy of each subquery, you can tune the combination method of subqueries. Adjustable parameters include, for example, the weighting between full-text search and semantic text search, and the number of retrievals each subquery returns. If the semantic text search has a higher precision compared to full-text search, it is possible to increase the weight of semantic text search used in the final ranking calculation. Additionally, by setting the number of retrievals (num_results) in subqueries to a larger value than usual to obtain trace information, and calculating search accuracy such as recall and precision for each number of retrievals, recall@k, and precision@k, you can determine the appropriate number of retrievals from subqueries. As a result of evaluating search accuracy, you may detect issues that cannot be improved by the combination method. In such cases, tuning of both semantic text search and full-text search is performed.

Chapter 4 Graph Management Feature

This chapter describes the graph management feature that provide graph storage and search.

Refer to the Apache AGE documentation for more details.

4.1 Overview of Graph Management Feature

The graph management feature enables you to store and search property graphs in Fujitsu Enterprise Postgres. A graph is a data structure that represents relationships between entities using nodes and edges that connect the nodes. A property graph is a type of graph in which nodes and edges can have information called properties. Graphs that represent relationships can be searched based on relationships and properties. Graph searches are performed using openCypher, a query language for graph databases.

4.2 Installation of Graph Management Feature

Graph management features are provided by the OSS's Apache AGE.

4.2.1 Setting Up the Graph Management Feature

Refer to "Apache AGE" in the Installation and Setup Guide for Server to set up Apache AGE.

4.2.2 Removing the Graph Management Feature

Refer to "Apache AGE" in the Installation and Setup Guide for Server to remove Apache AGE.

4.3 Creating a Graph

Graphs are stored in Fujitsu Enterprise Postgres as a single virtual data object called a graph. Internally, they are saved as multiple database objects under a schema that has one-to-one correspondence with the graph. Graphs are created and deleted using the create_graph and drop_graph functions, which are SQL functions provided by the graph management feature. If the second argument of the drop_graph function is set to true, the database object under the schema that corresponds to the graph will also be deleted.

Example) Creating a graph

```
SELECT create_graph('new_graph');
NOTICE:  graph "new_graph" has been created
 create_graph
-------------

(1 row)
```

Example) Deleting a graph

```
SELECT drop_graph('new_graph', true);
NOTICE: drop cascades to 2 other objects
DETAIL: drop cascades to table new_graph._ag_label_vertex
drop cascades to table new_graph._ag_label_edge
NOTICE: graph "new_graph" has been dropped
 drop_graph
-----------

(1 row)
```

Information

When you create a graph, a schema with the same name as the graph is created. Specify a name that does not overlap with existing schemas. In particular, you cannot use reserved names beginning with pg_. Also, do not use names beginning with pgx_.

The schema corresponding to a graph will be deleted when the graph is deleted. Do not create objects under this schema.

You can view a list of graphs and their corresponding schemas (namespaces) stored in a database with the graph management feature enabled by using the following method.

Example) Graph list

```
SELECT * FROM ag_catalog.ag_graph ;
 graphid |    name    | namespace
---------+------------+------------
   81957 | new_graph  | new_graph
   82576 | new_graph2 | new_graph2
   82598 | sample     | sample
(3 rows)
```

4.4 Storing Graph Data

Adding nodes and edges to a graph is done through Cypher queries using cypher functions, which are SQL functions provided by the graph management feature.

Example) Adding nodes and edges

```
SELECT * FROM cypher('new_graph', $$
 CREATE (:Person {name: 'Daedalus'})-[:FATHER_OF]->(:Person {name: 'Icarus'})
 $$) AS (a agtype);
 a
---
(0 rows)
```

In addition to the above, you can also load from a CSV file using the SQL functions load_labels_from_file and load_edges_from_file.

4.5 Protecting Graph Data

Graphs are automatically protected by backup, multiplexing, or replication settings that specify the tablespace, database, or instance in which the graph is saved.

This section describes the points to keep in mind when protecting graph data using Fujitsu Enterprise Postgres features.

Within Fujitsu Enterprise Postgres, graphs are stored as multiple objects under a schema that has one-to-one correspondence with the graph. To protect a graph, configure protection for the database objects that make up the graph.

4.5.1 Encrypting the Graph

A graph consists of multiple database objects, and even after the graph is created by the create_graph function, a new table will be created when a new label is added. If you want to encrypt multiple database objects that make up a graph, including those that will be created in the future, set the default tablespace of the entire database that stores the graph to an encrypted tablespace.

Example) When encrypting the entire database for which this feature is enabled

```
CREATE DATABASE rag_database TABLESPACE = encrypted_tablespace;
CREATE EXTENSION age;
```

To encrypt a graph when you cannot use a single tablespace per database, temporarily change the default tablespace to an encrypted tablespace before the operation that creates the graph.

Example) Specifying table space and creating a graph

```
SET default_tablespace = 'secure_tablespace';
SELECT create_graph('graph1');
```

4.5.2 Restricting Access to Graph

Access to a graph is restricted by access control for the database objects that make up the graph. Access restrictions are set using the confidentiality management feature of Fujitsu Enterprise Postgres. Since graphs have a one-to-one correspondence with schema objects, you can allow or deny access to a graph by specifying access rights for that schema using the confidentiality management feature as follows:

1. Refer to the "Confidentiality Management" in the Security Operations Guide to define confidentiality management role, confidentiality matrix, confidentiality level, and confidentiality group.

2. Grant confidentiality privilege on the schema to the confidentiality group.

```
SELECT pgx_grant_confidential_privilege('rag_matrix', 'level1', 'group1', '{"schema":
["USAGE"]}');
```

3. Add the schema corresponding to the graph as a confidentiality object to the confidentiality level.

```
SELECT pgx_add_object_to_confidential_level ('rag_matrix', 'level1',
'[{
    "type":"schema",
    "object":[
        {
            "schema":"new_graph"
        }
    ]
}]');
```

4. Add roles to the confidentiality group you created to set access rights to the graph.

```
SELECT pgx_add_role_to_confidential_group('rag_matrix', 'group1', '["rag_user"]');
```

If you want to set fine-grained access privileges, such as allowing only searches of graphs but not updating them, you can use SQL statements to directly set access privileges for database objects such as the tables that make up the graph.

The privileges required to access a graph are as follows:

- Privileges required to create a graph

    - CREATE privilege for the database

- Privileges required to create and delete new nodes and edges

    - CREATE and USAGE privilege for the schema with the same name as the graph name

    - When adding a node, ownership of the _ag_label_vertex table under the schema with the same name as the graph name

    - When adding an edge, ownership of the _ag_label_edge table under the schema with the same name as the graph name

    - UPDATE privilege for _label_id_seq under the schema with the same name as the graph name

- Privileges required to create and delete nodes and edges that use existing labels

    - USAGE privilege for the schema with the same name as the graph name

- When adding a node, privileges under the schema with the same name as the graph name

    - INSERT privilege (to create), SELECT privilege, and UPDATE privilege (to delete) for the _ag_label_vertex table or a table with the same name as the label name to be added

    - USAGE privilege for the _ag_label_vertex_id_seq sequence or a sequence with the same name as the label name to be added

- Privileges when adding an edge

    - INSERT (create), SELECT, and UPDATE (delete) privileges for the _ag_label_edge table under the schema with the same name as the graph name or the table with the same name as the label name to be added.

    - USAGE privilege for the _ag_label_edge_id_seq sequence or the sequence with the same name as the label name to be added.

- Privileges required to search graphs.

    - USAGE privilege for the ag_catalog schema.

    - USAGE privilege for the schema with the same name as the graph name.

    - SELECT privilege for the table object in the schema with the same name as the graph name.

📖 Information
....

Graph data structures do not have the concept of rows and columns, so PostgreSQL's row-level security and column-based access control features cannot be applied.
....

4.5.3 Recording Access to Graph

The audit log feature of Fujitsu Enterprise Postgres is used to record access to the graph in the audit log. When specifying database objects to be audited individually, specify each database object that constitutes the graph.

When adding new labels to a graph, new corresponding tables are created, so those tables must also be specified as targets for auditing.

4.6 Searching Graph

Graph searches are performed using Cypher queries. Cypher queries are specified as strings as arguments to the cypher function, which is an SQL function.

For more information about Cypher queries, refer to the Apache AGE documentation.

Example) Graph search

```
SELECT * FROM cypher('new_graph', $$
 MATCH (:Person {name: 'Daedalus'})-[:FATHER_OF]->(person)
 WITH person.name AS name ORDER BY person.name RETURN name
$$) AS (v agtype);
    v
----------
 "Icarus"
(1 row)
```

Graph searches allow you to retrieve specific property values or sets of properties for nodes or edges that match a condition. Properties are returned in agType data type added by the graph management feature, which is compatible with JSON types. Graph searches are closed to cypher functions, but the retrieved values can also be combined with searches of other tables in the database.

Example) Combination

```
SELECT (x::json->'properties')->'name' FROM cypher('new_graph', $$
MATCH (x)
```

```
RETURN x $$) AS (x agtype);
   ?column?
------------
 "Daedalus"
 "Icarus"
(2 row)
```

4.7 Adding Labels to Graph

Labels in a graph represent the type of node or edge, and are an important element in expressing the schema of knowledge data stored as a graph. Adding a label is easier than adding a column to a table, but you should plan carefully and consider the impact on your application. By default, only the database user who created the graph can add new labels.

When adding nodes or edges with new labels to a graph, new tables corresponding to each label are created. To encrypt a graph, change the default tablespace to an encrypted tablespace before adding labels. The access rights for these newly created tables are inherited from the access rights for the tables for unlabeled nodes or unlabeled edges, so no additional steps are required if you want to reuse those access rights.

You can check the tablespaces in which tables corresponding to each label of the graph nodes and edges are located with the following SQL.

```
SELECT g2.graphname AS graphname, c.relname AS relname, c.reltablespace AS reltablespace
FROM pg_class c,
rag_databse->  (SELECT g.name AS graphname, l.relation AS relation FROM
ag_catalog.ag_graph AS g, ag_catalog.ag_label AS l WHERE g.graphid = l.graph) AS g2 WHERE
c.oid = g2.relation;
 graphname  |      relname     | reltablespace
------------+------------------+---------------
 new_graph  | _ag_label_vertex |            0
 new_graph  | _ag_label_edge   |            0
 new_graph  | x                |            0
 new_graph  | y                |            0
 new_graph2 | _ag_label_vertex |            0
 new_graph2 | _ag_label_edge   |            0
 sample     | _ag_label_vertex |        80722
 sample     | _ag_label_edge   |        80722
(8 rows)
```

4.8 Performance Tuning of Graph Search

Graph data is stored internally as multiple tables. Graph data searches are implemented as SELECT statements on those tables. The access plan can be checked with the EXPLAIN statement in the same way as for normal SQL statements.

Graph data consists of a table with a record for each node and a table with a record for each edge, and queries corresponding to graph data are implemented by joining and filtering those tables. Therefore, the approach to performance issues in graph searches is the same as for SELECT statements that include joins.

Example) Checking the access plan of a graph search

```
SELECT * FROM cypher('new_graph', $$
EXPLAIN (COSTS off) MATCH (:Person {name: 'Daedalus'})-[:FATHER_OF]->(person)
WITH person.name AS name ORDER BY person.name RETURN name
$$) AS (v agtype);
 QUERY PLAN
---------------------------------------------------------------------------------------------
----------------------------------------------------------------------
 Sort
 Sort Key: (agtype_access_operator(VARIADIC ARRAY[_agtype_build_vertex(person.id,
_label_name('25995'::oid, person.id), person.properties), '"name"'::agtype]))
 -> Hash Join
 Hash Cond: (person.id = _age_default_alias_1.end_id)
 -> Append
```

```
-> Seq Scan on _ag_label_vertex person_1
-> Seq Scan on "Person" person_2
-> Hash
-> Hash Join
Hash Cond: (_age_default_alias_1.start_id = _age_default_alias_0.id)
-> Seq Scan on "FATHER_OF" _age_default_alias_1
-> Hash
-> Seq Scan on "Person" _age_default_alias_0
Filter: (properties @> '{"name": "Daedalus"}'::agtype)
(14 rows)
```

4.9 Using Graph Data in Applications

The cypher function returns a value of the agType data type. The agType type is a subset of the json type. If you want to handle this data type directly in your application, please install a driver for each language in your application. If you do not use a driver, the agType type will be returned to your application as a string or JSON type.

See

For driver installation, refer to below.

https://github.com/apache/age/tree/master/drivers

4.10 Visualizing Graph Data

You can use the OSS age-viewer as a tool to visualize graph data.

See

For age-viewer installation, refer to below.

https://github.com/apache/age?tab=readme-ov-file#graph-visualization-tool-for-age

4.11 Internal Structure of Graph Data

When you create a graph, a corresponding schema is created, and tables, indexes, and sequence objects are created under it. For more information, see the Apache AGE documentation.

4.12 Quantitative Limits

The total number of nodes and edges must be less than or equal to $2^{48} - 1$. This restriction applies only to one particular graph.

4.13 Reference

The following SQL functions are provided, refer to Apache AGE documentation for details:

Function	Description
create_graph	Creating a graph
drop_graph	Deleting a graph
cypher	Manipulating graphs with Cypher queries
load_labels_from_file and load_edges_from_file	Loading the graph data