FUJITSU

Fujitsu
Enterprise Postgres 17 SP1

Knowledge Data Management Feature User's Guide

Linux

J2UL-3007-01PEZ0(00)
March 2025

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 1.0: March 2025

Copyright

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

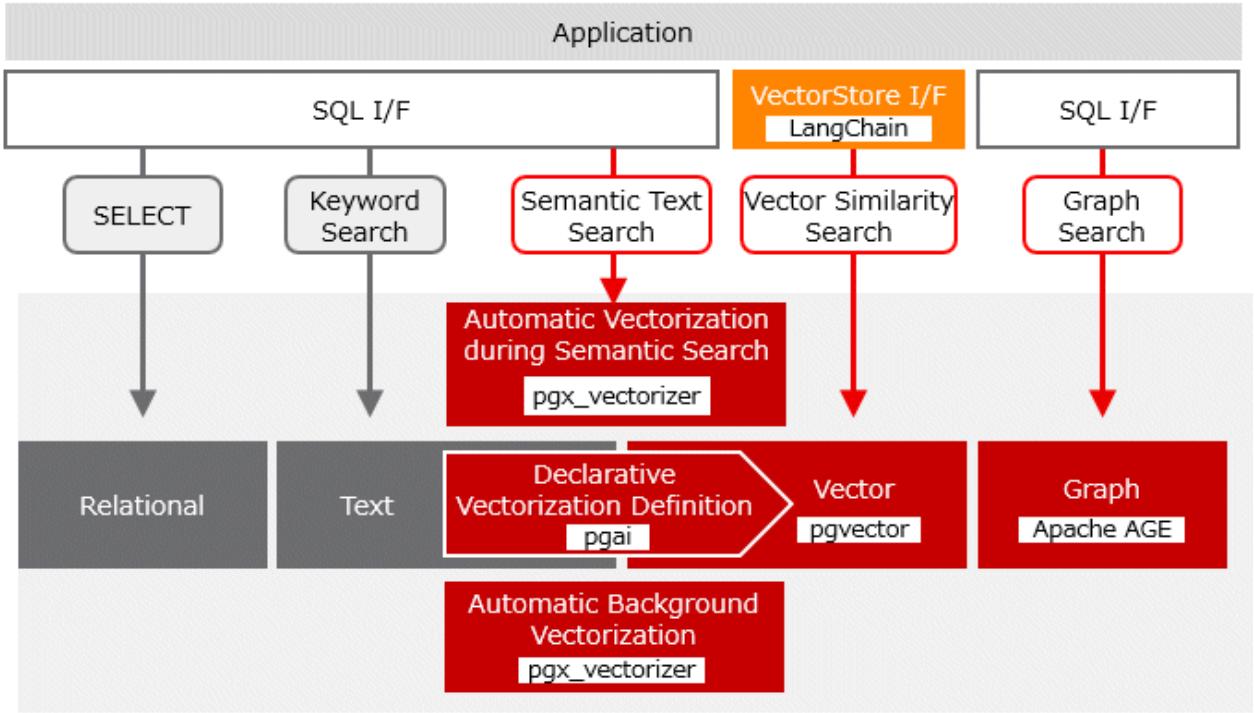
  Searches for semantic similarity between the specified text and the text stored in Fujitsu Enterprise Postgres using embedding in a semantic vector.

  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.

- 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
- 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
- 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

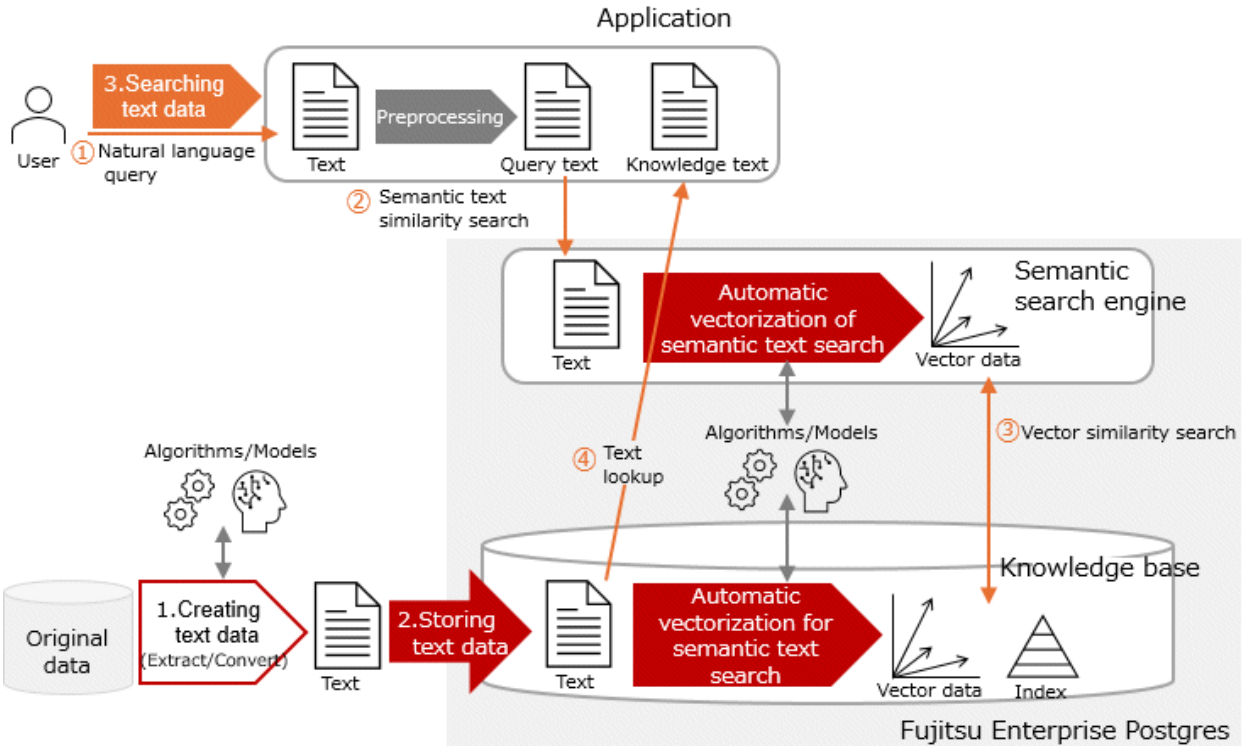
   Use different algorithms and models to extract text from different data in your organization.

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

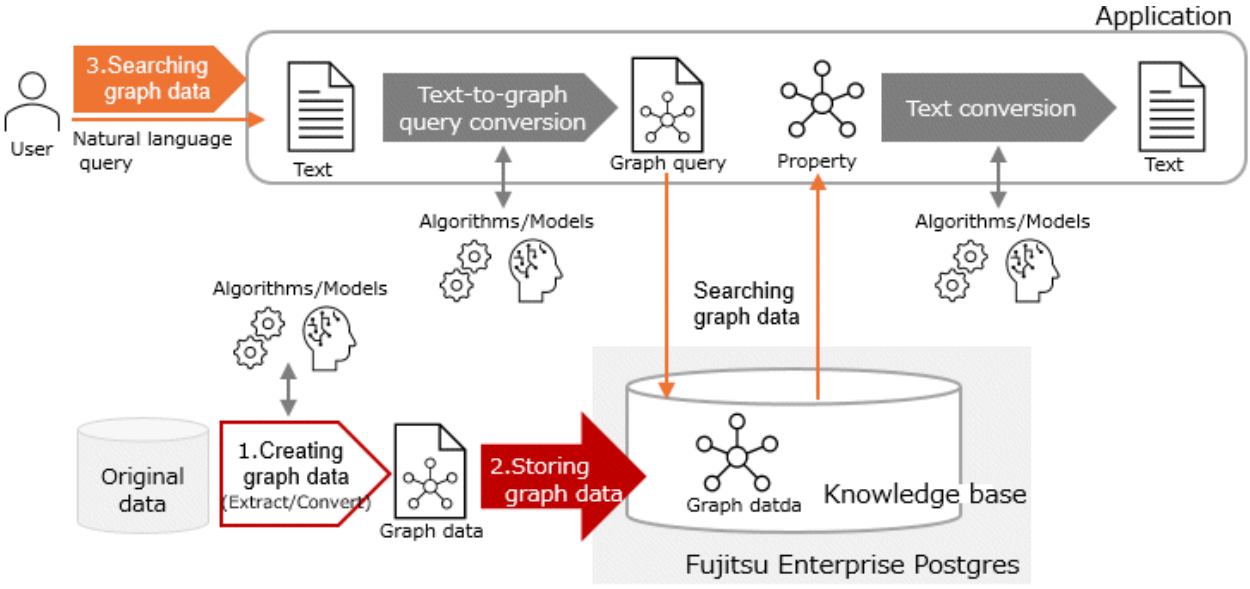
   Extract graph data from a variety of data in your organization, such as using a relation extraction model.

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 seach of vector data.

2.1 Setting Up the Vector Data Management Feature

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

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

2.2 Storing and Seaching 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.

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.

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

```
EXPLAIN SELECT * FROM items ORDER BY embedding <=> '[3,1,2]' LIMIT 5;
                                  QUERY PLAN
------------------------------------------------------------------------------------
Limit  (cost=16.60..16.81 rows=5 width=33)
   ->  Index Scan using idx_hnsw on items  (cost=16.60..437.60 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 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 search.

3.1 Overview of the Semantic Text Search and Automatic Vectorization Feature

The semantic text search and automatic vectorization feature utilizes a vector representation that preserves the semantic similarity between text data for the semantic similarity search of text. When you add text data, the worker process automatically generates the vector data in the background according to a predefined vectorization definition (vectorizer) and stores it in a table (an embedded table) containing the vector data corresponding to the table containing the text data.

When you define a vectorization of text data, a view (an embedded view) that combines text data with a column of vector data is also created automatically.

When you perform a semantic text search on an embedded view, a vector representation of the input text is generated internally, and a semantic text search is performed as a similarity search between the vector 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/
```

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.

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

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.

3.2.3 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');
```

Point

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

A vector expression is defined by using the SQL function create_vectorizer to define a set of parameters for vectorization called a vectorizer.

Example) Definition of vectorization

```
rag_database=> SELECT ai.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')
);
create_vectorizer
-------------------
                  1 - The ID of the created vectorizer
(1 row)
```

In the vectorization definition, you can specify information about the table that contains the text data to be vectorized, the embedding model and vector length that are directly related to vector representation, preprocessing to be performed before vectorization, and other information, as well as specify the timing of vectorization as a schedule.To perform automatic background vectorization within Fujitsu Enterprise Postgres, specify pgx_vectorizer.schedule_vectorizer for the scheduling argument.

Point

Do not change the name, primary key, column names, or other information of the table that contains the text to be converted into vectors, as this will cause the vectorization process to not work properly.

You can view the vectorization definition you created in the ai.vectorizer table.

```
SELECT * FROM ai.vectorizer where view_name = 'sample_embeddings';
id            | 1
source_schema | public
source_table  | sample_table
source_pk     | [{"pknum": 1, "attnum": 1, "attname": "id", "typname": "int4"}]
target_schema | public
target_table  | sample_embeddings_store
view_schema   | public
view_name     | sample_embeddings
trigger_name  | _vectorizer_src_trg_1
queue_schema  | ai
queue_table   | _vectorizer_q_1
config       | {"version": "0.8.0", "chunking": {"chunk_size": 800, "separators": ["\n\n", "\n", ".",
"?", "!", " ", ""], "config_type": "chunking", "chunk_column": "contents", "chunk_overlap": 400,
"implementation": "recursive_character_text_splitter", "is_separator_regex": false}, "indexing":
{"config_type": "indexing", "implementation": "none"}, "embedding": {"model": "all-minilm",
"dimensions": 384, "config_type": "embedding", "implementation": "ollama"}, "formatting":
{"template": "$chunk", "config_type": "formatting", "implementation": "python_template"},
"processing": {"batch_size": 2000, "concurrency": 1, "config_type": "processing", "implementation":
"default"}, "scheduling": {"config_type": "scheduling", "implementation": "none",
"schedule_interval": "01:00:00", "extra_implementation": "pgx_vectorizer"}}
disabled      | f
```

3.3.4 Granting Privilege to Execute Functions

When executing the pgx_similarity_search function, 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 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 "3.11.6 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_embedding_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_embedding_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_embedding_store, which contains vector data generated by this function. In this example, sample and sample_embedding_store contain user names in a column called username, and sample_embedding_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 "3.11.6 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 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 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 ai.create_vectorizer(…);
```

3.10 Performance Tuning of Semantic Text Search

A text semantic 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 text semantic search. You can check whether an index is being used in a text semantic 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 search. If the distance type specified in the semantic 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 Reference

3.11.1 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.

3.11.2 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.

3.11.3 Vectorizer Management Functions

For information about the vectorizer management functions provided by pgai, please refer to the pgai documentation.

The automatic vectorization feature for semantic text search provides the following functions in addition to the vectorizer management functions provided by pgai.

Function	Return type	Description
get_vectorizer_id(view_name pg_catalog.pg_regclass)	int	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 of the create_vectorizer function. If schedule_interval is not specified, 10 minutes will be specified.
alter_vectorizer_processing(vectorizer_id int, batch_size int, concurrency int)	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 int, 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 int)	int(worker pid)	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.

3.11.4 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

Function	Return type	Description
		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.

3.11.5 Semantic Search Functions

It provides the following functions 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.

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

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

pgx_vectorizer creates the following tables.

pgx_vectorizer.worker_setting_table

Column	Type	Constraint	Description
type	text	PRIMARY KEY	The type of parameter to set.

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

3.11.7 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 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.

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